The Problem of Correlation and Substitution in SPARQL

EXTENDED VERSION

Daniel Hernández¹, Claudio Gutierrez¹, and Renzo Angles²

¹ Universidad de Chile, Santiago, Chile
² Universidad de Talca, Curicó, Chile

Abstract. Implementations of a standard language are expected to give same outputs to identical queries. In this paper we study why different implementations of SPARQL (Fuseki, Virtuoso, Blazegraph and rdf4j) behave differently when evaluating queries with correlated variables. We show that at the core of this problem lies the historically troubling notion of logical substitution. We present a formal framework to study this issue based on Datalog that besides clarifying the problem, gives a solid base to define and implement nesting.

Keywords: Correlation, Substitution, SPARQL

1 Introduction

A subquery is a query expression that occurs in the body of another query expression, called the outer query. A correlated subquery is one whose evaluation is dependent in some way on data being processed in the outer query. Informally the data got from the outer query should be replaced or substituted in the corresponding places in the inner query. Thus the notion of substitution comes to the heart of the problem at hand.

It is well known the complexities that this notion involves. As is well known, it has been always a troubling concept and source of error even to renowned logicians. The query language SPARQL is not an exception: the notion of replacement (substitution) in the recommendation has an insufficient definition and even contradictory pieces.

3 The story is recounted by Church in [5], pp. 289-90 and Cardone and Hindley [4], p.7. Russell and Whitehead, although used the notion, missed its formal statement in his Principia (1913). Hilbert and Ackermann gave an “inadequate” statement in his 1928’s Logic. Carnap in Logische Syntax der Sprache and Quine in System of Logistic definitions still contain problems. Finally Hilbert and Ackermann in 1934 gave finally a correct statement.

4 See [11] where we report that the substitution notion presented on Sec. 18.6 of the current specification [10] contradicts the statements “Due to the bottom-up nature of SPARQL query evaluation, the subqueries are evaluated logically first, and the results are projected up to the outer query” and “Note that only variables projected out of the subquery will be visible, or in scope, to the outer query”, from Sec. 12.
We show in this paper that this notion is the source of problems that can be found in the evaluation of EXIST subqueries in SPARQL. The problem is highlighted when operators which incorporate possibly incomplete information are present. The following SPARQL query illustrates these problems. The query roughly asks for the id of persons and optionally their corporate email, subject to some conditions given by the expression in the FILTER EXISTS:

```
SELECT ?id ?email
WHERE {{
  ?id a :person
  OPTIONAL {
    ?id :corpMail ?email
  }

  FILTER EXISTS {
    {{
      ?id a :person
      OPTIONAL {
        ?id :privMail ?email
      }
    }

    FILTER (?email = *\.com)
  }
}
```

Surely the reader is facing the following problem: how to interpret this query? Well, you are not alone in your vacillation: The most popular implementations of SPARQL do not agree on it. For example, for the following database of persons

```
| id | 1    | 2    | 3    | 4    | 5    | 6    |
|----|------|------|------|------|------|------|
| corpMail | *\.com | *\.net | *\.com | *\.net | *\.com | *\.net |
| privMail | *\.net | *\.com |     |     |     |     |
```

Fuseki, Blazegraph, Virtuoso and rdf4j give almost all different results. Fuseki and Blazegraph give ((1,*\.com), (3,*\.com) and (5,-); Virtuoso gives (1,*\.com) and (3,*\.com); finally rdf4j gives (3,*\.com) and (5,-)\(^5\).

What is going on? Not simple to unravel. The main problem is how to assign the variables in order to evaluate the inner and outer expressions. Let us see why not all systems agree in showing up person 1. If we evaluate first the inner query then the variable ?email is bound to *\.net for person 1, thus the filter ?email = *\.com fails, and so the whole expression inside the first filter fails, hence person 1 is not shown in the output. Now, if the inner pattern is evaluated after binding the variable ?email to *\.com, then the OPTIONAL part does not match and thus the last filter pass, hence person 1 is outputted.

The intuition provided by the case of person 1 is that some systems evaluate the inner pattern before binding the ?email in the outer query, and others do it after the binding. Now this intuitive philosophy does not work to understand why Blazegraph and Fuseki outputs, (1,*\.com) and (5,-). In defense of these systems, let us recall that the specification is not clear or precise enough about this evaluation.

We —the Semantic Web advocates— are in a problem: No query language with such uncertainties will gain wide adoption. Of course the example is highly non-natural but, as we will show, it codes the essence of the problems of substitution of correlated variables in SPARQL. Substituting is a non trivial even when all values are constants, but when considering incomplete information, e.g. in the form of nulls or unbounds in SPARQL, the complexities rapidly scale up. In this setting, having a clean logical picture of what is going on is crucial.

\(^5\) The engines studied in this paper are Fuseki 2.5.0, Blazegraph 2.1.1, Virtuoso 7.2.4.2, and rdf4j 2.2.1.
Our aim is that the above problems can be modeled by using Datalog. Hence, we introduce Nested Datalog, a extension of classical Datalog to cope with nested expressions. Nested Datalog is used to describe two different substitution philosophies: *syntactic substitution*, which implies that the value to be substituted in a correlated variable comes from one source; and *logical substitution*, which implies that a variable is expecting values from two sources (from the valuation of the expression in the database and from the outer query).

The problem becomes more complex in the logical evaluation when one (or both) of the values is null. It turns out that it is not indifferent where we “compatibilize” both values (at the bottom of the program, at the top, in the middle). To model this it is needed one step further in the formalization in order to capture the nuances of incomplete information in the form of null values. To do this, we use *Modal Datalog*, a version of Datalog with modal features.

Once having the right setting, the subtleties of the notion of substitution in nested expressions having incomplete information (null values) becomes visible. Then, in Section 5, we show how SPARQL translates to this setting, and how the notion of substitution is expressed in it. Then we show the discrepancies of the different implementations and what do they mean under the light of this formal framework. We conclude with our view of how to handle this problem.

In summary, the contributions of this paper are the following:

1. Formalization of the problem (i.e. substitution in nested expressions), its study from a logical point of view, and analysis of the discrepancies of evaluation under different implementations of SPARQL.
2. To provide a logical framework to understand and formalize the notion of substitution in nested expressions in the presence of incomplete information for SPARQL.
3. Presentation of the logical (and consistent) alternatives defined and supported by our logical framework.

*Related work.* Different problems related to the notion of substitution are listed in the SPARQL specification errata. In a previous work we have reported some issues and presented three alternative solutions based on rewriting on the nested query before the substitution. After that, a W3C community group was created to address these issues. The community started defining queries and their expected outputs, for two alternative semantics. The first, proposed by Patel-Schneider an Martin, and the second proposed by Seaborne in the mailing list of the community group. None of these proposals study the problem in a formal framework as we do in this paper.

The idea of nested queries in Datalog is not new (see for example). We introduce nesting in a different way in order to be more suitable for studying correlation. Similarly, null values have been already studied in deductive databases (e.g.,), but with a focus (on computing certain answers) that is not the goal of this paper.

---

6 https://www.w3.org/community/sparql-exists/
2 Standard Datalog

We will briefly review notions of non recursive Datalog with equalities and safe negation. For further details see [15].

**Datalog Syntax.** A term is either a variable or a constant. An atom is either a predicate formula \( p(t_1, \ldots, t_n) \) where \( p \) is a predicate name and each \( t_i \) is a term, or an equality formula \( b_1 = b_2 \) where \( b_1 \) and \( b_2 \) are terms. A literal is either an atom (a positive literal \( L \)) or the negation of an atom (a negative literal \( \neg L \)). A rule \( R \) is an expression of the form \( L \leftarrow L_1, \ldots, L_n \) where \( L \) is a predicate formula called the head of \( R \), and \( L_1, \ldots, L_n \) is a set of literals called the body of \( R \). A fact \( F \) is a predicate formula with only constants. A program \( P \) is a finite set of rules and facts. A query \( Q \) is a pair \((L, P)\) where \( L \) is a predicate formula called the goal of \( Q \), and \( P \) is a program. We assume that all predicate formulas in a query \( Q \) with the same predicate name have the same number of arguments and that the terms in the head of each rule in \( Q \) and the goal of \( Q \) are all variables and are not repeated inside the same predicate.

**Semantics of Datalog.** Given a substitution \( \theta = \{X_1/a_1, \ldots, X_m/a_m\} \) from variables to constants, and a literal \( L \), then \( \theta(L) \) denotes the literal resulting of substituting in \( L \) each occurrence of \( X_i \) by \( a_i \), for \( 1 \leq i \leq m \). Given a set of facts \( S \), a substitution \( \theta \), and a positive literal \( L \) we say that \( S \) models \( L \) with respect to \( \theta \), denoted \( S, \theta \models L \), if either \( \theta(L) \) is an equality formula \( a = b \) where \( a \) and \( b \) are the same constant, or \( L \) is predicate formula and \( \theta(L) \in S \). Similarly, given a set of facts \( S \), a substitution \( \theta \) and an negative literal \( \neg L \), then \( S, \theta \models \neg L \), if \( S, \theta \not\models L \). Given a rule \( R = L_0 \leftarrow L_1, \ldots, L_n \) then a fact \( \theta(L_0) \) is inferred in a set of facts \( S \) if \( S, \theta \models L_j \), for \( 1 \leq j \leq n \).

A variable \( X \) is occurs positively in a rule \( R \) if \( X \) occurs in a positive predicate formula in the body of \( R \). A rule \( R \) is said to be safe if all its variables occur positively in \( R \). A program is safe if all its rules are safe. The safety restriction provides a syntactic restriction of programs which enforces the finiteness of derived predicates.

In this paper we do not consider equality formulas \( X = a \) as in most works (e.g., [15]), but equality formulas of the form \( \text{filter}(X = a) \). They differ in the form of evaluation. If a rule \( R \) has the literal \( X = a \) in the body, then \( X \) is said to be defined positively because the equality assigns the value \( a \) to \( X \). On the contrary, here equality formulas require that all variables be assigned before being evaluated. Formally, here equality formulas are built-in.

The dependency graph of a Datalog program \( P \) is a digraph \((N, E)\) where the set of nodes \( N \) is the set of predicates that occur in the literals of \( P \), and there is an arc \((p_1, p_2)\) in \( E \) for each rule in \( P \) whose body contains predicate \( p_1 \) and whose head contains predicate \( p_2 \). A Datalog program is said to be recursive if its dependency graph is cyclic, otherwise it is said to be non-recursive.

Let \( P \) be a program, \( E \) be a subset of the Herbrand base of \( P \), and \( \text{fact}(P, E) \) denote the set of facts occurring in \( E \) or \( P \), intuitively the facts inferred in zero steps. Then, the meaning of \( P \) is the result of adding to \( \text{fact}(P, E) \) as many new
facts as can be inferred from the rules of $P$ in $\text{fact}(P, E)$. The inference process is applied repeatedly until a fixpoint, denoted $\text{fact}^*(P, E)$, is reached. The answer of a query $Q = (L, P)$ in an extensional database $E$, denoted $\text{ans}_E(Q)$, is the subset of $\text{fact}^*(P, E)$ with facts having the same predicate than $L$.

The fixpoint depends on the order used to evaluate rules. Here we assume the order of Stratified Datalog, where for every arc $(p_1, p_2)$ in the dependency graph of a program $P$, a rule $R_2$ with head $p_2$ is not used in the inference process until every rule $R_1$ with head $p_1$ cannot be applied to infer another fact. Without this order a negative predicate formula $\neg L$ can be wrongly evaluated as true if the evaluation is done before a fact matching $L$ is inferred.

3 Nested Datalog

We will extend Datalog in order to be able to compose queries, by introducing nested queries as a new type of atom that occurs as a filter device (i.e., a built-in).

**Definition 1 (Syntax of Nested Datalog defined recursively).** 1. A Datalog query is a Nested Datalog query. 2. A Nested Datalog query is a Datalog query where Nested Datalog queries are allowed as atoms.

The inference process of Nested Datalog differs from the standard one by the addition of the semantics for evaluating Nested queries. If query $Q$ is an atom in a rule, its evaluation with respect to the substitution $\theta$ is true if and only if $\theta(Q)$ has at least one answer, where $\theta(Q)$ denotes the query resulting of “applying” to $Q$ the substitution $\theta$. This is the key notion we will study in what follows.

We will need the notion of assignment of a value to a variable. Consider a program $P$ with a single rule $R$ defined as $p(X) \leftarrow q(X), \text{filter}(X = Y)$. The assignment of the value $a$ to the variable $Y$ can be done by adding a literal $l(Y)$ to $R$, where $l$ is a fresh predicate, and the fact $l(a)$ to $P$. In the resulting program $Y$ can only take the value $a$. In what follows we will use the notation $\text{let}(Y = a)$ as a syntactical sugar to denote the result of assigning $a$ to $Y$ in $R$.

Substitution $\theta(Q)$ for a nested query $Q$ turns out to be rather subtle. There are two main approaches that we will call *syntactical* and *logical*.

**Syntactical substitution.** It works like standard replacement of a variable in an open sentence in logic or a free variable in a programming language. It occurs when a rule cannot be evaluated without having the value of a “free” variable occurring in it. Formally, a variable $X$ occurs free in a rule $R$ when it does not occur positively in $R$ and it occurs in a equality formula or in a nested query $Q$ where recursively $X$ occurs free in a rule of $Q$.

**Definition 2 (syntactic substitution).** Given a substitution $\theta$, a program $P$ and a variable $X$ occurring in $\theta$, then the syntactical substitution of $X$ in $P$ with respect to $\theta$ is done by adding the literal $\text{let}(X = \theta(X))$ to the body of each rule of $P$ where $X$ is free.

7 Allowing queries as atoms in rules means that a Nested Datalog rule can have the form $p(X) \leftarrow q(X), Q$ where the atom $Q$ is a Nested Datalog query.
Logical substitution. This is the problem of “substituting” \( \theta(X) \) in a program \( P \) that has no “free” \( X \) (i.e. all their rules \( R \) are “logically closed”). Conceptually, in this case the semantics is one such that after finding a solution of \( R \), it checks its “compatibility” with \( \theta \).

The essential problem of logical substitution is “when” (at what point in the evaluation) we will test this compatibility. For example, consider the program \( P \) with the rules \( p(X) \leftarrow q(X) \) and \( q(Y) \leftarrow r(Y) \). The variable \( Y \) in the second rule is logically connected with the variable \( X \) in the first rule. Thus, we can alternatively do the substitution in two places:

| Top-down | Bottom up |
|----------|-----------|
| \( p(X) \leftarrow q(X), \text{let}(X = \theta(X)) \) | \( p(X) \leftarrow q(X) \) |
| \( q(Y) \leftarrow r(Y) \) | \( q(Y) \leftarrow r(Y), \text{let}(Y = \theta(X)) \) |

This example illustrates two extremes, (1) Top down: evaluate \( P \) first, then, proceed to check the compatibility of the solution with \( \theta \). (2) Bottom up: Check compatibility of \( \theta \) (with the database) before starting the evaluation of \( P \). However, there are also several valid substitutions in the middle. In fact, we can start with a top-down substitution and then move the literals \( \text{let}(X = \theta(X)) \) down in the dependency graph of the program. The method used to move substitutions is equivalent to the standard method used to move selections \( \sigma_X = a \) in Relational Algebra because optimization concerns. In the appendix we provide a detailed description of the process of moving substitutions. The following result follows from it.

**Lemma 1.** Given a substitution \( \theta \) and a Nested Datalog program \( P \), then moving down literals of the form \( \text{let}(Y = \theta(X)) \) in the dependency graph of \( P \) does not change the semantics of \( P \).

Hitherto, we have defined logical substitution for a rule but not for a whole query. We now will define it using the top-down approach.

**Definition 3 (Top-down logical substitution).** Given a substitution \( \theta \), a query \( Q = (p(X_1, \ldots, X_n), P) \) and a variable \( X \) occurring in the goal of \( Q \) and \( \theta \), then the top-down logical substitution of \( X \) in \( Q \) is the query resulting of replacing the goal of \( Q \) by \( q(X_1, \ldots, X_n) \leftarrow p(X_1, \ldots, X_n), \text{let}(X = \theta(X)) \) to \( P \).

Logical and syntactical substitutions are not arbitrary. They are motivated by the EXISTS operator of SPARQL as is shown in the following query.

```
SELECT ?X
WHERE { ?X :hasMail ?Y
  FILTER EXISTS { SELECT ?X
    WHERE { ?X :hasMail ?Z FILTER (?Y <> ?Z) } } }
```

\(^8\) Other approaches can be obtained by moving the substitution point from upper levels to lower levels in the dependency graph of the query.
Intuitively this query finds people ?X with multiple emails. The variable ?X cannot be substituted by a constant in the nested query, because that substitution breaks the syntax of the SELECT clause (where only variables are allowed). Thus, ?X has to be substituted logically. On the other hand, ?Y is a free variable in the nested query, so ?Y has to be substituted syntactically. We claim that these substitutions are better understood if we rewrite this SPARQL query as the Nested Datalog query (p(X, Y), P₁) where P₁ is the following program:

\[ P₁ : \quad p(X, Y) \leftarrow \text{mail}(X, Y), \, (q(X), P₂), \]
\[ P₂ : \quad q(X) \leftarrow \text{mail}(X, Z), \, \text{filter}(Z \neq Y). \]

Hence, the result of applying a substitution \( \theta \) in \( P₂ \) produces a program with the rule \( q(X) \leftarrow \text{mail}(X, Z), \, \text{filter}(Z \neq Y), \, \text{let}(X = \theta(X)), \, \text{let}(Y = \theta(Y)) \).

There is a third form of substitution, namely improper. To gain some intuition, consider the following SPARQL query:

\[ \text{SELECT * WHERE ( ?X :r ?Y FILTER EXISTS ( SELECT ?Z WHERE ( ?X :s ?Z ) ) )} \]

The inner pattern of this query can be modeled as a query \((p(Z), P)\) where \(P\) contains the rule \(q(Z) \leftarrow s(X, Z)\). The variable \(X\) cannot be substituted logically, because it does not occur in the goal nor in the head of the unique rule of \(P\). Furthermore, \(X\) cannot be substituted syntactically, because is not free. However, some systems assume that \(X\) is correlated, as improper substitution where applied. Here, substitution is done by replacing the unique rule of \(P\) by the rules \(q(Z) \leftarrow u(X, Z)\) and \(u(X, Z) \leftarrow s(X, Z), \, \text{let}(X = \theta(X))\), where \(u\) is a fresh predicate.

Essentially, an improper substitution is a logical substitution that starts in some point of the nested query instead of the goal of the query, so that there is a gap in the logical chain. After that point, substitution is similar to logical substitution in the sense that it moves the value assigned to \(X\) from the head of a rule to the body, and thus to other rules below in the dependency graph.

In the example, the substitution is improper, because the logical chain start on the second rule so there is a gap between the goal of the nested query and the point where the logical chain starts.

Improper substitution breaks the design of Datalog where the scope of variables is the rule where they occur. It disallows renaming variables because its scope could be extended beyond the nested query, so breaking the design of logic and compositional languages where non free variables are renamed without changing the semantics of the expression, and free variables cannot be renamed. In our opinion, this compromises the compositionality of queries.

Now we have a parametrical definition of \(\theta(Q)\) for a query \(Q\) (i.e., without the values of a particular substitution \(\theta\)), independently of the kind of substitution used (syntactical, logical or improper) substitutions are used. A consequence, is the following result.

**Lemma 2.** Given a Nested Datalog query \(Q = (p(X₁, \ldots, Yₙ), P)\) and a simple extensional database \(E\) for \(Q\), then there exists a first order formula \(\phi\) such that each \(\text{ans}_E(Q) = \{p(X₁, \ldots, Yₙ) \mid E, \phi \models p(X₁, \ldots, Yₙ)\}\).
Proof. Is well known that without nesting each Datalog query $Q$ corresponds to a such formula $\phi$. The proof is done by translating each atom into a first order formula, then rules, and so forth. To consider Nested queries suffices giving a recursive translation for them. We define it as follows. The formula of a nested query $Q = (p(X_1, \ldots, X_n), P)$ is the first order formula $\exists Y_1 \ldots \exists Y_n (p'(Y_1, \ldots, Y_n) \land X_j = Y_j \land \cdots \land X_k = Y_k \land \phi_p)$, where $\phi_p$ is the formula of $P$ after renaming each predicate $q$ in $P$ with a fresh predicate $q'$ (this ensures that the evaluation is isolated from the outside of the nested query), and $X_j, \ldots, X_k$ are the variables that occur positively in the rule where $Q$ is nested. The equalities $X_j = Y_j \land \cdots \land X_k = Y_k$ model the logical substitution. Also, free variables $X$ in rules of $P$, are not added in $\forall X$ quantifiers of rules of $P$, so they are syntactically substituted. Improper substitution of a variable $X$ in a rule $E$ can be simulated by removing the $\forall X$ in the formula of $R$.

A corollary of this lemma is that nesting does not add expressive power to Datalog. Hence, if we chose using one form of substitution but not the other (e.g., using logical substitution but not syntactical), we will have the same expressive power than if we where chosen the contrary, or no substitution.

4 Modal Datalog

Modal Datalog is a version of Datalog where each rule is labeled with a modal logical operator. In what follows we develop it.

Most real world information includes incomplete data. The main technique to codify incompleteness have been the null values. A null, denoted $\perp$, represents either that the value is missing or non applicable. A relation containing null values is said incomplete, while one without them is said complete. The semantics of an incomplete relation is the set of all possible complete relations resulting of replacing consequently each null by a constant or a symbol $\top$, denoting a non-applicable value. We follow the semantics of $\perp$ and $\top$ by Lerat and Lipski [14]. However, in this paper we only consider nulls, because the evaluation process of Modal Datalog never generates non applicable values if they are not present in the database.

The answer of a query in a complete database $D$ is characterized by the set $\{ \mu \mid D, \mu \models \phi \}$ where $\phi$ is a first order formula whose free variables are instantiated by $\mu$. On the other hand, an incomplete database is interpreted as a set of complete databases $X$, the possible worlds. In this context, we can give modal characterizations to an answer $\mu$: $\mu$ is sure, denoted $X, \mu \models \Box \phi$, if $D, \mu \models \phi$ for all $D \in X$. Similarly, $\mu$ is a maybe answer, denoted $X, \mu \models \Diamond \phi$, if there exists a database $D \in X$ such that $D, \mu \models \phi$.

Modal Datalog essentially introduces a mode for each rule $R$ in a Datalog program. If the mode of $R$ is $\Box$, then $R$ is said to be sure and infers facts that are valid for all instances of the null values occurring in $R$ in the current

---

9 Rules $H \leftarrow B$ are translated as $\forall Z_1 \ldots \forall Z_n (B \rightarrow H)$ where $Z_1, \ldots, Z_n$ are the non free variables of the rule.
database. Otherwise, if the mode of $R$ is $\Diamond$, then $R$ is said to be a maybe rule and infers facts that are valid for at least one instance of the null values.

**Definition 4 (Modal Datalog Syntax).** A Modal Datalog Program is a set of rules of the form $\circ (p(X_1, \ldots, X_n) \leftarrow B)$ where $\circ$ is either $\Box$ or $\Diamond$, and $B$ is a set of literals allowing the value $\perp$.

A modal Datalog query is one built with Modal Datalog programs.

Next we present how modal predicates are derived from sets of facts and substitutions. We write $S, \theta \models \Box L$ if $L$ is derived from $S$ and $\theta$ with the label $\Box$ sure or maybe. We say that a Modal Datalog predicate formula $L_1$ is less informative than another $L_2$, denoted $L_1 \leq L_2$, if every instance of $L_1$ is an instance of $L_2$. Let $L$ be a literal, $S$ be a set of facts, and $\theta$ be a substitution. Then:

- $S, \theta \models \Box L$ if one of the following conditions holds:
  - $L$ is a positive predicate formula and $\theta(L) \in S$.
  - $L$ is a negative predicate formula $\neg q(t_1, \ldots, t_m)$ and there does not exist a fact $q(a_1, \ldots, a_m)$ in $S$ such that $\theta \models \Box (t_j = a_j)$, for $1 \leq j \leq m$.
  - $L$ is let($X = t$) and $\theta(X)$ is $t$.
  - $L$ is filter($t_1 = t_2$) or filter($t_1 \neq t_2$) and $\theta \models \Box (L)$.
  - $L$ is $(L', P')$ and $(L', \theta(P'))$ has at least one answer.
  - $L$ is $\neg (L', P')$ and $(L', \theta(P'))$ has no answers.

- $S, \theta \models \Diamond L$ if one of the following conditions holds:
  - $L$ is a positive predicate formula and there exists a fact $F \in S$ such that $F \leq \theta(L)$.
  - $L$ is a negative predicate formula $\neg q(t_1, \ldots, t_m)$ and there does not exist a fact $q(a_1, \ldots, a_m)$ in $S$ such that $\theta \models \Diamond (t_j = a_j)$, for $1 \leq j \leq m$.
  - $L$ is let($X = t$) and $\theta(X)$ is $t$.
  - $L$ is filter($t_1 = t_2$) or filter($t_1 \neq t_2$) and $\theta \models \Diamond (t_1 \neq t_2)$.
  - $L$ is $(L', P')$ and $(L', \theta(P'))$ has at least one answer.
  - $L$ is $\neg (L', P')$ and $(L', \theta(P'))$ has no answers.

**Definition 5 (Semantics of Modal Datalog).** Given a Modal Datalog query $(L, P)$, a database $E$, and a set of already inferred facts $S$, then a fact $F$ is inferred from a rule $\circ (H \leftarrow B)$ in $P$ ($\circ$ is either $\Box$ or $\Diamond$) if and only if there exists a substitution $\theta$ such that $S, \theta \models \circ L$ for all literals $L \in B$, $\theta$ is the less informative substitution $\theta'$ such that $S, \theta' \models \circ L'$ for all positive predicate formulas $L'$ in $B$, and $\theta(H) = F$.

**Example 1.** Let $E = \{ r(a), s(\perp), t(\perp) \}$ be a set of facts and $Q = (p(X), P)$ be the query where the program $P$ has the rules $R_1 : \Box (p(X) \leftarrow q(X), r(X))$ and $R_2 : \Diamond (q(X) \leftarrow s(X), t(X))$. Then, let us evaluate $\text{ans}_E(Q)$. The literal $r(X)$ in $R_1$ is true only with the substitution $\{X/a\}$, because $r(a)$ is the only available fact. To infer $p(a)$ we need first to infer $q(a)$ from rule $R_2$. Despite rule $R_2$ infers the less informative fact, i.e., $q(\perp)$. Hence, $\text{ans}_E(Q)$ is empty.
Now we are ready to define the promised notion of substitution in Modal Datalog. As before, we will make a distinction between syntactical substitution and logical substitution.

In the case of **syntactical substitution**, the approach is the same. We add a literal \( \text{let}(X = \theta(X)) \) to the rule where \( X \) occurs free. As this literal is syntactic sugar for introducing a predicate formula \( u(X) \), then \( \text{let}(X = \theta(X)) \) is indeed a positive predicate formula, so \( X \) occurs positively in it (see Def. 5). Then, \( X \) is not free anymore, and takes the value of \( \theta(X) \), that can be a null or a constant.

The **logical substitution** is more subtle. As we saw, in logical substitutions the value of a variable \( X \) comes from more than one source. For example, if one simply adds the literal \( \text{let}(X = \theta(X)) \) to the body of the rule \( \circ(p(X) ← q(X)) \) (\( \circ \) is either \( \Box \) or \( \Diamond \)), then the value of \( X \) will be provided by two sources, namely \( \theta(X) \) and the literal \( q(X) \). The problem is what to do if in one source \( X \) is null and in the other a constant. We solve this problem by splitting the rule in two, so one preserves the mode \( \circ \) of the original rule and in the other we use the mode \( \Diamond \) to merge the values coming from both sources.

The logical substitution of \( X \) in a rule \( R = \circ(p(X_1, \ldots, X_n) ← B) \) is the replacement of \( R \) by the rules \( \Diamond(p(X_1, \ldots, X_n) ← u(X_1, \ldots, X_n), \text{let}(X = \theta(X))) \) and \( \Box(u(X_1, \ldots, X_n) ← B) \), where \( u \) is a fresh predicate. The mode \( \Diamond \) in the first rule checks the compatibility and follows the SPARQL design, where a null value is considered compatible with a constant.

We showed that in Nested Datalog logical substitutions can be moved down in the dependency graph of a program without changing its results (Lemma 1). The following Lemma states that this feature does not hold in Modal Datalog.

**Lemma 3.** Given a substitution \( \theta \) and a Modal Datalog program \( P \), then moving down literals of the form \( \text{let}(Y = \theta(X)) \) in the dependency graph of \( P \) could change the semantics of \( P \).

**Proof.** It suffices to show an example that witnesses this. Consider the query \( Q = (p(X, Y), P) \) where \( P \) is the unique rule \( \Box(p(X, Y) ← r(X), s(Y), \text{filter}(X = Y)) \).

| Program A | Program B |
|-----------|-----------|
| \( \Diamond(p(X, Y) ← u(X, Y), \text{let}(X = \theta(X))) \) | \( \Box(p(X, Y) ← u(X), s(Y), \text{filter}(X = Y)) \) |
| \( \Box(u(X, Y) ← r(X), s(X), \text{filter}(X = Y)) \) | \( \Diamond(u(X) ← r(X), \text{let}(X = \theta(X))) \) |

In a database containing the facts \( r(⊥) \) and \( s(a, a) \), and \( \theta = \{X/a\} \), program A will have no solutions and program B will have the solution \( p(a, ⊥) \). \( \Box \)

The example in the previous proof shows:

**Corollary 1.** In Modal Datalog bottom-up and top-down evaluations do not behave in the same manner.

## 5 Substitution in FILTER EXISTS expressions

By using the machinery of Modal Datalog, we will model the evaluation of FILTER EXISTS expressions in SPARQL. Our objective is to provide a framework for safe semantics for correlated subqueries.
5.1 SPARQL codified as Modal Datalog

First we show that SPARQL can be coded in Modal Datalog using a translation inspired by [1], that is, via relational algebra. The syntax and semantics of the SPARQL fragment studied here are defined using Relational Algebra with set semantics (as is done in [6] and [2]).

We write \( r(R) \) to denote a relation \( r \) with schema (attributes) \( R \), that is, a set of mappings \( \mu \) from \( R \) to constants or null values. We extend standard relational algebra to handle null values by using modal evaluation when needed:

\[
\pi_{X_1,\ldots,X_n}(r) = \{ \mu[X_1,\ldots,X_n] | \mu \in r \},
\]

\[
\rho_{X/Y}(r) = \{ \mu[X/Y] | \mu \in r \},
\]

\[
r \cup s = (r \times \bot^R \setminus S) \cup (s \times \bot^S \setminus R),
\]

\[
\sigma_{\Box \phi}(r) = \{ \mu \in r | \mu \models \Box \phi \},
\]

\[
r \setminus s = \{ \mu \in r | \nexists \mu' \in s \forall X \in R \cap S : \Diamond (r[X] = s[X]) \},
\]

\[
r \bowtie \Diamond s = \{ \mu_1 \bowtie \mu_2 | \mu_1 \in r, \mu_2 \in s, \text{ and } \forall X \in R \cap S : \Diamond (r[X] = s[X]) \},
\]

where \( r(R), s(S) \) are two relations; \( \mu[T] \) is the truncation of the tuple \( \mu \) to the set of attributes \( T \); \( \mu[X/Y] \) is the renaming of the attribute \( X \) by \( Y \); \( \mu_1 \bowtie \mu_2 \) is the concatenation of tuples, where \( X \) takes the most informative value for each common attribute \( X \), or the available value if \( X \) is not common; and \( \bot^T \) is the relation \( t(T) \) with a single tuple filled with null values.

Here we study the fragment of SPARQL composed by the operators \( \sigma_{\Box \phi} \), \( \pi_{X_1,\ldots,X_n} \), \( \rho_{X/Y} \), \( \cup \), \( \bowtie \Diamond \), \( \setminus \). Note that we have selected one mode for each modal operator. The difference \( \setminus \) corresponds to the operator MINUS in [17], referred as DIFF in [3]. The fragment where the operands of \( \cup \) have the same attributes precludes the emergence of nulls when evaluating databases without nulls, thus coincides with Relational Algebra. Otherwise, the extended algebra is required. In this context, the operator OPTIONAL, denoted here as \( \bowtie \), is defined in [17] as \( R \bowtie S = (R \bowtie \Diamond S) \cup (R \setminus \Diamond S) \).

**Definition 6 (From algebra to Modal Datalog).** Given two relations \( r(R) \) and \( s(S) \), the Modal Datalog rules for each algebraic operator are defined as

\[\text{10} \text{ The standard MINUS is slightly different in the case when the subtracting mapping has no attributes, but both can be mutually simulated (see [13]).}
\]

\[\text{11} \text{ This definition of } \bowtie \text{ is slightly different with such stated by the standard. However, it is well known that the standard SPARQL operators are definable in the algebra presented here (e.g., see [13]).} \]
follows:

\[
\begin{align*}
\sigma_{\Box \varnothing}(r) & : \Box (p(R) \leftarrow r(R), \text{filter(\varnothing)}) \\
\pi_T(r) & : \Diamond (p(T) \leftarrow r(R)) \\
\rho_{X/Y}(r) & : \Diamond (p((R \setminus \{X\}) \cup \{Y\}) \leftarrow r(R), \text{filter}(X = Y), \text{let}(Y = \bot)) \\
(r \cup s) & : \Diamond (p(R \cup S) \leftarrow r(R), \text{let}(X_1 = \bot), \ldots, \text{let}(X_n = \bot)) \text{ and} \\
&s \leftarrow (p(R \cup S) \leftarrow s(S), \text{let}(Y_1 = \bot), \ldots, \text{let}(Y_m = \bot)), \\
&\text{where } \{X_1, \ldots, X_n\} = S \setminus R \text{ and } \{Y_1, \ldots, Y_m\} = R \setminus S \\
(r \bowtie s) & : \Diamond (p(R \cup S) \leftarrow r(R), s(S)) \\
r - \Box s & : \Box (p(R) \leftarrow r(R), \neg q(R \cap S) \text{ and } \Box (q(R \cap S) \leftarrow s(S))
\end{align*}
\]

The translation of each algebraic operator into a set of Modal Datalog rules by Def. 4 allows translating algebraic expressions into Modal Datalog queries.

Example 2. Given the relations \(r(X, Y), s(X, Y)\) and \(t(X, Z)\), the expression \((r \bowtie (s - \Box t))\) is translated as the query \((p(X, Y), P)\) where the program \(P\) has the rules \(\Diamond (p(X, Y) \leftarrow r(X, Y), q(X, Y)), \Box (q(X, Y) \leftarrow s(X, Y), \neg u(X)), \text{ and } \Box (u(X) \leftarrow t(X, Z))\).

Lemma 4. Let \(Q\) be an algebra expression and \(Q^*\) be theModal Datalog query obtained from \(Q\) according to Def. 4. For every database \(E\), it holds that \(Q\) and \(Q^*\) are equivalent, i.e. their evaluation results in the same answers. \(^{12}\)

5.2 Modal Datalog semantics of FILTER EXISTS

Hitherto, we have a semantics for a SPARQL fragment and a translation to Modal Datalog, except for expressions of the forms \(\sigma_Q(P)\) and \(\sigma_{\neg Q}(P)\), where \(P\) and \(Q\) are called respectively the outer and the inner patterns. \(^{13}\)

The philosophy of these operators is the following. Given a relation \(r\) and an algebraic expression \(Q\), we have that \(\sigma_Q(r)\) and \(\sigma_{\neg Q}(r)\) return the set of tuples \(\mu\) where \(\mu(Q)\) has at least a solution or no solutions, respectively. According to the SPARQL specification, \(\mu(Q)\) is the result of replacing in \(Q\) each variable \(X\) in the domain of \(\mu\) by \(\mu(X)\). As we indicated in the introduction, this definition is ambiguous and contradictory with other parts of the specification, and (as expected) systems have different interpretations for it.

We will unveil this problem by showing how the definition of \(\mu(Q)\) is viewed in Modal Datalog where the notion of substitution shows up in a clean logical manner.

\(^{12}\) Note that answers of \(Q\) have the form \((a, b)\) while answers of \(Q^*\) have the form \(p(a, b)\). In this lemma we assume that these answers are the same as they have the same components.

\(^{13}\) In the standard syntax these operators correspond to \((P \text{ FILTER EXISTS } Q)\) and \((P \text{ FILTER NOT EXISTS } Q)\), respectively.
Definition 7 (Filter Exists). Given a relation \( r \) and a SPARQL query \( Q \), the expressions \( \sigma_Q(r) \) and \( \neg\cdot_Q(r) \) are translated to the Modal Datalog rules
\[
\Box (p(T_r) \leftarrow r(T_r), (L, P)) \text{ and } \Box (p(T_r) \leftarrow \neg r(T_r), \neg(L, P))
\]
respectively.

Now we are ready to enumerate three sources of discrepancy in the interpretation of a rule \( R = o(H \leftarrow B, (L, P)) \):

1. **Free variables** A free variable \( X \) in \( P \) is in some cases assumed uncorrelated.
   The lack of correlation of a variable is simulated by replacing \( R \) by the rule
   \( o(H \leftarrow B, (L, P'), \text{let}(X' = \bot)) \), where \( X' \) is a fresh variable and \( P' \) is the result of replacing each occurrence of \( X \) in \( P \) by \( X' \).

2. **Improper substitution.** Some engines implement improper substitution.
3. **Substitution level.** A variable \( X \) in the goal of the nested query can be substituted logically in different places, ranging from top-down to bottom-up logical substitution. These substitutions are not equivalent (see Lemma [3]).

Next we present example queries to show the ways the systems address the above sources of discrepancy. We will use the dataset presented in the introduction. We assume that persons(\( X \)) is a pattern giving all persons \( X \) in the database, and \( m_1(X, Y) \) and \( m_2(X, Y) \) are patterns returning respectively the corporate and personal emails \( Y \) of a person \( X \).

**The first two discrepancies.** Consider the queries \( \sigma_{m_2(X, Z)}(T), \sigma_{\pi_Z(m_2(X, Z))}(T) \) and \( \sigma_{\pi_X(m_2(Y, Z))}(T) \), where \( T \) is \( m_1(\_*, \_*.\text{com}) \). These queries can be simulated with the same Modal Datalog query having goal \( p(X) \) and a single rule \( \Box (p(X) \leftarrow m_1(X, b), Q) \), where \( Q \) is either \( Q_1 = (q(X, Z), \Box (q(X, Z) \leftarrow m_2(X, Z))) \), \( Q_2 = (q(Z), \Box (q(Z) \leftarrow m_2(X, Z))) \), or \( Q_3 = (q(Y, Z), \Box (q(Y, Z) \leftarrow m_2(Y, Z), \text{filter}(Y = X))) \). Now, we have the following cases:

- **Case \( Q = Q_1 \):** It has a unique alternative which is substituting \( X \) logically, so returning only person 1. All systems agree with this answer.
- **Case \( Q = Q_2 \):** It has two interpretations, namely allowing or do not allowing improper substitution. In the first, answers include only person 1. In the second answers are persons 1 and 3. Blazegraph and Fuseki agree with the first interpretation, while rdf4j and Virtuoso with the second.
- **Case \( Q = Q_3 \):** It has two interpretations, depending whether \( X \) is assumed correlated or not. In the first interpretation, person 1 is the unique answer. In the second interpretation, there are no answers because \( X \) is \( \bot \) when evaluating \( \text{filter}(Y = X) \). Blazegraph, rdf4j and Virtuoso agree with the first interpretation, while Fuseki with the second.

**The last discrepancy.** To check the application of logical substitution, consider the query \( \sigma_Q(\text{persons}(X) \nexists (m_2(X, Y))) \) where \( Q \) is \( \sigma_{\pi_{*, \_*.\text{com}}}(\text{persons}(X) \nexists m_2(X, Y)) \), corresponding to the one presented in the introduction. Fig. 1 depicts the dependency graph of \( Q \). Some edges are labeled to refer alternative places where a substitution \( \theta \) can be applied. For instance, the top-down approach

\[\text{Due to space limitations, here we consider only the positive case } \sigma_Q(r). \text{ It is not difficult to extend the results for the negative case } \neg\cdot_Q(r).\]
Fig. 1. Dependency graph of the Nested Datalog query for the inner pattern $Q$.

consists in inserting the rule $\diamond (p(X, Y) \leftarrow p'(X, Y), \text{let}(X = \theta(X), \text{let}(Y = \theta(Y)))$ in the edge 1, and replacing $p$ in the rule below by $p'$, where $p'$ is a fresh predicate. Hence, only person 5 succeeds. No system agrees with this evaluation.

If substitutions are done in Level 2, then persons 3 and 5 succeed. Only rdf4j agrees with this interpretation.

Blazegraph and Fuseki agree with substitution in Level 3, i.e., just after variables instantiation. Strictly, this approach is not logical as it has some improper substitutions. Indeed, the logical chain of variable $Y$ below the edge 7 does not start in the goal of the nested query. Here persons 1, 3 and 5 succeed.

Virtuoso, Blazegraph and Fuseki follow a similar approach but they also append the literals $\text{let}(X = \theta(X))$ and $\text{let}(Y = \theta(Y))$ to the rule below edge 1. Hence, person 5 is discarded by the filter on this rule.

6 Conclusions

This work shows that the notion of substitution continues to haunt researchers and developers. We showed that it is at the core of the subtle problems that the specification and implementations of SPARQL face regarding subqueries of the form FILTER EXISTS.

We think the lasting contribution of this paper is the finding of several types of substitution in the presence of nested expression and incomplete information, that are playing some roles and seems that had passed unnoticed until now.

Although we consciously did not advance any proposal to fix the problems of substitution in nested expressions in SPARQL, the paper leaves a chart with the possible avenues to solve them. We think that the Working Groups of the W3C have the last word on this issue.
References

1. R. Angles and C. Gutierrez. The Expressive Power of SPARQL. In Proc. of the International Semantic Web Conference (ISWC), volume 5318 of LNCS, pages 114–129. Springer, 2008.
2. R. Angles and C. Gutierrez. The Multiset Semantics of SPARQL Patterns. In Proc. of the International Semantic Web Conference (ISWC), volume 9981 of LNCS, pages 20–36. Springer, 2016.
3. R. Angles and C. Gutierrez. Negation in SPARQL. In Proc. of Alberto Mendelzon International Workshop on Foundations of Data Management (AMW), volume 1644 of CEUR Workshop Proceedings, 2016.
4. F. Cardone and J. R. Hindley. History of lambda-calculus and combinatory logic. Handbook of the History of Logic, 5:723–817, 2006.
5. A. Church. Introduction to mathematical logic, volume 13. Princeton University Press, 1996.
6. R. Cyganiak. A Relational Algebra for SPARQL. Digital Media Systems Laboratory HP Laboratories Bristol. HPL-2005-170, page 35, 2005.
7. F. Dong and L. V. S. Lakshmanan. Deductive databases with incomplete information. In Proc. of the Joint International Conference and Symposium on Logic Programming (JICSLP), pages 303–317. MIT Press, 1992.
8. F. Dong and L. V. S. Lakshmanan. Intuitionistic Interpretation of Deductive Databases with Incomplete Information. Theor. Comput. Sci., 133(2):267–306, 1994.
9. L. Giordano and A. Martelli. Structuring logic programs: A modal approach. J. Log. Program., 21(2):59–94, 1994.
10. S. Harris and A. Seaborne. SPARQL 1.1 Query Language - W3C Recommendation, March 21 2013.
11. D. Hernandez, C. Gutierrez, and R. Angles. Correlation and substitution in SPARQL. CoRR, abs/1606.01441, 2016.
12. Q. Kong and G. Chen. On Deductive Database with Incomplete Information. ACM Trans. Inf. Syst., 13(3):355–369, 1995.
13. R. Kontchakov and E. V. Kostylev. On expressibility of non-monotone operators in SPARQL. In Proc. of the International Conference on Principles of Knowledge Representation and Reasoning (KR), pages 369–379. AAAI Press, 2016.
14. N. Lerat and W. Lipski Jr. Nonapplicable Nulls. Theor. Comput. Sci., 46(3):67–82, 1986.
15. M. Levene and G. Loizou. A guided tour of relational databases and beyond. Springer, 1999.
16. P. F. Patel-Schneider and D. Martin. Existential aspects of SPARQL. In Proc. of the International Semantic Web Conference (ISWC). Posters & Demonstrations Track, 2016.
17. J. Perez, M. Arenas, and C. Gutierrez. Semantics and Complexity of SPARQL. ACM Trans. Database Syst., 34(3):16:1–16:45, 2009.
A Moving logical substitutions to lower levels

In this appendix we will describe how a substitution literal \( \text{let}(X = \theta(Y)) \) can be moved to lower levels of the dependency graph without changing the semantics of a Nested Datalog query. Because Nested Datalog programs \( P \) are non recursive, the structure of the dependency graph of \( P \) is a tree, so substitution literals can be moved down repeatedly until the leaf predicates are reached.

We need another structure. Given a program \( P \), then we call the tree of \( P \) to the rooted tree whose nodes are labeled with the rules of \( P \) and where a node labeled rule \( R_1 \) is the child of a node labeled by a rule \( R_2 \) if the head of \( R_1 \) is in the body of \( R_2 \).

Now, we will describe how to move down an literal \( \text{let}(Y = \theta(X)) \) in a rule \( R \) of a program \( P \) where this literal occurs. Without loss of generality, consider that \( R \) is the following rule:

\[
H \leftarrow L_1, \ldots, L_n, \text{let}(Y = \theta(X)), L'_1, \ldots, L'_m,
\]

where \( L_1, \ldots, L_n \) are positive predicates formulas with intensional predicates where \( Y \) occurs and \( L'_1, \ldots, L'_m \) are the rest of the literals in the body of \( R \).

For \( 1 \leq i \leq n \), let \( p(X_1, \ldots, X_n) \) be \( L_i \) and \( J_i \) be the set of positions \( j \) of \( L_i \) such that \( X_j = Y \) (i.e., are the same variable). Then, rename \( p \) in \( L_i \) with a fresh predicate name \( p_i \) and do the following for each rule \( R' \) in \( P \) of the form \( p(Z_1, \ldots, Z_n) \leftarrow B \):

1. Let \( T \) be a copy of the thread whose root is \( R' \) in the tree of \( P \), and \( T' \) the result of renaming consequently all intentional predicates occurring in \( T \) by a fresh predicate, except \( p \) that is renamed as \( p_i \).
2. Append the literal \( \text{let}(Z_j = \theta(X)) \) to the body of the root of \( T' \), for each position \( j \) in \( J_i \).
3. Copy the rules of \( T' \) into \( P \).

Example 3. Consider the following program:

\[
\begin{align*}
p(X, Y) & \leftarrow q(X, Y), q(Y, X), \text{let}(X = \theta(X)) \\
q(X, Y) & \leftarrow r(X, Y) \\
r(X, Y) & \leftarrow s(X, Y)
\end{align*}
\]

where \( s \) is an EDB-predicate. Then, moving the literal \( \text{let}(X = \theta(X)) \) one level below results in the following program:

\[
\begin{align*}
p(X, Y) & \leftarrow q(X, Y), q(Y, X) \\
q_1(X, Y) & \leftarrow r_1(X, Y), \text{let}(X = \theta(X)) \\
r_1(X, Y) & \leftarrow s(X, Y) \\
q_2(X, Y) & \leftarrow r_2(X, Y), \text{let}(Y = \theta(X)) \\
r_2(X, Y) & \leftarrow s(X, Y)
\end{align*}
\]
Definition 8 (Bottom-up logical substitution). Let $\theta$ be a substitution and $Q = (L, P)$ be a Nested Datalog query. Then the bottom-up substitution $\theta(Q)$ is the query resulting after applying the top-down substitution of $\theta$ in $Q$ and then moving down the added literals until they reached the leafs of the tree of $P$.

It is not difficult to see that the process of moving a literal down once level results in an equivalent program. Hence, we got the following result.

Lemma 5. Let $Q$ be a Nested Datalog query. Let $\text{ans}^\downarrow$ and $\text{ans}^\uparrow$ be the respective evaluation procedures using top-down and bottom up substitution. Then, for every extensional database $E$ for $Q$ holds $\text{ans}^\downarrow_E(Q) = \text{ans}^\uparrow_E(Q)$. 
