Extensional Reasoning with Complete View Definitions and Incomplete Data: a Transformation-based Approach

Michael Genesereth[1,2,†], Manav Kant[1,2,*,†]

[1]Stanford University, 450 Jane Stanford Way, Stanford, USA
[2]CodeX – The Stanford Center for Legal Informatics, 559 Nathan Abbott Way, Stanford, USA

Abstract

Logical database languages, such as Datalog and Epilog, divide relations into two types—base relations (i.e., the dataset), which are defined extensionally, and view relations (i.e., the program), which are defined by rules. Whereas some logical database applications assume that the definitions of all relations are complete, others must contend with incompleteness in the base relations, the view relations, or both. Motivated by computable contracts, a key industry application of logic programming, we concentrate on the special case of reasoning with complete view definitions and incomplete datasets. The naive approach to evaluating queries on incomplete datasets requires exponential time complexity in the size of the data. In this paper, we introduce a novel transformation-based approach to this task that is broadly applicable across many classes of programs. For UCQs (Unions of Conjunctive Queries), a class of programs to which any non-recursive pure Datalog program can be unfolded, we formalize this approach, prove its correctness and show that it runs in polynomial time.

Keywords

Logic programming, computable contracts, reasoning under uncertainty

1. Introduction

Logical database languages, such as Datalog and Epilog, divide relations into two types—base relations (i.e., the dataset) and view relations (i.e., the program). Base relations are defined extensionally—each base relation is encoded as a collection of instances of the relation. View relations are defined by rules that specify extensions of the base relations.

Some logical database applications assume that the definitions of all relations are complete. The sets of factoids defining base relations include all true instances and only true instances. The rules defining view relations give complete sets of view instances for every complete set of base instances to which they are applied. Other applications must contend with incompleteness in the base relations, the view relations, or both. There is substantial logic programming literature concerned with the management of incompleteness in view definitions, e.g., disjunctive logic programs [1], existential logic programs, answer set programming [2], non-stratified programs (e.g., the stable model [3] and well-founded [4] semantics), and so forth.

In this paper, motivated by the key industry application of computable contracts (see Section 2), we concentrate on the novel direction of reasoning efficiently with *complete view definitions* and *incomplete definitions of base relations* in logic programming. In the databases literature, this topic has received substantial attention in the form of view-based query answering and rewriting [5, 6, 7] and answering queries on tables with nulls [8, 9, 10]. It is worth noting that our formulation of incomplete datasets (described in Section 3.4) differs fundamentally from tables with nulls, possessing some advantages in expressivity and query complexity that we describe in Section 4.

In what follows, we first motivate this theoretical work through the practical application area of computable contracts. We then formalize the syntax and semantics of extensions (sets of deduced facts)

Datalog 2.0 2026: 6th International Workshop on the Resurgence of Datalog in Academia and Industry, September 7, 2026, Klagenfurt, Austria
*Corresponding author.
†These authors contributed equally.
genesereth@stanford.edu (M. Genesereth); manavk@stanford.edu (M. Kant)
0000-0001-9124-7487 (M. Genesereth)

for complete view definitions on complete datasets and extend this to positive *and negative* extensions on incomplete datasets. Then, we introduce a novel transformation-based algorithm for computing answers for UCQs (Unions of Conjunctive Queries—see Definition 5.2), a class of programs to which any non-recursive pure Datalog program can be unfolded, on incomplete datasets and prove its correctness. Whereas the naive approach to this task has exponential data complexity, we show that this algorithm, which is also applicable to broader classes of programs (with some augmentation) has merely *polynomial data complexity*.

2. Motivation

Computable contracts are legal contracts specified with enough detail that questions about compliance in well-defined circumstances admit unambiguous answers, thus supporting automated execution and compliance checking [11]. The utility of these contracts, and of logic programming in implementing them, has been recognized across industries such as insurance [11], permitting [12] and manufacturing [13].

In order for many important questions (e.g., "Is beneficiary A covered for service B?", "Did contractor X correctly source material Y?") to be answered unambiguously, complete view definitions (i.e., rules that give all true answers and only true answers for any complete set of base facts to which they are applied) are necessary. However, complete *datasets* cannot be expected. In health insurance, a beneficiary may not know all relevant details of their medical history, and in manufacturing, negotiating parties may not feel comfortable revealing all sensitive supplier data [13]. Furthermore, due to the potential for massive datasets in the aforementioned fields, *efficient* reasoning (with respect to the data) is needed.

Our experiences designing and implementing computable contracts across various domains have motivated our work on deriving an efficient algorithm for reasoning with complete view definitions and incomplete datasets. We give the following case as a running example of the practical applicability of our theoretical results.

Example 2.1. Consider a universe (defined by the relation `u`) of 5 companies that constitute the manufacturing supply chain in a regulated industry.

```
u(machines_inc)
u(parts_inc)
u(raw_materials_inc)
u(suspicious_inc)
u(evil_inc)
```

Let `s` be a base relation where `s(X, Y)` means `X` is a supplier of `Y`. Consider, then, the following dataset.

```
s(parts_inc, machines_inc)
s(raw_materials_inc, parts_inc)
s(evil_inc, suspicious_inc)
```

We consider `X` to be a *tier 1* supplier of `Y` if `X` is a supplier of `Y`. We consider `X` to be a *tier 2* supplier of `Y` if `X` is a supplier of a supplier of `Y`. The view relation `t`, where `t(X, Y)` means `X` is a tier 1 or 2 supplier (hereafter referred to simply as a "tiered supplier") of `Y`, can then be defined as follows.

```
t(X, Y) :- s(X, Y)
t(X, Y) :- s(X, Z) & s(Z, Y)
```

Applying these rules to the dataset, we can conclude exactly

```
t(parts_inc, machines_inc)
t(raw_materials_inc, parts_inc)
t(evil_inc, suspicious_inc)
t(raw_materials_inc, machines_inc)
```

Now, say that `evil_inc` is on a list of banned corporations. A company considering a partnership with `machines_inc` may stipulate in their contract that `evil_inc` must *not* be one of their tiered suppliers. Assuming completeness in both the data and the view relations, the dataset facts and the conclusions we deduce from them are the *only* true facts—all others are false. Since `t(evil_inc, machines_inc)` was not concluded, the partnering company can be secure in the knowledge that `evil_inc` is not a tiered supplier of `machines_inc`, and thus the contract is being complied with.

But what if we no longer have completeness in the data, perhaps because the companies were uncomfortable sharing their complete supplier lists? Then, while the dataset facts and the conclusions we deduce from them are certainly true, the other facts are not necessarily false. They are simply not known to be true. It is now *possible* that `evil_inc` is a tiered supplier of `machines_inc` (e.g., `suspicious_inc` may be a direct supplier of `machines_inc`). In this setting, consider including a set of negative dataset facts $N = \{$`s(evil_inc, machines_inc)`, `s(evil_inc, parts_inc)`, `s(suspicious_inc, machines_inc)`, `s(raw_materials_inc, machines_inc)`$\}$, which we *know to be false* (the companies may have been more comfortable sharing information about who is *not* their supplier). Despite still not knowing the truth values of many of the supplier facts, knowing that `evil_inc` is a direct supplier neither of `machines_inc` nor of any of its possible direct suppliers allows us to affirmatively conclude that `evil_inc` is not a tiered supplier of `machines_inc`. Our transformation-based algorithm (see Example 5.1 for the transformed tiered supplier relation) allows us to efficiently do this.

3. Background

3.1. Datasets

A *vocabulary* is a collection of *object constants* and *relation constants*, together with a specification of the arities (i.e., the number of arguments allowed in any expression involving the constant) associated with all relation constants. In what follows, we write constants as strings of alphanumeric characters beginning with a number or a lower case letter, e.g., `a`, `b`, `23`, `p`, `q`, `r`. We assume that the set of object constants is finite and, as such, extensionally define a special unary *universe* relation, `u`, which is true of all object constants in the vocabulary. We denote the set of objects in the vocabulary as U.

A *factoid* (or *ground atom*) is an expression formed from an n-ary relation constant and n object constants. We write factoids in traditional mathematical notation—the relation constant followed by its *arguments* enclosed in parentheses and separated by commas. The *Herbrand base* for a vocabulary is the set of all factoids that can be formed from the constants in the vocabulary. For example, for a vocabulary with just two object constants `a` and `b` and a single binary relation constant `r`, the Herbrand base is $\{$`r(a,a)`, `r(a,b)`, `r(b,a)`, `r(b,b)`$\}$. A *dataset* is any subset of the Herbrand base, i.e., an arbitrary set of the factoids that can be formed from the vocabulary, where all of the factoids in the dataset are assumed to be true.

3.2. View Definitions

A *logic program* is a set of rules that define new relations in terms of existing relations. In this paper, we write view definitions in the form of Prolog-like rules.

The vocabulary of a logic program is a superset of the vocabulary of any dataset to which it is applied. It includes the object and relation constants used in the dataset plus all additional object and relation constants used in the rules.

Logic programs can also include *variables*, written as strings of alphanumeric characters beginning with a capital letter (e.g., `X`, `Y`, `Speed`), which allow us to state relationships among objects without naming specific objects.

Atoms are analogous to dataset factoids except that they can optionally contain variables as well as object constants. A *literal* is either an atom or a negation of an atom (i.e., an expression stating that the atom is false). A simple atom is called a *positive* literal and the negation of an atom is called a *negative*

literal. In what follows, we write negative literals using the negation sign $\sim$. For example, if `p(a,b)` is an atom, then $\sim$`p(a,b)` denotes the negation of this atom.

A *rule*, such as `r(X) :- p(X,Y) & ~q(Y)`, is an expression consisting of a distinguished atom, called the *head*, and zero or more literals, together called the *body*. Here, `r(X)` is the head, the expression `p(X,Y) & ~q(Y)` is the body; and `p(X,Y)` and $\sim$`q(Y)` are *subgoals*.

A rule is essentially a reverse implication where variables in the body that do not appear in the head are existentially quantified in the antecedent. For example, the rule above states that `r` is true of any object `X` *if* there exists an object `Y` such that `p` is true of `X`, `Y` and `q` is not true of `Y`. If we know that `p(a,b)` is true and `q(b)` is false, then, using this rule, we can conclude that `r(a)` is true. A *logic program* is a set of facts and rules of the form just described.

To eliminate problems such as ambiguities and potentially infinite answer sets, we concentrate exclusively on logic programs where the rules have two special properties, viz. safety and stratification. A rule in a logic program is *safe* if and only if every variable that appears in the head also appears in at least one positive literal in the body and every variable that appears in a negative literal in the body appears in a *preceding* positive literal in the body. A logic program is safe if and only if every rule in the program is safe.

We say that a set of view definitions is *stratified with respect to negation* if and only if its rules can be partitioned into *strata* in such a way that (1) every stratum contains at least one rule, (2) the rules defining relations that appear in positive subgoals of a rule appear in the same stratum as that rule *or* in some lower stratum, and (3) the rules defining relations that appear in negative subgoals of a rule occur in some *lower* stratum (not the same stratum).

3.3. Extensions on Datasets

An *instance* of an expression (atom, literal, or rule) is one in which all variables have been consistently replaced by ground terms. Given a rule ρ and a dataset Δ, the *application* of ρ to Δ (written $A(\rho, \Delta)$) is the set of all ψ such that (1) ψ is the head of an arbitrary instance of ρ, (2) every positive subgoal in the instance is a member of Δ, and (3) no negative subgoal in the instance is in Δ.

The *closure* of a logic program Ω on a dataset Δ (written $C(\Omega, \Delta)$) is the result of repeatedly and systematically applying the rules in Ω to Δ until reaching a fixpoint. To make this more precise, consider the sequence of datasets defined as follows. $\Delta_0 = \Delta$, and $\Delta_{i+1} = \Delta_i \cup \bigcup_{\rho \in \Omega} A(\rho, \Delta_i)$. The closure of Ω on Δ (i.e., $C(\Omega, \Delta)$) is the union of the datasets in this sequence (i.e., $\bigcup \Delta_i$). Note that if Ω is a UCQ (see Definition 5.2), then $C(\Omega, \Delta) = \Delta \cup \bigcup_{\rho \in \Omega} A(\rho, \Delta)$, since the view relation is not in the body of any rule, meaning a fixpoint is reached once all rules are applied to the dataset.

Finally, we define the *positive extension*, or the set of all facts that can be "deduced" by applying rules to data, of a stratified logic program Ω on a dataset Δ (written $E^+(\Omega, \Delta)$) as follows. Our definition relies on a decomposition of Ω into strata $\Omega_1, \ldots, \Omega_k$. Let $\Delta_0 = \Delta$, and let $\Delta_{i+1} = C(\Omega_{i+1}, \Delta_i)$. Since there are only finitely many rules in a logic program and every stratum must contain at least one rule, there are only k sets to consider, where k is a finite number. In this case, $E^+(\Omega, \Delta) = \Delta_k$. For a UCQ Ω, $E^+(\Omega, \Delta) = C(\Omega, \Delta)$, since there is only one stratum. The *negative extension* of Ω on Δ is simply the complement of the positive extension, $E^-(\Omega, \Delta) = H(\Omega \cup \Delta) - E^+(\Omega, \Delta)$, where $H(\Omega \cup \Delta)$ is the Herbrand base of $\Omega \cup \Delta$.

3.4. Incomplete Datasets

So far, we have used a single dataset to define base relations. All factoids in the dataset are assumed to be true, and all factoids not in the set are assumed to be false. In order to deal with incomplete data, we need to refine this approach. To this end, we partition the Herbrand base into three disjoint subsets—a positive dataset (factoids that are true), a negative dataset (factoids that are false), and an unknown dataset (factoids whose truth value is unknown).

We represent an incomplete dataset as a pair, denoted $\langle P, N \rangle$, consisting of a positive dataset P and a negative dataset N. If H is the set of all base factoids, then the unknown dataset is simply $H - P - N$.

We say that a dataset Δ is a *completion* of $\langle P, N\rangle$ if and only if Δ contains all of the factoids in P and none of the factoids in N, i.e., it is a superset of P and it is disjoint from N.

3.5. Extensions on Incomplete Datasets

A *possible positive extension* of a logic program Ω applied to an *incomplete* dataset $\langle P, N\rangle$ is the positive extension of Ω applied to *some* completion on $\langle P, N\rangle$; a *possible negative extension* of Ω applied to $\langle P, N\rangle$ is the negative extension of Ω applied to *some* completion on $\langle P, N\rangle$.

Finally, the *positive extension* of a logic program Ω applied to an *incomplete dataset* $\langle P, N\rangle$ (hereafter written as $E^+(\Omega, \langle P, N\rangle)$) is the intersection of all possible positive extensions of $\langle P, N\rangle$; the negative extension of Ω on $\langle P, N\rangle$ (hereafter written as $E^-(\Omega, \langle P, N\rangle)$) is the intersection of all possible negative extensions of Ω applied to $\langle P, N\rangle$. Viewed another way, the positive/negative extension of Ω applied to $\langle P, N\rangle$ is the set of all factoids that occur in *every* positive/negative extension of Ω applied to $\langle P, N\rangle$.

3.6. Naive Algorithm

Naively, we can compute the positive and negative extensions of a logic program on an incomplete dataset by simply iterating over all completions of the dataset, computing the extensions of the program on those complete datasets [14], and taking the intersections of the results. Of course, doing things this way is exponential in the size of the Herbrand base.

4. Tables with Nulls

This section provides additional context and motivation for our work. It may be skipped without affecting understanding of our algorithm or its analysis.

In the database literature, where relations are expressed as tables, null values are used to express incompleteness in the data. There are three original classes of tables with nulls—Codd tables, naive tables and conditional tables [15, 8]—which can be interpreted under either the *open world* assumption (OWA) or the *closed world* assumption (CWA). In the ensuing comparison between tables with nulls and incomplete datasets, we focus on conditional tables under the CWA. We make the closed world assumption because no tuple can be surely false under the open world assumption (i.e., there can be no analog to the certainly false subset N under the OWA) and focus on conditional tables due to their support for expressing certainly false facts through conditions.

While conditional tables are powerful tools for representing incomplete data in databases, incomplete datasets possess some advantages which make them particularly conducive to both *the expression of* and *the efficient querying of certain answers on* incomplete data involving substantial uncertainty in logic programming applications.

Definition 4.1. Conditional tables [9, 8] generalize relational tables in two ways

1. In the column entries, variables (representing unknown values) are allowed in addition to the usual constants
2. Each tuple is associated with a condition, which is a Boolean combination of atoms of the form $x = y$, $x = a$, $a = b$ where $x, y \in$ Var and $a, b \in$ Cons

A conditional database is really a set of ordinary databases, one of which corresponds to the actual database. Under the CWA, the incomplete database represented by a conditional table T is defined as

$$\begin{aligned}\text{Rep}(T) = \{r = v(T) : v \text{ is a valuation}, v(T) = \\ \{v(t) : t \in T \text{ and } v \text{ makes the condition in } T \text{ true}\}\},\end{aligned}$$

where a valuation is a map from variables and constants to constants which is identity on the constants [15].

4.1. Expressivity

In conditional tables with nulls, the possible databases are obtained by instantiating the variables in the table to constants and retaining only those rows for which the corresponding condition holds. This, in a sense, requires delineating all *possibly true* facts within the table, whereas incomplete datasets require specifying only the *known* true and false facts.

Theorem 4.2. *There are forms of incomplete datasets which can be expressed by a constant number of facts, but which tables with nulls require arbitrarily many rows to express.*

Proof. Given some Herbrand base H, consider the incomplete dataset $P = N = \varnothing$, i.e., everything is uncertain (nothing is known). Note that we have written 0 facts to delineate this dataset. However, to express this in a conditional table, we would need at least as many rows as possibly true (i.e., uncertain) facts, as the number of rows is an upper bound on the number of facts retained in any valuation of the table. Thus, arbitrarily many rows are required for arbitrarily large Herbrand bases. □

The above theorem shows that whereas tables with nulls cannot, in practice, represent the dataset $P = N = \varnothing$ for sufficiently large Herbrand bases (as computer memory is finite), incomplete datasets can, trivially, do so. This capability can be crucial in applications where much uncertainty is expected.

4.2. Query Complexity

Although conditions support a rich notion of uncertainty, due to their potentially convoluted nature, even determining whether a fact is certainly true in the database represented by a conditional table T is coNP-complete in the data, i.e., $\cap(\text{Rep}(T))$ has a coNP-complete membership test [15].

Meanwhile, determining whether a fact is certainly true or false in an incomplete dataset $\langle P, N\rangle$ merely involves looking it up in P or N, respectively. The core technique behind our transformation-based algorithm (see Section 5) is to transform the query into one that only takes certainly true and false dataset facts, corresponding with P and N, respectively, into account. This fundamentally takes advantage of the delineation of known facts in incomplete datasets.

5. Program Transformation

We now demonstrate an approach to *efficiently* obtain extensions of logic programs on incomplete datasets by transforming the programs in a way that allows us to apply algorithms for obtaining extensions on *complete datasets*. The key is to replace each (base and view) relation r in the original program by two relations—one that denotes the "known" positive instances, rp, and another that denotes the "known" negative instances, rn. The transformation uses the universe relation, u, which is presumed to be true of all objects in U, the universe (see Section 3.1). Since we assume the universe is finite, u can be defined extensionally. While there are domains in which the size of U can result in computational slowdowns (e.g., when the universe consists of all companies on earth), in many domains, the universe can be restricted to a reasonable size (e.g., just the relevant companies in a particular supply chain), keeping computation efficient. We remark, further, that while the naive approach to querying certain answers is exponential in the size of U, our approach is polynomial.

Example 5.1. The tiered supplier relation (see Example 2.1) will be transformed into

```
tp(X, Y) :- sp(X, Y)
tp(X, Y) :- sp(X, Z) & sp(Z, Y)
tn(X, Y) :- u(X) & u(Y) & ∼ as(X, Y) & ∼ bs(X, Y)
as(X, Y) :- u(X) & u(Y) & ∼ sn(X, Y)
bs(X, Y) :- u(Z) & u(X) & u(Y) & ∼ sn(X, Z) & ∼ sn(Z, Y)
```

In words, the relation `tp` conveys that we know X is a tiered supplier of Y exactly if we know X is a direct supplier of Y or there is some Z in the universe such that we know X is a supplier of Z and Z is a supplier of Y. The relations `as` and `bs` convey that X is possibly a tiered supplier of Y exactly if it is not known that X is *not* a direct supplier of Y (i.e., X is possibly a direct supplier of Y) or there is some Z in the universe such that X is possibly a supplier of Z and Z is possibly a direct supplier of Y. Thus, the relation `tn` conveys that we know X is not a tiered supplier of Y exactly if X is not possibly a tiered supplier of Y.

Definition 5.2. A UCQ with head relation p consisting of R rules, $\rho_1, \ldots, \rho_R$, is a program of the form

$$\begin{aligned} p\left(\overline{X}\right) &:- q_1^1\left(\overline{Y}_1^1\right) \,\&\, \cdots \,\&\, q_{N_1}^1\left(\overline{Y}_{N_1}^1\right) \\ &\vdots \\ p\left(\overline{X}\right) &:- q_1^R\left(\overline{Y}_1^R\right) \,\&\, \cdots \,\&\, q_{N_R}^R\left(\overline{Y}_{N_R}^R\right), \end{aligned}$$

where $\overline{X}$ and all $\overline{Y}_j^i$ represent tuples of variables. Note that any non-recursive pure Datalog program can be unfolded into this form.

Definition 5.3. The transformation, $t(\Omega)$, of a UCQ Ω of the form specified in Definition 5.2 will be

$$\begin{aligned} pp\left(\overline{X}\right) &:- q_1^1 p\left(\overline{Y}_1^1\right) \,\&\, \cdots \,\&\, q_{N_1}^1 p\left(\overline{Y}_{N_1}^1\right) \\ &\vdots \\ pp\left(\overline{X}\right) &:- q_1^R p\left(\overline{Y}_1^R\right) \,\&\, \cdots \,\&\, q_{N_R}^R p\left(\overline{Y}_{N_R}^R\right) \\ pn\left(\overline{X}\right) &:- u\left(X_1\right) \,\&\, \cdots \,\&\, u\left(X_K\right) \,\&\, \sim a_1 s\left(\overline{X}\right) \,\&\, \cdots \,\&\, \sim a_R s\left(\overline{X}\right) \\ a_1 s\left(\overline{X}\right) &:- u\left(Z_1^1\right) \,\&\, \cdots \,\&\, u\left(Z_{M_1}^1\right) \,\&\, u\left(X_1\right) \,\&\, \cdots \,\&\, u\left(X_K\right) \,\&\, \\ &\qquad \sim q_1^1 n\left(\overline{Y}_1^1\right) \,\&\, \cdots \,\&\, \sim q_{N_1}^1 n\left(\overline{Y}_{N_1}^1\right) \\ &\vdots \\ a_R s\left(\overline{X}\right) &:- u\left(Z_1^R\right) \,\&\, \cdots \,\&\, u\left(Z_{M_R}^R\right) \,\&\, u\left(X_1\right) \,\&\, \cdots \,\&\, u\left(X_K\right) \,\&\, \\ &\qquad \sim q_1^R n\left(\overline{Y}_1^R\right) \,\&\, \cdots \,\&\, \sim q_{N_R}^R n\left(\overline{Y}_{N_R}^R\right), \end{aligned}$$

where each $a_i s$ corresponds with the rule ρ_i, $Z_1^i, \ldots, Z_{M_i}^i$ are the variables in the body unbound by the head and $\overline{X} = X_1, \ldots, X_K$.

Note that $t(\Omega)$ is safe and stratified. It has exactly two strata, where we can consider the rules for $pp\left(\overline{X}\right)$ and $a_1 s\left(\overline{X}\right), \ldots, a_R s\left(\overline{X}\right)$ to constitute the lower stratum, Ω_1, and the rule for $pn\left(\overline{X}\right)$ to constitute the higher stratum, Ω_2.

Definition 5.4. Our algorithm for computing the positive and negative extensions of Ω on an incomplete dataset $\langle P, N\rangle$ consists of computing the positive extension for $t(\Omega)$ on the transformed *complete* dataset $\text{posify}(P) \cup \text{negify}(N)$, where posify and negify simply mark all factoids with p and n, respectively.

6. Algorithm Analysis

In what follows, let Ω be a UCQ of the form delineated in Definition 5.3 and $\langle P, N\rangle$ an incomplete dataset.

6.1. Correctness

6.1.1. Negative Extensions

We show completeness, followed by soundness, of our transformation-based algorithm for computing negative extensions on UCQs.

Definition 6.1. Recall from Definition 5.3 that $\overline{X}$ is the tuple of variables in the head of Ω. For some tuple $\overline{x}$ of object constants in U (the universe) where $\overline{x}$ has the same length as $\overline{X}$, we define $f_{\overline{x}} : \text{Var} \to \text{Var} \cup U$, where Var is the set of variables, to be the function binding $\overline{X}$ to $\overline{x}$. Formally, for any variable Z,

$$f_{\overline{x}}(Z) = \begin{cases} x_i, & \text{if } Z = X_i \\ \\ Z, & \text{if } Z \notin \{X_i\}_i. \end{cases}$$

Then, where $\overline{Z} = Z_1, \ldots, Z_n$, define

$$\varphi_{\overline{x}}\left(\overline{Z}\right) = f_{\overline{x}}(Z_1), \ldots, f_{\overline{x}}(Z_n).$$

Theorem 6.2. *For any ground tuple $\overline{t}$, if $r\left(\overline{t}\right) \in E^-\left(\Omega, \langle P, N\rangle\right)$, then $rn\left(\overline{t}\right) \in E^+\left(t(\Omega), \textit{posify}(P) \cup \textit{negify}(N)\right)$.*

Proof. Assume $r\left(\overline{t}\right) \in E^-\left(\Omega, \langle P, N\rangle\right)$. Then, for all completions Δ of $\langle P, N\rangle$,

$$r\left(\overline{t}\right) \in E^-\left(\Omega, \Delta\right) = H(\Omega \cup \Delta) - E^+\left(\Omega, \Delta\right).$$

Thus, $r\left(\overline{t}\right) \notin E^+\left(\Omega, \Delta\right) = C(\Omega, \Delta) = \Delta \cup \bigcup_{\rho \in \Omega} A(\rho, \Delta)$ for all completions Δ of $\langle P, N\rangle$.

Note that r is either a base relation or the view relation in Ω. Say it is a base relation, q. Then, for all completions Δ of $\langle P, N\rangle$, since $\Delta \subseteq C(\Omega, \Delta)$, we have that $q\left(\overline{t}\right) \notin \Delta$. Thus, $q\left(\overline{t}\right) \in N$, meaning, by definition, that $rn\left(\overline{t}\right) = qn\left(\overline{t}\right) \in \text{negify}(N) \subseteq E^+\left(t(\Omega), \text{posify}(P) \cup \text{negify}(N)\right)$.

Then, say r is the view relation, p. Since $p\left(\overline{t}\right) \notin \bigcup_{\rho \in \Omega} A(\rho, \Delta)$ for all Δ completing $\langle P, N\rangle$, we claim it must be the case that for all rules ρ_i and all bindings $\psi : \text{Var} \to U$ of variables left unbound by $\varphi_{\overline{t}}$ (see Definition 6.1), at least one subgoal $q_j^i\left(\psi\left(\varphi_{\overline{t}}\left(\overline{Y}_j^i\right)\right)\right) \in N$, meaning $q_j^i n\left(\psi\left(\varphi_{\overline{t}}\left(\overline{Y}_j^i\right)\right)\right) \in \text{negify}(N)$. Assume, for the sake of contradiction, that this is not the case, i.e., there is some rule ρ_i with a binding ψ_i such that $q_j^i\left(\psi_i\left(\varphi_{\overline{t}}\left(\overline{Y}_j^i\right)\right)\right) \notin N$ for all j. Then, there is some dataset Δ^* completing $\langle P, N\rangle$ such that

$$q_1^i\left(\psi_i\left(\varphi_{\overline{t}}\left(\overline{Y}_1^i\right)\right)\right), \ldots, q_{N_i}^i\left(\psi_i\left(\varphi_{\overline{t}}\left(\overline{Y}_{N_i}^i\right)\right)\right) \in \Delta^*,$$

meaning $p\left(\overline{t}\right) \in A(\rho_i, \Delta^*)$, a contradiction. By the program transformation (see Definition 5.3), it is then clear that $a_1 s\left(\overline{t}\right), \ldots, a_R s\left(\overline{t}\right) \notin C(\Omega_1, \Delta) = \Delta_1$, meaning $rn\left(\overline{t}\right) = pn\left(\overline{t}\right) \in C(\Omega_2, \Delta_1) = \Delta_2 = E^+\left(t(\Omega), \text{posify}(P) \cup \text{negify}(N)\right)$. □

Theorem 6.3. *For any ground tuple $\overline{t}$, if $rn\left(\overline{t}\right) \in E^+\left(t(\Omega), \textit{posify}(P) \cup \textit{negify}(N)\right)$, then $r\left(\overline{t}\right) \in E^-\left(\Omega, \langle P, N\rangle\right)$.*

Proof. Assume $rn\left(\overline{t}\right) \in E^+\left(t(\Omega), \text{posify}(P) \cup \text{negify}(N)\right)$. If r is a base relation, q, then $qn\left(\overline{t}\right) \in \text{negify}(N)$, meaning $r\left(\overline{t}\right) = q\left(\overline{t}\right) \in N \subseteq E^-\left(\Omega, \langle P, N\rangle\right)$.

If r is the view relation, p, then for all rules ρ_i in Ω and all bindings ψ of variables unbound by the head, there is some subgoal $q_j^i\left(\overline{Y}_j^i\right)$ such that $q_j^i n\left(\psi\left(\varphi_{\overline{t}}\left(\overline{Y}_j^i\right)\right)\right) \in \text{negify}(N)$, meaning $q_j^i\left(\psi\left(\varphi_{\overline{t}}\left(\overline{Y}_j^i\right)\right)\right) \in N$. Thus, for all datasets Δ completing $\langle P, N\rangle$, $p\left(\overline{t}\right) \notin E^+\left(\Omega, \Delta\right) = C(\Omega, \Delta)$, meaning $p\left(\overline{t}\right) \in E^-\left(\Omega, \Delta\right)$. Thus, $r\left(\overline{t}\right) = p\left(\overline{t}\right) \in E^-\left(\Omega, \langle P, N\rangle\right)$. □

6.1.2. Positive Extensions

Since the proofs showing correctness for positive extensions are simpler versions of the analogous proofs for negative extensions, we state this theorem without proof.

Theorem 6.4. *For any ground tuple* $\bar{t}$, $r\left(\bar{t}\right) \in E^+\left(\Omega, \langle P, N\rangle\right)$ *if and only if* $rp\left(\bar{t}\right) \in E^+\left(t(\Omega), \mathit{posify}(P) \cup \mathit{negify}(N)\right)$.

6.2. Complexity

We now turn our attention to the efficiency of our proposed algorithm. Specifically, we evaluate the algorithm's *data complexity*, which, in the context of incomplete datasets, we define to mean the worst-case complexity of determining whether some ground atom $p\left(\bar{t}\right)$ is certainly true or false for some fixed program Ω, with respect to the size h of the Herbrand base $H(\Delta)$ of the dataset. First, recall that the naive approach (see Section 3.6) requires evaluating Ω over each completion of $\langle P, N\rangle$. In the worst case (e.g., $P = N = \varnothing$), the number of completions is exponential in h, meaning the naive algorithm has data complexity $\Omega(2^h)$. Our algorithm is much more efficient.

Theorem 6.5. *Our algorithm has polynomial time complexity in the size of the Herbrand base, with a one-time dataset-independent program preprocessing step.*

Proof. We carry out the dataset-independent program transformation step of our algorithm offline, yielding $t(\Omega)$. Processing an incomplete dataset $\langle P, N\rangle$ then involves two steps—transforming the dataset $\langle P, N\rangle \mapsto \text{posify}(P) \cup \text{negify}(N)$ and querying $t(\Omega)$ on the transformed dataset. Since our dataset transformation step simply involves marking each fact with a character, this step has linear complexity in $|P| + |N|$. Note that $|\text{posify}(P)| = |P|$ and $|\text{negify}(N)| = |N|$.

We now analyze the complexities of querying $pp\left(\bar{t}\right)$ and $pn\left(\bar{t}\right)$ on the dataset $\text{posify}(P) \cup \text{negify}(N)$. For this analysis, we conservatively assume all rules are applied, subgoals are processed left to right, and no indexing is done. We follow the evaluation procedure and its associated computational analysis outlined here [16]. In essence, iterating from left to right over the subgoals, the atom in each subgoal is compared against all dataset facts associated with its particular relation. A match occurs if the unbound variables in the atom can be consistently bound to object constants such that the atom matches the given dataset fact. Recall that under our semantics, variables can only bind to objects in U, the universe. Due to our safety property, no negative literal should have any unbound variables. Each match creates a branch in the procedure associated with the variable bindings that have been made. The goal is concluded if there is some branch where all positive subgoals and no negative ones have matched.

For any of the R rules $\rho_i \in \Omega$, there are M_i variables in the body unbound by the head and N_i subgoals (see Definition 5.3). Let M_i^n be the number of variables in the n^{th} subgoal from the left unbound by any goal to its left, where necessarily $\Sigma_{n=1}^{N_i} M_i^n = M_i$, and let $M = \max_{i\in[R]} M_i$ and $N = \max_{i\in[R]} N_i$. For any $\bar{t}$, the number of steps required to query $pp\left(\bar{t}\right)$ is then at most

$$
\begin{aligned}
\nu^+ &\leq \sum_{i=1}^{R}\left(|P| + \sum_{n=2}^{N_i} |U|^{M_i^1+\cdots+M_i^{n-1}} \cdot |P|\right) \\
&\leq \sum_{i=1}^{R} N_i \cdot |U|^{M_i} \cdot |P| \\
&\leq (R \cdot N) \cdot |U|^{M} \cdot |P|,
\end{aligned}
$$

which is polynomial in $|U| + |P|$. Similarly, the number of steps needed to query $pn\left(\bar{t}\right)$ is at most

$$\begin{aligned}\nu^{-} &\leq K \cdot |U| + \sum_{i=1}^{R} \left(|U| + \cdots + |U|^{M_i} + |U|^{M_i} \cdot (K \cdot |U| + N_i \cdot |N|)\right) \\ &\leq K \cdot |U| + \sum_{i=1}^{R} \left(M_i \cdot |U|^{M_i} + K \cdot |U|^{M_i+1} + N_i \cdot |U|^{M_i} \cdot |N|\right) \\ &\leq K \cdot |U| + (R \cdot M) \cdot |U|^{M} + (R \cdot K) \cdot |U|^{M+1} + (R \cdot N) \cdot |U|^{M} \cdot |N|,\end{aligned}$$

which is polynomial in $|U| + |N|$.

Thus, both for querying the positive and the negative extensions, our algorithm is polynomial in $|U| + |P| + |N|$. Since $|U| \leq h$ and $|P| + |N| \leq h$, it is then clear that this algorithm has polynomial time complexity in the size of the Herbrand base. □

7. Future Work

This transformation-based approach to reasoning with incomplete data in logic programming applies to broader classes of programs than just UCQs, such as programs with stratified negation, recursions, and constraints. For semipositive programs (a class of programs to which any stratified program can be unfolded), a simple extension of our UCQ transformation algorithm is already *sound*, but ensuring *completeness* will require preprocessing steps (e.g., resolution) and augmentations (e.g., consistency checking), a direction we are currently pursuing.

Acknowledgments

We thank David Warren and Rada Chirkova for helpful comments.

References

[1] T. Eiter, G. Gottlob, H. Mannila, Disjunctive datalog, ACM Trans. Database Syst. 22 (1997) 364–418. doi:10.1145/261124.261126.

[2] G. Brewka, T. Eiter, M. Truszczyński, Answer set programming at a glance, Commun. ACM 54 (2011) 92–103. doi:10.1145/2043174.2043195.

[3] M. Gelfond, V. Lifschitz, The stable model semantics for logic programming, in: R. Kowalski, Bowen, Kenneth (Eds.), Proceedings of International Logic Programming Conference and Symposium, MIT Press, 1988, pp. 1070–1080. URL: http://www.cs.utexas.edu/users/ai-lab?gel88.

[4] A. Van Gelder, A tutorial on the well-founded semantics, PROGRAMMING AND COMPUTER SOFTWARE C/C OF PROGRAMMIROVANIE 24 (1998) 32–42.

[5] S. Abiteboul, O. M. Duschka, Complexity of answering queries using materialized views, in: Proceedings of the Seventeenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, PODS '98, Association for Computing Machinery, New York, NY, USA, 1998, p. 254–263. doi:10.1145/275487.275516.

[6] D. Calvanese, G. de Giacomo, M. Lenzerini, M. Vardi, View-based query processing and constraint satisfaction, in: Proceedings Fifteenth Annual IEEE Symposium on Logic in Computer Science (Cat. No.99CB36332), 2000, pp. 361–371. doi:10.1109/LICS.2000.855784.

[7] A. Nash, L. Segoufin, V. Vianu, Views and queries: Determinacy and rewriting, ACM Trans. Database Syst. 35 (2010). doi:10.1145/1806907.1806913.

[8] T. Imieliński, W. Lipski, Incomplete information in relational databases, J. ACM 31 (1984) 761–791. doi:10.1145/1634.1886.

[9] G. Grahne, Conditional Tables, Springer US, Boston, MA, 2009, pp. 446–447. doi:10.1007/978-0-387-39940-9_1253.

[10] E. Toussaint, P. Guagliardo, L. Libkin, Knowledge-Preserving Certain Answers for SQL-like Queries, in: Proceedings of the 17th International Conference on Principles of Knowledge Representation and Reasoning, 2020, pp. 758–767. doi:10.24963/kr.2020/78.
[11] O. Goodenough, S. Salkind, Computable contracts and insurance: An introduction, MIT Computational Law Report, 2022. URL: https://law.mit.edu/pub/computablecontractsandinsuranceanintroduction/release/1, accessed 2026-07-08.
[12] T. Krueger, A. Mohapatra, M. Genesereth, Symbium: Using Logic Programming to Streamline Citizen-to-Government Interactions, Springer Nature Switzerland, Cham, 2023, pp. 271–276. doi:10.1007/978-3-031-35254-6_22.
[13] V. Chaudhri, F. Ameri, C. P. Cappas, P. Colohan, A. Davison, M. Kant, A. Padilla, S. Samples, E. Sallinger, C. Shimizu, T. C. Son, R. Uttarala, M. Witte, E. Young, A little semantics goes a long way. it's time to go further: Lessons from three supply chain use cases (2026). URL: https://www.semantic-web-journal.net/content/editorial-little-semantics-goes-long-way-it%E2%80%99s-time-go-further-lessons-three-supply-chain-use, under open community review.
[14] J. D. Ullman, Principles of Database and Knowledge-Base Systems, Computer Science Press, 1988.
[15] G. Grahne, Incomplete information., 2009.
[16] M. Genesereth, V. K. Chaudhri, Introduction to logic programming, Springer Nature, 2022.