Towards Constraint-based Explanations for Answers and Non-Answers

Boris Glavic
Illinois Institute of Technology
bglavic@iit.edu

Sven Köhler
Google
skoehler80@gmail.com

Sean Riddle
Athenahealth Corporation
swriddle@gmail.com

Bertram Ludäscher
University of Illinois Urbana-Champaign
ludaesch@illinois.edu

Abstract

Explaining why an answer is present (traditional provenance) or absent (why-not provenance) from a query result is important for many use cases. Most existing approaches for positive queries use the existence (or absence) of input data to explain a (missing) answer. However, for realistically-sized databases, these explanations can be very large and, thus, may not be very helpful to a user. In this paper, we argue that logical constraints as a concise description of large (or even infinite) sets of existing or missing inputs can provide a natural way of answering a why- or why-not provenance question. For instance, consider a query that returns the names of all cities which can be reached with at most one transfer via train from Lyon in France. The provenance of a city in the result of this query, say Dijon, will contain a large number of train connections between Lyon and Dijon which each justify the existence of Dijon in the result. If we are aware that Lyon and Dijon are cities in France (e.g., an ontology of geographical locations is available), then we can use this information to generalize the query output and its provenance to provide a more concise explanation of why Dijon is in the result. For instance, we may conclude that all cities in France can be reached from each other through Paris. We demonstrate how an ontology expressed as inclusion dependencies can provide meaningful justifications for answers and non-answers, and we outline how to find a most general such explanation for a given UCQ query result using Datalog. Furthermore, we sketch several variations of this framework derived by considering other types of constraints as well as alternative definitions of explanation and generalization.

Categories and Subject Descriptors H.2.4 [*Relational databases*]

Keywords Provenance, Databases, Missing Answers, Datalog, Ontology

1. Introduction

Explaining why a query does or does not return an answer of interest has been studied extensively as data provenance (the positive case) and the missing answer problem (the negative case). Here we refer to these two types of problems as answering *why* and *why-not* questions, respectively. Since the number of potential explanations for a why-not question is typically very large, many approaches for missing answers (e.g., [HH10, MGMS10]) have resorted to only returning explanations that fulfill a given optimality criterion: e.g., find the smallest set of missing input tuples that if inserted into the database would yield the missing answer. Alternatively, approaches that support provenance of query languages with negation [KLZ13] would enumerate all these explanations. The first approach has the disadvantage that it does not present a full picture of why the answer is missing (e.g., the optimization criterion may not be a good fit for every domain) while the latter approach may return excessive amounts of provenance.

In this paper, we take a different approach by computing concise summaries of explanations for why and why-not questions by generalizing sets of rule derivations (explanations) using logical integrity constraints known to hold for the given schema. In our framework, an explanation is a set of successful or unsuccessful rule derivations which justify the presence or absence of a query result.

We treat why and why-not questions in the same summarization framework. This idea is inspired by the observation from [KLZ13, RKL14], which demonstrated that why and why-not questions can be treated uniformly in a provenance framework that supports full negation (such as the provenance games introduced in that work). Explaining why a tuple t is missing from the result of a query Q is equivalent to explaining why t is in the result of the negation of Q. Consider a relation $\texttt{Train}(X, Y)$ which contains direct train connections from a city X to a city Y. For instance, in Datalog, the negation of a query Q returning cities with at least one outbound train connection

$$\texttt{Q}(X) \text{:-} \texttt{Train}(X, Y).$$

can be expressed as

$$\texttt{Qneg}(X) \text{:-} \texttt{adom}(X), \texttt{adom}(Y), \neg\, \texttt{Train}(X, Y).$$

where predicate `adom` contains all values of the active domain of the database instance. Our framework treats why and why-not questions uniformly in terms of generalizing explanations. The only difference between the two types of questions is what constitutes an explanation: an explanation for a why question is a set of successful rule derivations, each justifying the answer, while an explanation for a why-not question is a set of unsuccessful rule derivations that together justify that the answer is not in the result.

In this work we use inclusion dependencies that express a simple ontology rooted in the constants of a database instance to generalize explanations. For example, to explain why there are no train connections with at most one transfer between Chicago and Berlin we may list all hypothetical train connections between Chicago and Berlin with one intermediate stop that could (but do not) exist. Instead of listing all combinations, we may explain this absence of a train connection by the higher-level rule that there are no train connections between North America and Europe. In this work, we use ontologies to find such more general explanations for UCQ queries. The idea of using an ontology for summarizing explanations has also been applied by ten Cate *et al.* [tCCST15] and Wang *et al.* [WDM15].[1] The major difference between these approaches and ours is that we summarize the successful or failed derivation of

[1] Sven Köhler, Sean Riddle, and Bertram Ludäscher explored different approaches to why-not provenance, including provenance games [KLZ13, RKL14] and using (integrity) constraints. In particular, Sean Riddle proposed to use the latter to produce higher-level explanations for why-not

r_1 : max2Hop(X,Y) :– max1Hop(X,Y).
r_2 : max2Hop(X,Y) :– max1Hop(X,Z), max1Hop(Z,Y).
r_3 : max1Hop(X,X) :– City(X,Y,Z).
r_4 : max1Hop(X,Y) :– Train(Y,X).
r_4 : max1Hop(X,Y) :– Train(X,Y).

Train

fromCity	toCity
new york	washington dc
new york	chicago
chicago	seattle
washington dc	seattle
berlin	paris
berlin	munich
paris	lyon
paris	dijon

City

city	state	country
new york	nystate	usa
chicago	illlinois	usa
washington dc	dc	usa
seattle	washington	usa
lyon	-	france
dijon	-	france
paris	-	france
berlin	-	germany
munich	-	germany

Figure 1: Example Database and Views

a query result (i.e., its provenance) while they aim to summarize a subset of the missing answers to a query. By including query structure into the explanations we can unearth causes for answers and non-answers that cannot be detected by approaches that are query agnostic [tCCST15]. Thus, in a sense we are combining missing answers and provenance techniques with generalization based on ontologies.

EXAMPLE 1 (Why-not Question). *Consider a relation* Train *storing train connections between cities (Figure 1), as well as a view* max2Hop *that determines which cities are reachable from each other with at most one transfer. Here we assume that train connections are bi-directional and use the convention that a city is reachable from itself by not taking any train (see the rules defining* max1Hop *in Figure 1). Assume that a user wants to understand why there is no train connection between Chicago and Berlin, i.e., why the following tuple is missing:*

max2Hop(chicago, berlin)

Note that the type of a user question (why or why-not) is determined based on whether the question tuple is missing or not. We can explain this missing answer by enumerating all potential instantiations of rules for max2Hop *that, if one of them had been successful, would have derived the missing answer (with head* max2Hop(chicago, berlin)*), but failed because one or more grounded atoms from the body did not exist in the instance. For example, one such failed rule derivation is*

r_1 : max2Hop(chicago, berlin) :– max1Hop(chicago, berlin)

That is, there is no direct train connection between Chicago and Berlin. This rule derivation together with all other such unsuccessful rule derivations explains *the failure to derive* max2Hop(chicago, berlin). *Note that a rule derivation fails if at least one goal in its body fails. For rules with more than one goal, we only report the failed goals in a failed rule derivation (replacing successful goals with a placeholder* ⊤*), because only these goals caused the derivation to fail. For instance, another failed rule derivation (rule* r_2*) for the example question is*

r_2 : max2Hop(chicago, berlin) :– max1Hop(chicago, seattle), max1Hop(seattle, berlin)

In this rule derivation the first goal succeeds (Seattle can be reached with at most one train connection from Chicago), but the second goal fails. Thus, when explaining the missing connection between Chicago and Berlin we would report the rule derivation as:

r_2 : max2Hop(chicago, berlin) :– ⊤, max1Hop(seattle, berlin)

2. Basic and Generalized Explanations

We call the set of all such failed rule derivations an *explanation* (for why questions we consider successful rule derivations). In the following we limit the exposition to UCQ queries. For programs like the one shown in Figure 1 we allow the user to pose questions over predicate max2Hop by materializing max1Hop and treating it as a base relation. A generalization of the approach to FO queries requires more advanced provenance techniques (e.g., [KLS12, KLZ13]) to determine successful or failed rule derivations relevant for a user question. Our approach generalizes to FO queries, but due to space limitations we do not present the required techniques here.[2]

DEFINITION 1 (Explanation). *Let Q be a UCQ expressed in Datalog with head predicate $Q(\vec{X})$ and I be a database instance. We support user questions of the form $Q(\vec{c})$ where $\vec{c}$ is a list of constants with the same arity as predicate Q. The type of question (why or why-not) is automatically inferred based on whether $Q(\vec{c})$ exists or not. The explanation* EXPL$(Q(\vec{c}), I)$ *for a why question $Q(\vec{c})$ is the set of all a substitutions of the variables in a rule r from Q with constants such that $Q(\vec{c})$ is the rule head and the body of the rule is fulfilled in I. Similarly, the explanation for a why-not question $Q(\vec{c})$ is the same type of rule derivation except that the body is not fulfilled in I and we replace successful subgoals with* ⊤.

The difference between explanations for why and why-not questions is that the former are successful rule derivations whereas the latter are failed rule derivations restricted to failed goals.[3] As mentioned before, an explanation may be very large, in particular for why-not questions over large instances. For example, the failed derivations of rule r_2 for the example question are all derivations max2Hop(chicago, berlin) :– max1Hop(chicago, c), max1Hop(c, berlin) for any constant c from the active domain. The explanation for the example question contains one entry for each such derivation where each successful goal is replaced with ⊤.

EXAMPLE 2 (Using Constraints to Generalize Explanations). *Assume an ontology of cities is available as a set of inclusion dependencies describing the subsumption relationships between concepts and mapping constants from the activate domain of the instance to the concepts they belong too. Figure 2 shows such an ontology for our running example and a set of Datalog rules defining subsumption relationships. Note that here we treat every constant in the active domain as a concept. For instance, according to this ontology,* paris *belongs to concept of* FrenchCity *and* FrenchCity *is subsumed by the concept* EuropeanCity. *Based on this ontology*

questions and came up with the "continent disconnect implies train disconnect" example used here. Bertram Ludäscher discussed IC-based provenance with Balder ten Cate, who acknowledged Bertram [tCCST15], who hereby acknowledges Sean Riddle.

[2] An explanation for an FO query resembles the connected component of a provenance game graph [KLS12] which contains the user question.

[3] Indeed, basic explanations in our model are a specialized form of provenance games and in turn provenance games [RKL14, KLZ13] can be understood as a form of SLD(NF) resolution. Thus, it is not surprising that a uniform treatment of why- and why-not is possible in our framework.

$$\texttt{ACity}(X) :- \texttt{NACity}(X).$$
$$\texttt{ACity}(X) :- \texttt{EuropeanCity}(X).$$
$$\texttt{NACity}(X) :- \texttt{USCity}(X).$$
$$\texttt{EuropeanCity}(X) :- \texttt{GermanCity}(X).$$
$$\texttt{EuropeanCity}(X) :- \texttt{FrenchCity}(X).$$
$$\texttt{USCity}(X) :- \texttt{IllinoisCity}(X).$$
$$\texttt{USCity}(X) :- \texttt{WashingtonCity}(X).$$
$$\texttt{USCity}(X) :- \texttt{DCCity}(X).$$
$$\texttt{USCity}(X) :- \texttt{NYStateCity}(X).$$
$$\texttt{IllinoisCity}(X) :- \texttt{City}(X, \text{illinois}, Z).$$
$$\texttt{WashingtonCity}(X) :- \texttt{City}(X, \text{washington}, Z).$$
$$\texttt{DCCity}(X) :- \texttt{City}(X, \text{dc}, Z).$$
$$\texttt{NYStateCity}(X) :- \texttt{City}(X, \text{nystate}, Z).$$
$$\texttt{GermanCity}(X) :- \texttt{City}(X, Y, \text{germany}).$$
$$\texttt{FrenchCity}(X) :- \texttt{City}(X, Y, \text{france}).$$

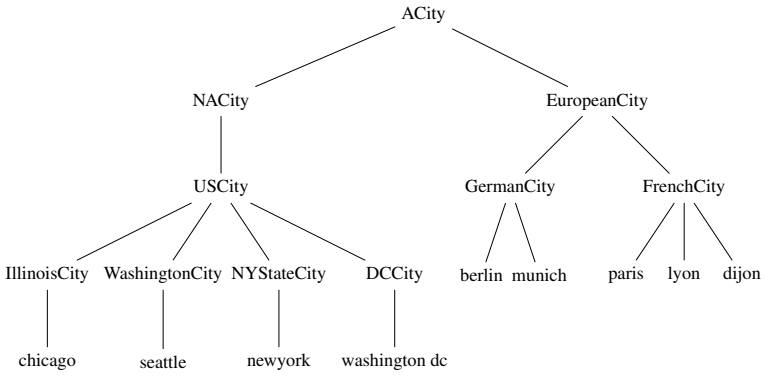

Figure 2: Example Ontology and Mapping

we can compactly describe parts of an explanation for a question using rule derivations with concepts instead of constants. Such a generalized explanation *represents the set of all rule derivations with constants belonging to these concepts. For instance, one explanation for the previous example is that there is no direct connection between any city in Illinois and Berlin:*

$$\texttt{max2Hop}(\texttt{IllinoisCity}, \text{berlin}) :- \texttt{max1Hop}(\texttt{IllinoisCity}, \text{berlin})$$

Generalization of explanations can be defined in several ways (see Section 5 alternative notions). Here we settle for explanations that generalize the user question (the rule heads) and where the rule derivations are subsumed by the explanation for a question, i.e., for each head $Q(\vec{d})$ covered by the generalized explanation, the set of rule derivations with head $Q(\vec{d})$ covered by the explanation are a subset of $\text{EXPL}(Q(\vec{d}), I)$. Some generalized explanations are more specific than others and a natural question that arises is the search for a generalized explanation that is most general in some sense. Next, we first define ontologies rooted in a database instance and then define explanations and subsumption of explanations.

DEFINITION 2 (Ontologies over Instances). *Let $\mathcal{O}$ be an ontology over constants from an instance I, i.e., a pair $(\mathcal{C}, \sqsubseteq)$ where $\mathcal{C}$ is a set of concepts and $\sqsubseteq \subseteq \mathcal{C} \times \mathcal{C}$ is a subsumption relation (reflexive and transitive). We use $\sqsubset$ to denote the strict version of $\sqsubseteq$, i.e., $\sqsubset (C_1, C_2)$ holds if $C_1 \sqsubseteq C_2$ and $C_2 \not\sqsubseteq C_1$. We require that every constant c in $\texttt{adom}(I)$ (the active domain of instance I) is included in $\mathcal{C}$ and that the concept corresponding to a constant c does not subsume any other concepts: $c' \sqsubseteq c \rightarrow c = c'$.*

Figure 2 shows an ontology that is defined by a set of Datalog rules. Using this notion of ontologies mapped to an instance we can define explanations and reason over what it means for an explanation to generalize another explanation.

DEFINITION 3 (Generalized Explanation). *Consider a UCQ Q, instance I, and ontology $\mathcal{O}$. A generalized explanation E for a question $Q(\vec{c})$ using rule r from Q is a substitution $r[\vec{X} \leftarrow \vec{C}]$ of the variables of rule r with concepts from $\mathcal{C}$ and optionally a substitution of a goal with $\top$ such that the conditions shown below hold. We say that a rule derivation $r[\vec{X} \leftarrow \vec{d}]$ is* covered *by a generalized explanation if the constants from $\vec{d}$ are subsumed by the concepts from $\vec{C}$.*

1. *the concepts assigned to the head of r subsume the concepts from $\vec{c}$*
2. *for every vector $\vec{d}$ of constants from $\texttt{adom}(I)$ subsumed by $\vec{C}$ we have for $head(\vec{d})$ (the constants $\vec{d}$ restricted to the head of r)*
   (a) $Q(head(\vec{d}))$ exists iff $Q(\vec{c})$ exists
   (b) $r[\vec{X} \leftarrow \vec{d}]$ is contained in the explanation for $\texttt{Q}(head(\vec{d}))$
3. *A subgoal of r is failed in all rule derivations covered by $\vec{C}$ iff it is replaced with $\top$ in E.*

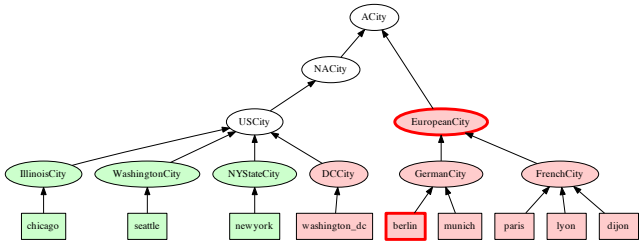

Figure 3: Generalization Process for Question $\texttt{Q}(\text{berlin})$

Intuitively, the above definition states that an generalized explanation is derived from a rule derivation for a question by generalizing the constants mentioned in the question using the ontology. Furthermore, the rule derivations covered by a generalized explanation should be contained in a basic explanation. These conditions ensure that an explanation 1) generalizes the user question and 2) summarizes parts of explanations.

EXAMPLE 3. *We now further illustrate the inner workings of generalization using an unary query $\texttt{Q}(X) :- \texttt{max1Hop}(\text{chicago}, X)$. This query returns all cities that can be reached with at most one train connection from Chicago. Consider a user question $\texttt{Q}(\text{berlin})$, i.e., why can Berlin not be reached from Chicago. The basic explanation to this question is the failed rule derivation $\texttt{Q}(\text{berlin}) :- \texttt{max1Hop}(\text{chicago}, \text{berlin})$. To generalize this explanation we move up in the ontology to find concepts that subsume Berlin and only subsume cities missing from the query re-*

sult. Figue 3 shows the ontology from the previous example. We have colored all constants (boxes) belonging to the query result in green and all missing answers as red. Concepts are colored red if they only subsume missing answers, green if they only subsume existing answers, and white if they subsume both missing and existing answers. In the example, the generalized explanation $\texttt{Q}(\texttt{GermanCity}) :- \texttt{max1Hop}(\text{chicago}, \texttt{GermanCity})$ *which uses concept* `GermanCity` *generalizes the explanation for* $\texttt{Q}(\text{berlin})$*, because* `GermanCity` *subsumes* berlin *and subsumes only missing answers. However, concept* `EuropeanCity` *provides us with a more general explanation since it subsumes* `GermanCity` *and also can be used to create a generalized explanation for this question. In fact, for the given instance and ontology,* `EuropeanCity` *is most general.* `ACity`*, the only concept that subsumes* `EuropeanCity`*, can not be used to explain the question because it subsumes existing answers (e.g.,* seattle*). In our framework, a generalized explanation is sound, but not necessarily complete. That is, it is a compact representation of a subset of the rule derivations from a basic explanation. For instance, the explanation for* $\texttt{Q}(\text{berlin})$ *contains a rule derivation for* washington dc *which is not covered by the generalized explanation using* `EuropeanCity`*. In Section 5 we will discuss alternative definitions of generalization, some of which are sound and complete.*

Note that, since we include constants from the instance into the ontology, generalized explanations may contain constants. For instance,

$$E_1 = \texttt{max2Hop}(\texttt{IllinoisCity}, \text{berlin}) :- \texttt{max1Hop}(\texttt{IllinoisCity}, \text{berlin})$$

is a generalized explanation for the running example question, because its concepts subsume $(\text{chicago}, \text{berlin})$ and for any instance c of `IllinoisCity`, $\texttt{max2Hop}(\text{c}, \text{berlin}) :- \texttt{max1Hop}(\text{c}, \text{berlin})$ belongs to the explanation for $\texttt{max2Hop}(\text{c}, \text{berlin})$. While E_1 is more general than any generalized explanation using only constants, there exist other explanations that are more general than E_1. We next define the notion of a most general explanation.

DEFINITION 4 (Most General Explanation). *Given two generalized explanations E with concepts $C_1, \ldots, C_n$ and E' with concepts $C'_1, \ldots, C'_n$ for a question $Q(\vec{c})$, we say that E dominates E' written as $E > E'$ iff*

$$\forall i \in \{1, \ldots, n\} : C_i \sqsupseteq C'_i$$

and

$$\exists i \in \{1, \ldots, n\} : C_i \sqsupset C'_i$$

A most general explanation for a question is a generalized explanation that is not dominated by any other generalized explanation for this question.

From now on we will use $r(\vec{C})_{p\vec{o}s}$ to denote a generalized explanation using rule r where the concepts from $\vec{C}$ are bound to the variables of r and the goals at the positions in list $p\vec{o}s$ are replaced with $\top$.

EXAMPLE 4 (Most General Explanation). *A most general explanation for question* $\texttt{max2Hop}(\text{chicago}, \text{berlin})$ *is*

$$r_2(\texttt{NYStateCity}, \texttt{EuropeanCity}, \texttt{WashingtonCity})$$

That is, no city in Europe can be reached from any city in New York state with one intermediate stop in a city in Washington state, because there are no direct train connections from both New York and European cities to any Washington city. A most general explanation for rule r_1 is $r_1(\texttt{NACity}, \texttt{EuropeanCity})$, i.e., there are no direct train connections between North American and European cities.

The example below demonstrates how our approach can be used to answer why questions.

EXAMPLE 5 (Why-Question). *As an example for answering why questions consider the explanations for why Lyon can be reached with at most one transfer from Dijon (two cities in France):*

$$\texttt{max2Hop}(\text{dijon}, \text{lyon})$$

The explanation for this question contains a single derivation $r_2(\text{dijon}, \text{lyon}, \text{paris})$, i.e., we can reach Lyon from Dijon by transferring in Paris. Concepts that subsume lyon*,* dijon*, and* paris *are* `FrenchCity`*,* `EuropeanCity` *and* `ACity`*. Assigning these concepts to variables in r_2 we derive candidate explanations. For each such candidate we check whether it fulfills the conditions of Definition 3. The result of this process are the following explanations:*

$$r_2(\texttt{FrenchCity}, \text{lyon}, \text{paris})$$
$$r_2(\text{dijon}, \texttt{FrenchCity}, \text{paris})$$
$$r_2(\texttt{FrenchCity}, \texttt{FrenchCity}, \text{paris})$$

Among these explanations only $r_2(\texttt{FrenchCity}, \texttt{FrenchCity}, \text{paris})$ is a most general explanation, because lyon *and* dijon *are both subsumed by the concept* `FrenchCity`*. Intuitively, the meaning of this explanation is that in the given database instance I we can reach any city in France from any other city in France through Paris.*

So far we have only shown one most general explanation for each rule. However, in general there may exist several most general explanations for a rule, even if all pairs of concepts in the ontology that are incomparable according to subsumption have no overlap.

EXAMPLE 6 (Multiple Most General Explanations). *Consider the question* $\texttt{max2Hop}(\text{newyork}, \text{seattle})$*. In the given database instance there are two train connections with at most one transfer that connect New York to Seattle: we have to either transfer in Chicago or in Washington DC. We get two most general explanations:*

$$r_2(\texttt{NYStateCity}, \texttt{WashingtonCity}, \texttt{IllinoisCity})$$
$$r_2(\texttt{NYStateCity}, \texttt{WashingtonCity}, \texttt{DCCity})$$

3. Computing Explanations With Datalog

We now illustrate how to compute the most general explanations for a question using Datalog. As long as the subsumption relation is materialized or can be computed with Datalog, we can use a Datalog program without recursion (FO query) to compute the most general explanations for a question over a UCQ.[4] Our approach generates such a Datalog program as follows.

Modelling the Ontology. We introduce view predicates `isConcept`, `subsumes`, and `subsumesEqual` to model the ontology. Some of the facts and rules defining these predicates for our running example are shown below. We create a fact $\texttt{isConcept}(C)$ for every concept C in the ontology and introduce rules for creating concepts representing the constants in the active domain. A subsumption rule is introduced for every inclusion dependency of the ontology. We then compute the transitive closure of subsumption. Finally, we define a predicate that models $\sqsubseteq$.

[4] We provide implementations of the examples from this paper as a git repository: https://bitbucket.org/bglavic/tapp-2015-towards-constraint-explanations-datalog-snippets

$$\texttt{has\_r}_2\texttt{\_XX}(C_1,C_2,C_3,G_1,G_2) :- \texttt{subsumesEqual}(B_1,C_1), \texttt{subsumesEqual}(B_2,C_2), \texttt{subsumesEqual}(B_3,C_3), \texttt{r}_2\texttt{\_der}(B_1,B_2,B_3,G_1,G_2).$$
$$\texttt{has\_r}_2\texttt{\_not\_00}(C_1,C_2,C_3) :- \texttt{has\_r}_2\texttt{\_not\_XX}(C_1,C_2,C_3,\mathit{true},G_2).$$
$$\texttt{has\_r}_2\texttt{\_not\_00}(C_1,C_2,C_3) :- \texttt{has\_r}_2\texttt{\_not\_XX}(C_1,C_2,C_3,G_1,\mathit{true}).$$
$$\texttt{has\_max2Hop\_success}(C_1,C_2) :- \texttt{subsumesEqual}(B_1,C_1), \texttt{subsumesEqual}(B_2,C_2), \texttt{max2Hop}(B_1,B_2).$$
$$\texttt{expl\_r}_2\texttt{\_fail\_00}(C_1,C_2,C_3,B_1,B_2) :- \texttt{subsumesEqual}(B_1,C_1), \texttt{subsumesEqual}(B_2,C_2), \texttt{subsumesEqual}(B_3,C_3), \texttt{r}_2\texttt{\_fail}(B_1,B_2,B_3), \neg\, \texttt{has\_r}_2\texttt{\_not\_00}(C_1,C_2,C_3), \neg\, \texttt{has\_max2Hop\_success}(C_1,C_2).$$
$$\texttt{dominated\_r}_2\texttt{\_fail\_00}(C_1,C_2,C_3,B_1,B_2) :- \texttt{expl\_r}_2\texttt{\_fail\_00}(C_1,C_2,C_3,B_1,B_2), \texttt{expl\_r}_2\texttt{\_fail\_00}(C_1',C_2',C_3',B_1,B_2), \texttt{dominates}_3(C_1,C_2,C_3,C_1',C_2',C_3').$$
$$\texttt{most\_gen\_r}_2\texttt{\_fail\_00}(C_1,C_2,C_3,B_1,B_2) :- \texttt{expl\_r}_2\texttt{\_fail\_00}(C_1,C_2,C_3,B_1,B_2), \neg\, \texttt{dominated\_r}_2\texttt{\_fail\_00}(C_1,C_2,C_3,B_1,B_2).$$
$$\texttt{whynot\_r}_2\texttt{\_00}(C_1,C_2,C_3) :- \texttt{most\_gen\_r}_2\texttt{\_fail\_00}(C_1,C_2,C_3,\text{chicago},\text{berlin}).$$

Figure 4: Rules for Computing the Most General Explanations for $\texttt{max2Hop}(\text{chicago},\text{berlin})$

$$\texttt{isBasicConcept}(X) :- \texttt{city}(X,Y,Z).$$
$$\texttt{isConcept}(X) :- \texttt{isBasicConcept}(X).$$
$$\texttt{isConcept}(\text{acity}).$$
$$\texttt{isConcept}(\text{nacity}).$$
$$\ldots$$
$$\texttt{isConcept}(X) :- \texttt{city}(X,Y,Z).$$
$$\ldots$$

$$\texttt{subsumes}(\text{nacity},\text{acity}).$$
$$\texttt{subsumes}(X,\text{illinoiscity}) :- \texttt{City}(X,\text{illinois},Z).$$
$$\ldots$$
$$\texttt{subsumes}(X,Y) :- \texttt{subsumes}(X,Z), \texttt{subsumes}(Z,Y).$$
$$\texttt{subsumesEqual}(X,X) :- \texttt{isConcept}(X).$$
$$\texttt{subsumesEqual}(X,Y) :- \texttt{subsumes}(X,Y).$$

Furthermore, we introduce predicates for checking whether a list of concepts is dominated by another list of concepts. We introduce separate predicates for each possible list length that may occur (determined by the input program). The rules for lists of two concepts are shown below. Each of these rules represents the situation where one of the subsumption relationships between concepts is strict whereas the others may or may not be strict. Thus, we need n rules to define $\texttt{dominates}_\texttt{n}$.

$$\texttt{dominates}_2(X,Y,Z,A) :- \texttt{subsumes}(X,Z), \texttt{subsumesEqual}(Y,A).$$
$$\texttt{dominates}_2(X,Y,Z,A) :- \texttt{subsumesEqual}(X,Z), \texttt{subsumes}(Y,A).$$

Modelling Rule Success and Failure. For each rule related to the user question (can be determined through static analysis of the program) we add rules returning successful and failed derivations for this rule. These rules compute basic explanations. The success rule for a rule r has the same body as r, the head predicate is named $\texttt{r\_success}$, and the head contains all variables from rule r. Thus, a success rule for rule r fires iff rule r fires and returns the bindings of all variables for all successful rule derivations. A fail rule for rule r checks that each variable is bound to an existing basic concept (a concept that exists as a constant in the instance) and checks that the rule r did not fire for this derivation (using the negation of $\texttt{r\_success}$). For instance, for rule r_2 we generate:

$$\texttt{r}_2\texttt{\_success}(X,Y,Z) :- \texttt{Train}(X,Z), \texttt{Train}(Z,Y).$$
$$\texttt{r}_2\texttt{\_fail}(X,Y,Z) :- \texttt{isBasicConcept}(X), \texttt{isBasicConcept}(Y), \texttt{isBasicConcept}(Z), \neg\, \texttt{r}_2\texttt{\_success}(X,Y,Z).$$

These types of rules have been successfully applied for Datalog debugging [KLS12] and computing provenance [KLZ13]. Recall that for failed rule derivations we need to know which goals have failed to be able to compute explanations. Instead of using the rules introduced above we use a predicate $\texttt{r\_der}$ with a head that has all variables from rule r plus one variable for each goal in the body that models whether this goal is fulfilled for a rule derivation. Below we show these rules for rule r_2 from the running example. For instance, the third rule returns derivations of rule r_2 where the first goal failed while the second goal succeeded.

$$\texttt{r}_2\texttt{\_der}(X,Y,Z,\text{true},\text{true}) :- \texttt{max1Hop}(X,Z), \texttt{max1Hop}(Z,Y).$$
$$\texttt{r}_2\texttt{\_der}(X,Y,Z,\text{true},\text{false}) :- \texttt{isBasicConcept}(Y), \texttt{max1Hop}(X,Z), \neg\, \texttt{max1Hop}(Z,Y).$$
$$\texttt{r}_2\texttt{\_der}(X,Y,Z,\text{false},\text{true}) :- \texttt{isBasicConcept}(X), \neg\, \texttt{max1Hop}(X,Z), \texttt{max1Hop}(Z,Y).$$
$$\texttt{r}_2\texttt{\_der}(X,Y,Z,\text{false},\text{false}) :- \texttt{isBasicConcept}(X), \texttt{isBasicConcept}(Y), \texttt{isBasicConcept}(Z), \neg\, \texttt{max1Hop}(X,Z), \neg\, \texttt{max1Hop}(Z,Y).$$

For example, in the example instance there is a direct train connection from Paris to Dijon, but not from Dijon to Berlin. Thus, the derivation

$$r_2 : \texttt{max2Hop}(\text{paris},\text{berlin}) :- \texttt{max1Hop}(\text{paris},\text{dijon}), \texttt{max1Hop}(\text{dijon},\text{berlin})$$

fails because subgoal $\texttt{max1Hop}(\text{dijon},\text{berlin})$ fails (while subgoal $\texttt{max1Hop}(\text{dijon},\text{berlin})$ succeeds). Based on this rule deriva-

tion, tuple $\texttt{r}_2\texttt{\_der}(\text{paris}, \text{berlin}, \text{true}, \text{false})$ will be derived by the second rule shown above.

Modelling Explanations and Generalization. In this step we find the most general explanations for each rule related to the user question (and for each combination of failed subgoals in case of why-not questions). We explain this last step for example question $\texttt{max2Hop}(\text{chicago}, \text{berlin})$ using rule r_2 with both subgoals failed. The generated rules are shown in Figure 4.

The first four rules are helper rules that test the existence of certain types of rule derivations covered by a list of concepts. Rule $(\texttt{has\_r}_2\texttt{\_XX}(C_1, C_2, C_3))$ checks whether a rule derivation exists for constants (B_1, B_2, B_3) belonging to a combination (C_1, C_2, C_3) of concepts such that the goals are successful/failed as indicated by G_1 and G_2. For instance, $\texttt{has\_r}_2\texttt{\_XX}(C_1, C_2, C_3, \text{true}, \text{true})$ denotes that there exists a derivation of rule r_2 for constants (B_1, B_2, B_3) which belong to concepts (C_1, C_2, C_3) such that both goals are won. The following two rules define a predicate $\texttt{has\_r}_2\texttt{\_not\_00}$ that returns combinations of concepts (C_1, C_2, C_3) for which there exists constants (B_1, B_2, B_3) contained by these concepts such that in the rule derivation $r_2(B_1, B_2, B_3)$ not both goals have failed. Finally, $\texttt{has\_max2Hop\_success}(C_1, C_2)$ computes which pair of concepts cover at least one existing tuple in $\texttt{max2Hop}$.

The rule with head $\texttt{expl\_r}_2\texttt{\_fail\_00}(C_1, C_2, C_3, B_1, B_2)$ finds concepts (C_1, C_2, C_3) that form an explanation for a question $\texttt{max2Hop}(B_1, B_2)$ using rule r_2 with both goals failed (indicated by suffix 00 in the predicate name). By binding B_1 and B_2 to the constants from an input why-not question, we can compute all such explanations for this question. The two conditions in the definition of explanation are checked by 1) checking that the concepts (B_1, B_2) are subsumed by the concepts (C_1, C_2), 2a) by checking that these concepts do not cover a rule derivation where at least one of the goals succeeds, and 2b) by checking that the concepts for the head of r_2 (C_1 and C_2) do not cover an existing $\texttt{max2Hop}$ tuple. Note that both universal checks of the definition (for every constant belonging to the concepts, the rule derivation fails; and only non-existing tuples are covered) are encoded as a doubly negated existential condition (e.g., there are *no* combinations of constants belonging to the concepts for which *not* both goals have failed in the corresponding rule derivation).

Now that we have the means to compute the set of explanations for a question, we can proceed to compute the most general such explanation. The rule $\texttt{dominated\_r}_2\texttt{\_fail\_00}(C_1, C_2, C_3, B_1, B_2)$ checks whether for an explanation (C_1, C_2, C_3) for a missing answer (B_1, B_2) there exists a more general explanation (C_1', C_2', C_3'). The rule $\texttt{most\_gen\_r}_2\texttt{\_fail\_00}(C_1, C_2, C_3, B_1, B_2)$ then checks whether an explanation (C_1, C_2, C_3) is a most general explanation for a question $\texttt{max2Hop}(B_1, B_2)$, i.e., an explanation is most general if it is not dominated by any other explanation. Finally, the last rule returns all most general explanations for $\texttt{max2Hop}(\text{chicago}, \text{berlin})$. These rules can be adapted automatically to another rule r' and different combination of failed subgoals by varying the number of arguments and changing predicate names.

A few remarks are in order. We can use this Datalog program to compute the answers to any question $\texttt{max2Hop}(c_1, c_2)$ by replacing chicago with c_1 and berlin with c_2 in the last rule. An analog program can be derived for a why question by switching rule derivation failure and success in the rules. For example, an explanation for a why question is a list of concepts that dominate a successful rule derivation and there is no list of constants subsumed by the concepts for which the rule derivation fails.

4. Comparison with Existing Techniques

The approach from ten Cate *et al.* [tCCST15] also finds explanations to a missing answer (and could be easily adapted to support why questions by inverting the conditions in their definition of explanation) through generalization based on concept subsumption. However, this approach generalizes missing query results and does not provide any information on what happened within the query, i.e., it is not taking the provenance of the missing answer into account. The example below shows that our approach can provide more insightful explanations by considering the query's provenance.

EXAMPLE 7. *Reconsider our example question* $\texttt{max2Hop}(\text{dijon}, \text{lyon})$. *The approach from ten Cate* et al. *would return the concept pair* $(\texttt{FrenchCity}, \texttt{FrenchCity})$ *by generalizing the view relation* $\texttt{max2Hop}$*, i.e., all french cities are reachable from each other. However, a more insightful explanation is that all french cities are connected through Paris which corresponds to our most general explanation* $r_2(\texttt{FenchCity}, \texttt{FrenchCity}, \text{paris})$.

One novel insight in ten Cate *et al.*'s [tCCST15] work is that queries can be used to define the concepts of an ontology. This is useful for applications where no ontology for the domain is available or where the ontology is not useful for a particular query. That is, there are combinations of queries and ontologies for which an explanation cannot be compactly described by the ontology. For instance, our geographical ontology is useless for summarizing explanations for a query that selects cities by the size of their population. In the worst-case our approach would return most general explanations that are single rule derivations. It would be interesting to apply such query-based ontologies to our problem. This idea is also closely related to data summarization techniques using queries [EGAG+14] and Roy *et al.*'s [RS14] approach for explaining aggregate query results. Subsumption has been used as a technique for including semantic knowledge in query evaluation [KG94]. In this framework, results to Datalog programs are reduced based on subsumption relationships between facts provided by the user. For example, a tuple $(\text{a}, \text{b}, \text{co})$ representing a path between a and b of cost co may be subsumed by a tuple $(\text{a}, \text{b}, \text{co}')$ if $\text{co} < \text{co}'$. This work demonstrated that reduction based on subsumption can be applied during query evaluation. It would be interesting to investigate whether this idea can be used to efficiently compute generalized explanations.

5. Discussion

So far we have used one particular type of constraint (inclusion dependencies encoding an ontology) to generalize explanations and have limited the discussion to one possible notion of explanation. Under this definition the concepts of an explanation cover a subset of all rule derivations from a basic explanation and are not allowed to cover anything else. Obviously, several meaningful variations of this problem definition exist.

Explanations Based on other Constraint Types. Inclusion dependencies are only one type of constraint that can be used to explain a why or why-not question. As the following example shows, we may also be able to answer such questions using other types of constraints.

EXAMPLE 8. *Consider a base relation*

$$\texttt{President}(Name, Citizenship, Country)$$

recording presidents, their citizenship, and the country they were president of. The following denial constraint[5] *has been de-*

[5] Denial constraints are used to encode patterns that should not occur, i.e., a denial constraint is violated if it's body is fulfilled.

fined to encode that a person who is a US president has to have US citizenship:

$$:\!-\texttt{President}(X,Y,Z), Y \neq \text{us}, Z = \text{us}.$$

Assume the user is interested in why her query

$$\texttt{Q}(X,Y) :\!-\texttt{President}(X,Y,Z), Z = \text{us}.$$

does not return any US presidents that are not US citizens: $\texttt{Q}(X,Y), Y \neq \text{us}$. *Ignoring the constraint, missing answer approaches would either enumerate all relevant* `President` *tuples that are missing from the input or try to find one solution that fulfills some optimization goal (e.g., smallest side-effect on the view predicates). However, a more concise explanation is that the constraint prevents any tuples from being returned by the query.*

Aside from motivating the use of different types of constraints for answering why or why-not questions, the example above also points to the possibility of finding explanations at the schema level. That is, the constraint explains the missing answer independent of the instance. For that we may draw ideas from semantic query optimization [CGM90]. This type of generalization may also be applicable to inclusion dependencies. For instance, we could recast the denial constraint above as an inclusion dependency: the concept US presidents is subsumed by the concept US citizens.

Alternative Definitions of Generalization. We have defined a most general explanation as a combination of concepts that dominate a subset of $\textsc{Expl}(Q(\vec{c}), I)$ for a question $Q(\vec{c})$ and where none of these concepts can be generalized further without breaking this inclusion condition. Alternatively, we could define a most general explanation as an explanation such that no explanation exists that covers a larger subset of $\textsc{Expl}(Q(\vec{c}), I)$. Another meaningful way of defining most general explanations is to relax the condition that an explanation can only cover rule derivations from basic explanations and search for an explanation that minimizes the symmetric difference between all its covered rule derivations and $\textsc{Expl}(Q(\vec{c}), I)$. That is, we could allow explanations to cover rule derivations that are not contained in basic explanations, i.e., where the derivation does not correspond to a successful rule firing (for why questions) respective failed rule firing (for why-not questions). For instance, we may consider an explanation $r_2(\texttt{FrenchCity}, \texttt{FrenchCity}, \texttt{FrenchCity})$ for question $\texttt{max2Hop}(\text{dijon}, \text{lyon})$ even though the derivation $r_2(\text{dijon}, \text{lyon}, \text{lyon})$ covered by this explanation is not successful. This type of explanations would be complete, but not necessarily sound. Furthermore, instead of searching for a single explanation we could search for sets of explanations of minimal size that precisely cover a basic explanation (both sound and complete). For example, reconsider the question from Example 6. We may return both explanations shown in this example, because together they cover the basic explanation for this question. Finally, we could search for sets of explanations that minimize a measure based on both the number of explanations in the set and the symmetric difference between the combined coverage of this set and $\textsc{Expl}(Q(\vec{c}), I)$.

6. Conclusions

We have presented a framework for computing generalized explanations for answers and non-answers to Datalog queries. Generalization relies on an existing ontology (modelled as inclusion dependencies) that is mapped to the database instance. Using concepts from the ontology we can compute concise explanations that are easy to understand and compactly represent a potentially large subset of a basic explanation. An interesting avenue of future work is to add support for additional types of constraints (e.g., denial constraints as outlined in Section 5). We have demonstrated how to compute most general explanations using Datalog. In the future we would like to explore extensions of the approach, e.g., finding minimally sized explanations that cover the full set of answers or non-answers. Furthermore, we would like to investigate more efficient techniques for finding explanations, e.g., by materializing concept information or computing explanations independent of instances.

Acknowledgements

Work supported in part through an unrestricted gift from the Oracle corporation and by the NSF under awards IIS-1118088 (Euler), ABI-1147273 (ETC), ACI-0830944 (DataONE).

References

[CGM90] U. S. Chakravarthy, J. Grant, and J. Minker. Logic-based approach to semantic query optimization. *TODS*, 15(2):162–207, 1990.

[EGAG+14] K. El Gebaly, P. Agrawal, L. Golab, F. Korn, and D. Srivastava. Interpretable and informative explanations of outcomes. *PVLDB*, 8(1):61–72, 2014.

[HH10] M. Herschel and M. Hernandez. Explaining Missing Answers to SPJUA Queries. *PVLDB*, pp. 185–196, 2010.

[KG94] W. Kießling and U. Güntzer. Database reasoning—a deductive framework for solving large and complex problems by means of subsumption. In *Management and Processing of Complex Data Structures*, pp. 118–138. Springer, 1994.

[KLS12] S. Köhler, B. Ludäscher, and Y. Smaragdakis. Declarative datalog debugging for mere mortals. *Datalog in Academia and Industry*, pp. 111–122, 2012.

[KLZ13] S. Köhler, B. Ludäscher, and D. Zinn. First-Order Provenance Games. In *In Search of Elegance in the Theory and Practice of Computation*, pp. 382–399. Springer, 2013.

[MGMS10] A. Meliou, W. Gatterbauer, K. Moore, and D. Suciu. The Complexity of Causality and Responsibility for Query Answers and non-Answers. *PVLDB*, 4(1):34–45, 2010.

[RKL14] S. Riddle, S. Köhler, and B. Ludäscher. Towards Constraint Provenance Games. In *USENIX Workshop on the Theory and Practice of Provenance (TaPP)*, 2014.

[RS14] S. Roy and D. Suciu. A formal approach to finding explanations for database queries. In *SIGMOD*, pp. 1579–1590, 2014.

[tCCST15] B. ten Cate, C. Civili, E. Sherkhonov, and W. Tan. High-Level Why-Not Explanations using Ontologies. In *PODS*, pp. 31–43, 2015.

[WDM15] X. Wang, X. L. Dong, and A. Meliou. Data X-Ray: A Diagnostic Tool for Data Errors. In *SIGMOD*, pp. 1231–1245, 2015.