WHY SO? or WHY NO? Functional Causality for Explaining Query Answers

Alexandra Meliou, Wolfgang Gatterbauer, Katherine F. Moore, and Dan Suciu

University of Washington
{ameli,gatter,kfm,suciu}@cs.washington.edu

Abstract. In this paper, we propose *causality* as a unified framework to *explain query answers* and non-answers, thus generalizing and extending several previously proposed definitions of provenance and missing query result explanations.
Starting from the established definition of *actual causes* by Halpern and Pearl [12], we propose *functional causes* as a refined definition of causality with several desirable properties. These properties allow us to apply our notion of causality in a database context and apply it uniformly to define the causes of query results and their individual contributions in several ways: (i) we can model both *provenance* as well as *non-answers*, (ii) we can define explanations as either *data* in the input relations or relational *operations* in a query plan, and (iii) we can give graded degrees of responsibility to individual causes, thus allowing us to *rank causes*. In particular, our approach allows us to explain contributions to relational *aggregate functions* and to rank causes according to their respective responsibilities, aiding users in identifying errors in uncertain or untrusted data. Throughout the paper, we illustrate the applicability of our framework with several examples.
This is the first work that treats "positive" and "negative" provenance under the same framework, and establishes the theoretical foundations of causality theory in a database context.

1 Introduction

When analyzing uncertain data sets, users are often interested in *explanations* for their observations. Explaining the causes of surprising query results allows users to better understand their data, and identify possible errors in data or queries. In a database context, explanations concern results returned by explicit or implicit queries. For example, "Why does my personalized newscast have more than 20 items today?" Or, "Why does my favorite undergrad student not appear on the Dean's list this year?" Database research that addresses these or similar questions is mainly work on *lineage of query results*, such as why [8] or where provenance [3], and very recently, explanations for non-answers [15,4]. While these approaches differ over what the response to questions should be, all of them seem to be linked through a common underlying theme: understanding *causal relationships* in databases.

Humans usually have an intuition about what constitutes a cause of a given effect. In this paper, we define the foundational notion of *functional causality* that can model this intuition in an exact mathematical framework, and show how it can be applied to encode and solve various causality related problems. In particular, it allows us to uniformly model the questions of WHY SO? and WHY NO? with regards to query answers. It also allows us to represent different previous approaches, thus illustrating causality to be a critical element unifying prior work in this field.

N(ewsFeeds)

nid	*story*	*tag*
1	... share lead in Singapore championship ...	Golf
2	... economic downturn affected sensitive ...	Business
3	... with sequences shot in Singapore ...	Movies
4	... when President Obama meets former ally ...	Obama
5	... Singapore slow down hiring ...	Business
6	... Oscars 2010: Academy's 'best' choice ...	Movies
7	... HP launches cloud lab in Singapore ...	Technology
8	... struggles to corral votes for health bill ...	Health
9	... VLDB conference this year in Singapore ...	DB_conf
10	... at the the Indianapolis Motor Speedway ...	Indy 500
11	... Indianapolis host to SIGMOD 2010 ...	DB_conf
12	... VLDB in Singapore promises to be ...	DB_conf
13	... more people in Indianapolis this year ...	Indy_500
14	... Gatorade drops Tiger woods ...	Golf

R(outing)

tag
Obama
DB_conf
Golf
Technology
Health

Query answer:
P(ersonalized alerts)

cities
Paris
Singapore
Athens
Vancouver

Fig. 1: Example of a personalized alert-feed (P) as a result of a query filtering all news (N) based on a carefully constructed routing table (R).

Example 1. *A major travel agency monitors a large number of news feeds in order to identify trends, opportunities, or alerts about various cities. Central to this activity is a carefully personalized routing table and query, which filters what information to forward to each specialized travel agent by carefully chosen keywords. Fig. 1 shows the routing table for one user R, as well as a sample news feed. The query issuing alerts to this user is:*

```
select    C.name
from      NewsFeeds N, Routing R, City C
where     C.name substring N.story and N.tag = R.tag
group by  C.name
having    count(*) > 20
```

The result is a list of cities that are drawn to the attention of this particular agent, shown in Fig. 1. As popular destinations, Paris and Athens are predictable answers, and so is Vancouver because of the recent Olympics. But this agent believes Singapore is an error, and wants to know what entries in the Routing table caused it to appear on her watch list. She wants to ask "Why am I being alerted about Singapore?". The system should answer that the keywords DB_conf*,* technology*, and* golf *are* causes *with various degrees of* responsibility.

As illustrated in Example 1, we want to allow users to ask simple questions based on the results they receive, and hence, allow them to learn what may be the *cause* of any surprising or undesirable answer. Such questions can refer to either presence (WHY SO?) or absence (WHY NO?) of results. Furthermore, the user should be provided with a ranking of causes based on their individual contribution or *responsibility*. Unexpected results are often an indication of errors, and tracking their causes is a crucial step in repairing faulty data, or mistakes in queries. Our ultimate goal is to define a language that allows users to specify causal queries for given results. In this paper, we (i) lay the theoretical groundwork and define a formal model that allows us to capture such causality-related questions in a uniform framework, and (ii) illustrate the applicability of our scheme through various examples.

Summary and outline. We start by reviewing existing work on causality in AI in Section 2, and propose *functional causes* as a refined notion that mitigates problems of existing definitions (Sect. 2.1). In Sect. 3 we highlight several desirable properties of functional causes, which are important for their applicability in a database context. Section 4 gives several examples of applying our framework to give WHY SO? and WHY NO? explanations to *database queries*. We show that our unifying approach generalizes provenance

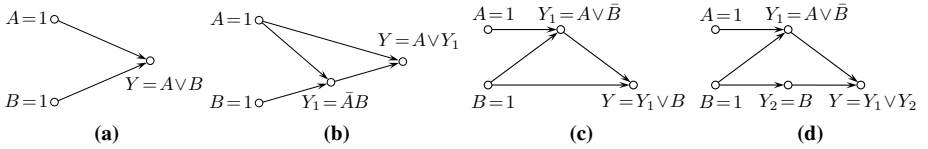

Fig. 2: (a) Alice (A) and Bob (B) each throw a rock at a bottle, which breaks if it gets hit by either rock ($Y = A \vee B$). (b) Alice's throw *preempts* Bob's ($A = 1 \Rightarrow Y_1 = 0$). (c,d) *Expansion* causes problems for the HP definition: Introducing node Y_2, which merely repeats the value of B, does not change the function $Y(\boldsymbol{X})$, but makes A an actual cause.

as well as non-answers (Sect. 4.1), handles contributions to aggregate functions by ranking causes according to their responsibilities for the result (Sect. 4.2), and can also model causes other than tuples (Sect. 4.3).

2 Causality Definitions

Due to space limitations, we briefly overview the two most established definitions of causality from the AI and philosophy literature, and refer the reader to our technical report [20] for more details, discussion of issues and implications, examples, and proofs of all results.

Counterfactual Causes. With a long tradition in philosophy [16], the argument of counterfactual causality is that the relationship between cause and effect can be understood as a counterfactual statement, i.e. an event is considered a cause of an effect if the effect would not have happened in the absence of the event. We focus on the boolean case, and in our notion, the variable assignment (event) $X = x^0$ is a cause of expression ϕ, iff $X = x^0 \wedge \phi$ and $[X \leftarrow \neg x^0] \Rightarrow \neg\phi$. However, counterfactual causality cannot explain causality for slightly more complicated scenarios such as for *disjunctive causes*, i.e. when there are two potential causes of an event.

Actual Causes. The HP definition of causality [12] is based on counterfactuals, but can correctly model disjunction and many other complications. It is the most established definition in the field of structural causality, and relies on the use of a *causal network* (much like a Bayesian network), representing dependencies between variables (e.g. Fig. 2a). In a database context, the variables can be tuples, but they can represent in general any element that may be causally relevant. Every node in the causal network is governed by a *structural equation* that determines the node's assignment based on its input. A causal model is commonly denoted as $M = (\boldsymbol{N}, \mathcal{F})$, where $\boldsymbol{N}$ the set of variables, and $\mathcal{F}$ the set of structural equations. The idea is that X is a cause of Y if Y counterfactually depends on X under "some" *permissive contingency*, where "some" is elaborately defined.[1] The heart of the definition is condition AC2 in [12, Def. 3.1], which is effectively a generalization of counterfactual causes. The requirement is that there exists some assignment of the variables for which $\boldsymbol{X}$ is counterfactual, and that this assignment does not make any fundamental changes to the *causal path* of $\boldsymbol{X}$ (the descendants of $\boldsymbol{X}$ in the causal network). The use of the causal network makes the HP definition very flexible, allowing it to capture different scenarios of causal relationships. For example, it correctly handles disjunctive causes and preemption, i.e. when there are two potential causes of an event and one chronologically *preempts* the other (e.g. Fig. 2b).

The HP definition does however have some limitations which make its application to a database context problematic. In the well studied *Shock C* example (see [22]), actual cau-

[1] Contingencies relate to possible world semantics: "Is there a possible world that makes $\boldsymbol{X}$ counterfactual?"

sality produces unintuitive results; a variable is determined to be a cause of a tautology, which in a data context is semantically spurious. A less known but equally important issue of the definition is its lack of robustness to minor network variations. The addition of "dummy" nodes, which do not affect the function or assignments of other nodes, can change the causality of variables (Fig. 2c,2d). This is problematic in a database setting, where we care about query semantics rather than syntax. We revisit this issue in Sect. 3.1, and also refer the reader to our technical report [20] for an extensive discussion.

2.1 Functional Causes

A fundamental challenge in applying causality to queries is that causality is defined over an entire *network*: it is not enough to know the dependency of the effect on the input variables, we also need to reason about intermediate dependent nodes. This requirement is difficult to carry over to a database setting, where we care about the semantics of a query rather than a particular query plan. Our approach is to represent a causal network with two appropriate functions that *semantically capture* the causal dependencies of a network. The two key notions we need for that are *potential functions* and *dissociation expressions*.

Figure 3 represents a causal network in our framework. In contrast to the HP approach, only input variables from $\boldsymbol{X}$ can be causes and part of permissive contingencies. As in the HP approach, every dependent node Y is described by a structural equation F_Y, which assigns a truth value to Y based on the values of its parents. The *Boolean formula* Φ_Y of Y defines its truth assignment based on the input variables $\boldsymbol{X}$, and is constructed by recursing through the structural equations of Y's ancestors. For example, in Fig. 2b, $\Phi_Y(\boldsymbol{X}) = A \vee (\bar{A} \wedge B)$, where $\boldsymbol{X} = \{A, B\}$. We denote as $\Phi(\boldsymbol{X}) = \Phi_{Y_j}(\boldsymbol{X})$, where Y_j is the effect node, and we say that the causal network has formula Φ. The *potential function* P_Φ is then simply the unique multilinear polynomial representing Φ. It is equal to the probability that Φ is true given the probabilities of its input variables.

Definition 1 (Potential Function). *The potential function $P_\Phi(\boldsymbol{x})$ of a Boolean formula $\Phi(\boldsymbol{X})$ with probabilities $\boldsymbol{x} = \{x_1, \ldots, x_k\}$ of the input variables is defined as follows:*

$$P_\Phi(\boldsymbol{x}) = \sum_{\boldsymbol{\varepsilon} \to \{0,1\}^k} \left(\prod_{i=1}^{k} x_i^{\varepsilon_i} \right) \Phi(\boldsymbol{\varepsilon}), \quad x_i^{\varepsilon_i} = \begin{cases} x_i & \text{if } \varepsilon_i = 1 \\ 1 - x_i & \text{if } \varepsilon_i = 0 \end{cases}$$

The potential function is a sum with one term for each truth assignment $\boldsymbol{\varepsilon}$ of variables $\boldsymbol{X}$. Each term is a product of factors of the form x_i or $1 - x_i$ and only occurs in the sum if the formula is true at the given assignment ($\Phi(\boldsymbol{\varepsilon}) = 1$). For example, if $\Phi = X_1 \wedge (X_2 \vee X_3)$ then $P_\Phi = x_1x_2(1-x_3)+x_1(1-x_2)x_3+x_1x_2x_3$, which simplifies to $x_1(x_2+x_3-x_2x_3)$. We use delta notation to denote changes ΔP in the potential function due to changes in the inputs: Given an actual assignment $\boldsymbol{x}^0$ and a subset of variables $\boldsymbol{S}$, we define $\Delta P_\Phi(\boldsymbol{S}) := P_\Phi(\boldsymbol{x}^0) - P_\Phi(\boldsymbol{x}^0 \oplus \boldsymbol{S})$, where $\boldsymbol{x}^0 \oplus \boldsymbol{S}$ (denoting XOR) indicates the assignment obtained by starting from $\boldsymbol{x}^0$ and inverting all variables in $\boldsymbol{S}$.

To semantically capture differences in causality between networks with logically equivalent boolean formulas (e.g. Fig. 2a,2b), we use *Dissociation Expressions* (DEs):

Definition 2 (Dissociation Expression). *A dissociation expression with respect to a variable X_0 is a Boolean expression defined by the grammar:*

$$\Psi ::= X \in \boldsymbol{X}$$

$$\Psi ::= \sigma(\Psi_1, \Psi_2, \ldots, \Psi_k), \quad X_0 \in V(\Psi_i) \cup V(\Psi_j) \Rightarrow V(\Psi_i) \cap V(\Psi_j) \subseteq \{X_0\}$$

where $V(\Psi_i)$ is the set of input variables of formula Ψ_i.

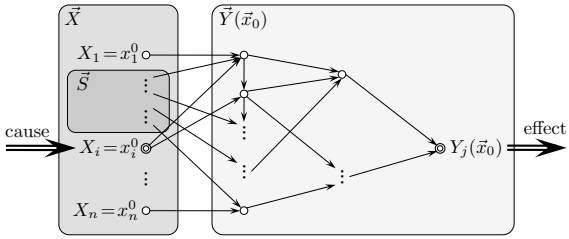

Fig. 3: FC framework: the causal network is partitioned into the input variables $\boldsymbol{X}$ with cause under consideration X_i, and dependent variables $\boldsymbol{Y}$ with effect variable Y_j. Support $\boldsymbol{S} \subseteq \boldsymbol{X} \setminus \{X_i\}$ corresponds to permissive contingency from the HP framework.

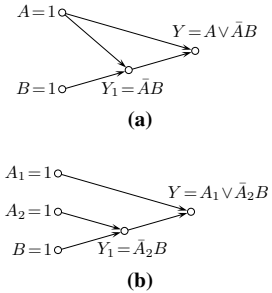

Fig. 4: A causal network CN (a) and its dissociation network DN (b) with respect to B.

Dissociation expressions allow us to semantically *capture within a Boolean formula*, the causal dependencies of a variable X_0 in a causal network. This is possible by recording the effect of X_0 along different network paths and disallowing any variable from being combined with X_0 in more than one subexpression. We illustrate with a detailed example.

Example 2. *In the network of Fig. 4a, variable A contributes to the causal path of B at two locations. This "independent" influence can be represented by the dissociation expression $\Psi = A_1 \vee (\bar{A}_2 \wedge B)$, which essentially separates A into two variables A_1 and A_2 (see Fig. 4b). $\Psi' = A \vee (\bar{A} \wedge B)$ is not a valid DE with respect to B because, for its subexpressions $\Psi'_1 = A$ and $\Psi'_2 = \bar{A} \wedge B$, it is $B \in V(\Psi'_1) \cup V(\Psi'_2)$ but $V(\Psi'_1) \cap V(\Psi'_2) = \{A\} \not\subseteq \{B\}$. We demonstrate how Ψ captures semantically the network structure: The HP definition checks actual causality of B in the network of Fig. 4a by determining the value of Y for a setting $\{A=0, B=1\}$, while forcing Y_1 to its original value. The dissociation expression $\Psi(A_1, A_2, B) = A_1 \vee (\bar{A}_2 \wedge B)$, with potential function $P_\Psi(a_1, a_2, b) = a_1 + b - a_1 b - a_2 b + a_1 a_2 b$, allows us to perform the same check by simply computing $P_\Psi(0, 1, 1)$. In this case $P_\Psi(0, 1, 1) = 0 \neq P_\Psi(1, 1, 1)$, which was the original variable assignment, meaning that the change altered values on the causal path.*

The grammar-based definition of dissociation expressions allows us to identify expressions that are valid DEs with respect to a variable. We will now define mappings, called *foldings*, from DEs to Boolean formulas, which are used to formally define correspondence between formulas. For instance, $\bar{A} \vee B$ is a valid dissociation expression with respect to B but does not correspond to formula $A \vee (\bar{A} \wedge B)$. A folding basically maps a set of input variables X' to another set X, transforming formula Ψ to Ψ'. If Ψ' is grammatically equivalent to Φ, then Ψ is a dissociation expression of Φ. For example, $f(\{A_1, A_2, B\}) = \{A, A, B\}$ defines a folding from $\Psi = A_1 \vee (\bar{A}_2 \wedge B)$ to the formula $\Phi = A \vee (\bar{A} \wedge B)$. In simple terms, a DE Ψ with a folding to Φ is a representation of Φ with a larger number of input variables.

Definition 3 (Expression Folding). *Given $f : \boldsymbol{X'} \rightarrow \boldsymbol{X}$ mapping variables $\boldsymbol{X'}$ to $\boldsymbol{X}$, the* folding *$(\mathcal{F}, f)$ of a dissociation expression $\Psi(\boldsymbol{X'})$ defines a formula $\Phi = F(\Psi)$, s.t:*

$$\Psi ::= X' \Rightarrow \mathcal{F}(X) = f(X')$$
$$\Psi ::= \sigma(\Psi_1, \Psi_2, \ldots, \Psi_k) \Rightarrow \mathcal{F}(\Psi) = \sigma\left(\mathcal{F}(\Psi_1), \mathcal{F}(\Psi_2), \ldots, \mathcal{F}(\Psi_k)\right)$$

The dissociation of input variables into several new input variables captures the distinct effect of variables on the causal path, thus providing the necessary network semantics. Using $|\Psi|$ to denote the cardinality of the input set of Ψ, then $|\Psi| \geq |\Phi|$, and if $|\Psi| = |\Phi|$ then $\Psi = \Phi$.

Theorem 1 (DE Minimality). *If $\mathcal{D}$ the set of all DEs w.r.t. $X_0 \in \boldsymbol{X}$ with a folding to $\Phi(\boldsymbol{X})$, then $\exists$ unique $\Psi_i \in \mathcal{D}$ of minimum size: $|\Psi_i| = \min_{\Psi \in \mathcal{D}} |\Psi|$ and $\forall j \neq i$, $|\Psi_j| = |\Psi_i| \Rightarrow \Psi_j = \Psi_i$.*

The DE of minimum size replicates those variables, and only those variables, that affect the causal path at more than one location. It is simply called the dissociation expression of Φ, with input nodes $\boldsymbol{X}_t$ (Fig. 4b). A folding maps $\boldsymbol{X}_t$ back to the original input variables: $\boldsymbol{X} = f(\boldsymbol{X}_t)$. The reverse mapping is denoted $\boldsymbol{X}_t = [\boldsymbol{X}]_t = \{X_i \,|\, f(X_i) \in \boldsymbol{X}\}$. We often refer to the *dissociation network* of Φ, meaning the causal network representing the DE of Φ (e.g. Fig. 4b).

Definition 4 (Functional Cause). *The event $X_i = x_i^0$ is a cause of ϕ in a causal model iff:*

FC1. *Both $X_i = x_i^0$ and ϕ hold under assignment $\boldsymbol{x}^0$*

FC2. *Let P_Φ and P_Ψ be the potential functions of Φ and its DE w.r.t. X_i, respectively. There exists a support $\boldsymbol{S} \subseteq \boldsymbol{X} \backslash \{X_i\}$, such that:*

(a) $\Delta P_\Phi(\boldsymbol{S} \cup X_i) \neq 0$

(b) $\Delta P_\Psi(\boldsymbol{S}'_t) = 0$, *for all subsets $\boldsymbol{S}'_t \subseteq [\boldsymbol{S}]_t$*

Condition FC2(b) is analogous to AC2(b) of the HP definition, which requires checking that the effect does not change for all possible combinations of setting the dependent nodes to their original values. Similarly, FC ensures that no part of the changed nodes (the support $\boldsymbol{S}$) is counterfactual in the dissociation network.

Intuition. The definition of functional causes captures three main points: (i) a counterfactual cause is always a cause, (ii) if a variable is not counterfactual under any possible assignment of the other variables, then it cannot be a cause, and (iii) if $X = x^0$ is a counterfactual cause under some assignment that inverts a subset $\boldsymbol{S}$ of the other variables, then no part of $\boldsymbol{S}$ should be by itself counterfactual.

We use the rock thrower example from [12], depicted in Fig. 2a and 2b, to demonstrate how functional causes (like actual causes) can handle preemption.

Example 3. *The two different models of the problem, with and without preemption (Fig. 2b and 2a respectively) are characterized by logically equivalent Boolean expressions: $A \vee \bar{A}B = A \vee B$. However, B is not a cause (actual or functional) in Fig. 2b, because Bob's throw is* preempted *by Alice's. The minimal dissociation expression for $\Phi = A \vee (\bar{A} \wedge B)$ with respect to B is $\Psi = A_1 \vee (\bar{A}_2 \wedge B)$, and is depicted in Fig. 4b. Then:*

$$P_\Phi = a + b - ab \;\text{ and }\; P_\Psi = a_1 + b - a_1 b - a_2 b + a_1 a_2 b$$

For $\boldsymbol{S} = \{A\}$, $\Delta P_\Phi(B, \boldsymbol{S}) \neq 0$. If (F, f) the folding of Ψ into Φ, then $[\boldsymbol{S}]_t = \{A_1, A_2\}$, and $\Delta P_\Psi(A_1) \neq 0$, so B is not a functional cause.

Hence, the definition of functional causes effectively captures the difference between the two networks for the two thrower example (Fig. 2a,2b) while *only focusing on the input nodes*, as opposed to the HP definition that requires the inspection of the values of all the dependent nodes under all assignments. In the case of the simple network, $P_\Phi = P_\Psi$ and for $\boldsymbol{S} = \{A\}$, B can be shown to be a cause. However, in the more complicated network, the potential function of the dissociation expression gives priority to A's throw and determines that B is not a cause of the bottle breaking.

If the causal network is a tree, then the causal formula is itself a dissociation expression with potential P_Φ. Then, (FC2) simplifies to: (a) $\Delta P_\Phi(\boldsymbol{S}, X_i) \neq 0$ and (b) $\forall \boldsymbol{S}' \subseteq \boldsymbol{S}$: $\Delta P_\Phi(\boldsymbol{S}') = 0$. Causal networks which are trees form an important category of causality

problems as they model many practical cases of database queries, and they are characterized by desirable properties, as we show in Sect. 3.3.

Responsibility. Responsibility is a measure for degree of causality, first introduced by Chockler and Halpern [6]. We redefine it here for functional causes.

Definition 5 (Responsibility). *Responsibility ρ of a causal variable X_i is defined as $\frac{1}{|\boldsymbol{S}|+1}$ where $\boldsymbol{S}$ the minimum support for which X_i is a functional cause of an effect under consideration. $\rho := 0$ if X_i is not a cause.*

Responsibility ranges between 0 and 1. Non-zero responsibility ($\rho > 0$) means that the variable is a functional cause, $\rho = 1$ means it is also a counterfactual cause.

3 Formal Properties

Functional causality encodes the *semantics of causal structures* with the help of potential functions which are dependent only on the input variables. Functional causes are a refined notion of actual causes. Even though the definition of AC does not exclude dependent variables, functional causality does not consider them as possible causes, as their value is fully determined from the input variables. The relationship of functional causality of input variables to actual and counterfactual causality is demonstrated in the following theorem.

Theorem 2. *Every $X = x^0$ that is a counterfactual cause is also a functional cause, and every $X = x^0$ that is a functional cause is also an actual cause.*

Actual causes are more permissive than functional causes, as indicated by the limitations mentioned in Sect. 2. The issue is analyzed extensively in [20]. In this section we demonstrate that *functional causality* provides a more powerful and robust way to reason about causes than *actual causality*. In addition, we give a transitivity result and use it to derive complexity results for certain types of causal network structures.

3.1 Causal Network Expansion

Functional, as well as actual causes, rely on the causal network to model a given problem. The two different models of the thrower example displayed in Fig. 2(a,b) demonstrate that changes in the network structure can help model priorities of events, which in turn can redefine causality of variables.

In Fig. 2b, B is removed as a cause by the addition of an intermediate node in the causal network structure that models the preemption of the effect by node A (Alice's rock is the one that breaks the bottle). This change is also visible in the causal Boolean formula, which is transformed from $\Phi = A \vee B$ to $\Phi_1 = A \vee (\bar{A} \wedge B)$. As we know from Boolean algebra, the two formulas are equivalent as they have the same truth tables. However, they are not *causally equivalent*, as they yield different causality results. Therefore, the grammatical form of the Boolean expression is important in determining causality, and the functional definition captures that through dissociation expressions. It is important to understand how changes in the causal network affect causality, and whether we can state meaningful properties for those changes.

We define *causal network expansion* in a standard way by the addition of nodes and/or edges to the causal structure. A network CN_e with formula Φ_e is a *node expansion* (respectively *edge expansion*) of CN with formula Φ if it can be created by the addition of a node (respectively edge) to CN, while $\Phi_e \equiv \Phi$. CN_e is a *single-step expansion* if it is either a node or an edge expansion of CN.

Definition 6 (Expansion). *A causal network CN_e is an* expansion *of network CN iff $\exists$ set $\{CN_1, CN_2, \ldots, CN_k\}$ with $CN_1 = CN$ and $CN_k = CN_e$, such that CN_{i+1} is a single step expansion of CN_i, $\forall i \in [1, k]$.*

Networks represented by the formulas $\Phi_1 = A \vee (\bar{A} \wedge B)$ and $\Phi_2 = (A \wedge \bar{B}) \vee B$ are both expansions of $\Phi = A \vee B$, but note that Φ_1 and Φ_2 are not expansions of one another. As shown by the thrower example, network expansion can remove causes. As the following theorem states, it can only remove, not add causes.

Theorem 3. *If CN_e with formula Φ_e is an expansion of CN with formula Φ and $X_i = x_i^0$ is a cause in ϕ_e then $X_i = x_i^0$ is also a cause in ϕ.*

Specifically in the case where no negation of literals is allowed, changes to the structure do not affect the causality result:

Theorem 4. *If CN_e with formula Φ_e is an expansion of CN with formula Φ that does not contain negated variables then ϕ and ϕ_e have the same causes.*

The properties of formula expansion are important, as they prevent unpredictability due to causal structure changes. Note that the Halpern and Pearl definition does not handle formula expansion as gracefully. Figure 2 demonstrates with an example that the HP definition allows introducing new causes with expansion. $A = 1$ is not a cause in the simple network of Fig. 2c but becomes causal after adding node Y_2 in Fig. 2d. Therefore, network expansion is *unpredictable for actual causes*, as there are examples where it can both remove (Fig. 2b) or introduce new causes (Fig. 2d). This is a strong point for our definition, as causality is tied to the network structure, and erratic behavior due to minor structure changes, as is the case in this example, is troubling.

3.2 Functional causes and transitivity

Functional causality only considers *input nodes* in the causal network as permissible causes for events. Under this premise, the notion of *transitivity of causality* is not well-defined, since dependent variables are never considered permissible causes of events in their descendants. In order to ask the question of transitivity, we allow a dependent variable Y_1 to become a possible cause in a *modified causal model* M' with Y_1 as additional input variable. We achieve this with the help of an external intervention $[Y_1 \leftarrow y_1^0]$, setting the variable to its actual value y_1^0. The new model is then $M' = (\boldsymbol{N}, \mathcal{F}')$ with modified structural equations $\mathcal{F}' = \mathcal{F} \setminus \{F_{Y_1}\} \cup \{F'_{Y_1}\}$, where $F'_{Y_1} = y_1^0$, and hence new input variables $\boldsymbol{X}' = (\boldsymbol{X}, Y_1)$ with original assignment $\boldsymbol{x}'^0 = (\boldsymbol{x}^0, y_1^0)$.

We can now ask the question of transitivity as follows: Assume that an assignment $X = x^0$ is a cause of $Y_1 = y_1^0$ in a causal model M. Further assume that $Y_1 = y_1^0$ is a cause of $Y_2 = y_2^0$ in the modified network $[Y_1 \leftarrow y_1^0]$. Is then $X = x^0$ a cause of $Y_2 = y_2^0$ in the original network M? In agreement with recent prevalent (yet not undisputed) opinion in causality literature [14,22], functional causality is *not* transitive, in general.

Intransitivity of causality is not uncontroversial [17] and humans generally feel a strong intuition that causality *should be* transitive. It turns out that functional causality is actually transitive in an important type of network structure that relates to this intuition: Transitivity holds if there is no causal connection between the original cause (X) and the effect (Y_2) except through the intermediate node (Y_1). This property allows us to deduce a lower complexity for determining causality in restricted settings in Sect. 3.3.

Definition 7 (Markovian). *A node N is Markovian in a causal network CN iff there is no path from any ancestor of N to any descendent of N that does not pass through N.*

Proposition 5 (Markovian transitivity). *Given a causal model M in which $X=x^0$ is a cause of $Y_1=y_1^0$ with responsibility ρ_1, and Y_1 is Markovian. Further assume that $Y_1=y_1^0$ is a cause of $Y_2=y_2^0$ with responsibility ρ_2 in the modified causal model $[Y_1 \leftarrow y_1^0]$. Then $X=x^0$ is a cause of $Y_2=y_2^0$ in M with responsibility $\rho = (\rho_1^{-1} + \rho_2^{-1} - 1)^{-1}$*

3.3 Complexity

Analogous to Eiter and Lukasiewicz's result that determining actual causes for Boolean variables is NP-hard [9], determining functional causality is also NP-hard, in general.

Theorem 6 (Hardness). *Given a Boolean formula Φ on causal network CN and assignment $\boldsymbol{x}^0$ of the input variables, determining whether $X_i = x_i^0$ is a cause of $\phi = \Phi(\boldsymbol{x}^0)$ is NP-hard.*

Even though determining functional causality is hard, there are important cases that can be solved in polynomial time.

If the causal network is a tree, then the dissociation network is the same as the causal network and there is a single potential function. Determining causality on a tree can be simplified, as a result of the Markovian transitivity property (Proposition 5) and the fact that all nodes in a tree are Markovian.

Lemma 7 (Causality in Trees). *If $X_i = x_i^0$ is a cause of the output node Y in a tree causal network, and $\boldsymbol{p} = \{X, Y_1, Y_2, \ldots, Y\}$ the unique path from X to Y, then every node in $\boldsymbol{p}$ is a functional cause of all of its descendants in $\boldsymbol{p}$. Consequently, X is a cause of all $Y_i \in \boldsymbol{p}$.*

Following from Lemma 7, causality in cases of tree-shaped causal structures with bounded arity (number of parents per node) is decidable in polynomial time.

Theorem 8 (Trees with arity $\leq k$). *Given a tree-shaped causal network with formula Φ and bounded arity and actual assignment $\boldsymbol{x}^0$ of the input variables, determining whether $X_i = x_i^0$ is a cause of $\phi = \Phi(\boldsymbol{x}^0)$ is in P.*

An even better result is given by Theorem 9, that covers the case of causal structures where the function at every node is a primitive boolean operator (`AND`, `OR`, `NOT`), without any restrictions on the arity.

Theorem 9 (Trees with Primitive Operators). *Given a tree causal network with formula Φ where the function of every node is a primitive boolean operator, i.e.* `AND`, `OR`, `NOT`, *and assignment $\boldsymbol{x}^0$ of the input variables, determining whether $X_i = x_i^0$ is a cause of $\phi = \Phi(\boldsymbol{x}^0)$ is in P.*

As demonstrated by Olteanu and Huang in [25], the lineage expressions of safe queries do not have repeated tuples. Lineage expressions for conjunctive queries with no repeated tuples correspond to causal networks that are trees. Following directly from Theorem 9, we get complexity results for safe queries.

Corollary 10 (Causes of Safe Queries). *Determining the functional causes of safe queries can be done in polynomial time.*

N(ews feeds)

nid	*story*	*source*
1	... doing utmost to prevent more floods ...	AsiaOne
2	... economic downturn affected ...	NYTimes
3	... with sequences shot in Singapore ...	AsiaOne
4	... BP's chief executive apologizes ...	NYTimes
5	... apology for oil disaster ...	AsiaOne
6	... VLDB held in Singapore ...	NYTimes
7	... discussed in a recent talk the ...	NYTimes
8	... European stimulus measures ...	NYTimes
9	... Singapore welcomes VLDB ...	AsiaOne

F(iltered feed)

story
... doing utmost to prevent more floods ...
... economic downturn affected ...
... sequences shot in Singapore ...
... BP's chief executive apologizes ...
... VLDB held in Singapore ...
... discussed in a recent talk the ...
... European stimulus measures ...

Fig. 5: News feed with aggregated data from different sources (left), and filtered feed (right).

In these tractable cases, due to the transitivity property, responsibility can also be computed in polynomial time, using the formula of Proposition 5.

Another important category of tractable networks are those that correspond to DNF and CNF formulas with no negated literals. This category covers important cases of join queries in a database context.

Theorem 11 (Positive DNF/CNF). *Given a positive DNF (or CNF) formula Φ and assignment $\boldsymbol{x}^0$ of the input variables, determining whether $X_i = x_i^0$ is a cause of $\phi = \Phi(\boldsymbol{x}^0)$ is in PTIME.*

4 Explaining Query Results

In this section, we show how causality can be applied to address examples from the database literature, like provenance and "Why Not?" queries, as well as examples showcasing causality of aggregates. We also demonstrate how our causality framework can model different types of elements that can be considered contributory to a query result, like *query operations* instead of tuples.

4.1 WHY SO? and WHY NO?

We revisit our motivating example (Example 1), but introduce a slight variation that aggregates data from different news sources to demonstrate how functional causality can be used to answer WHY SO? and WHY NO? questions.

Example 4 (News aggregator). *A user has access to the News feed relation N, depicted in Fig. 5. N contains news articles from two different sources, the NY Times and the Singapore Press holdings portal, Asia One. The user, who resides in Singapore, likes to read more local news from Asia One, but she prefers the NY Times with regards to global interest news. Hence, she does not want to read on topics from Asia One that are also covered by the NY Times. Her filtered feed is constructed by the query:*

```
select  N.story
from    N
where   N.source='NYTimes' or
        not exists ( select  *
                     from    N as N1
                     where   topic(N1.story)=topic(N.story)
                             and N1.source='NYTimes')
```

where `topic()` *is a topic extractor modeled as a user-defined function. The user's filtered feed will contain stories from NY Times, and only those stories from Asia One that NY Times does not cover. Simply, if S_{NY} is an article in NY Times covering a topic, and S_A an article in Asia One about the same topic, whether the user will see this topic in her feed or not follows a causal model similar to that of Fig. 4a, with boolean formula $\Phi = S_{NY} \vee (\bar{S}_{NY} \wedge S_A)$. The topic appears in F if it appears in either NY Times or Asia One, but the first gets priority.*

When asking what is the cause of getting an article on the Orchard Rd floods, the user gets tuple 1 from relation N, as it is counterfactual. When asking what is the cause of seeing an article on VLDB, she gets the NY Times article (tuple 6), even though Asia One also had a story about it (tuple 9). The analysis is equivalent to the rock thrower example.

The framework can be used in a similar fashion to respond to "Why No?" questions. Assume tuple t_{10} =`(10,'... immigration officials arrest 300...',NYTimes)`, which was present in yesterday's news feed, but was since then removed. Tuple t_{10} is a functional cause to the WHY NO? question: "Why do I not see news on immigration", as it is counterfactual. Its removal from the feed caused the absence of immigration topics in the user's filtered view.

4.2 Aggregates

We next show how functional causality can be applied to determine causes and responsibility for aggregates. We focus here only on positive integers and give complexity results for WHY SO? and WHY NO? for WHY IS $\texttt{SUM} \geq c$? and WHY IS $\texttt{SUM} \not\geq c$?. In the following we denote with $\Omega \in \{\texttt{SUM}, \texttt{MAX}, \texttt{AVG}, \texttt{MIN}, \texttt{COUNT}\}$ an aggregate function evaluated over a multiset of values ($\Omega(\boldsymbol{V})$), $\boldsymbol{X}$ is a vector of boolean values representing presence of absence of tuples, and op is an operator from the set $\{\geq, >, \leq, <, =, \neq\}$.

Definition 8 (Why so? and Why no?). *Let $\omega^0 = \Omega(\boldsymbol{x}^0)$ be the value of an aggregate function for current assignment $\boldsymbol{x}^0$. The question of* WHY SO? *(respectively,* WHY NO?*) for a condition ω^0* op *c that is* `true` *(respectively,* `false`*) under the current assignment corresponds to the question of which set of tuples $\{t_i\}$ from the tuple universe with original assignment $x_i^0 = 1$ (respectively, 0) is a cause of the event $\phi = \big(\omega^0 \texttt{ op } c = \texttt{true}\big)$ (respectively,* `false`*) with responsibility ρ_i.*

Example 5 (Sum example). *Consider a tuple universe $\boldsymbol{T} = [(10), (20), (30), (50), (100)]$ and a view $R(A)$ with the subset of tuples $\boldsymbol{R} = \{(20), (30), (100)\}$. Now consider the query* `select SUM(R.A) from R` *executed over the view R which returns 150. In our notation, this is represented with a vector $\boldsymbol{V} = [10, 20, 30, 50, 100]$, current assignment $\boldsymbol{x}^0 = [0, 1, 1, 0, 1]$, and $\texttt{SUM}(\boldsymbol{x}^0) = 150$ (see Fig. 6a).*

WHY $\texttt{SUM} \geq c$?: t_3 is a cause of $\texttt{SUM}(\boldsymbol{x}^0) \geq 30$ with responsibility $\frac{1}{2}$. FC2(a): $\texttt{SUM}(\boldsymbol{x}^1) \not\geq 30$ for $\boldsymbol{x}^1 = [0, 1, \underline{0}, 0, \underline{0}]$. FC2(b): $\texttt{SUM}(\boldsymbol{x}^{1*}) \geq 30$ for every assignment $\boldsymbol{x}^{1*}$ with $x_3^{1*} = 1$ and any subset of $\{x_5^1 = 0\}$ inverted to its original assignment. In contrast, t_2 is not a cause: While FC2(a) holds for $\boldsymbol{x}^1 = [0, \underline{0}, \underline{0}, \underline{1}, \underline{0}]$ with $\texttt{SUM}(\boldsymbol{x}^1) \not\geq 30$ (and then t_2 would be counterfactual), FC2(b) is not fulfilled for $\boldsymbol{x}^{1*} = [0, 1, \underline{0}, 0, \underline{0}]$.

WHY $\texttt{SUM} \not\geq c$?: t_4 is a cause of $\big(\texttt{SUM}(\boldsymbol{x}^0) \geq 180\big) =$ `false`*, as both x_4 and the condition are* `false` *under current assignment, but would hold for $\boldsymbol{x}^1 = [0, 1, 1, \underline{1}, 1]$.*

R

	A	
t_2	20	$x_2=1$
t_3	30	$x_3=1$
t_5	100	$x_5=1$
SUM	150	

$T-R$

	A	
t_1	10	$x_1=0$
t_4	50	$x_4=0$

(a)

		WHY SUM($\boldsymbol{x}^0$) $\geq$					WHY SUM($\boldsymbol{x}^0$) $\ngeq$			
t_i	x_i^0	20	30	40	60	130	160	180	210	220
t_1	0	–	–	–	–	–	1	–	$\frac{1}{2}$	–
t_2	1	$\frac{1}{3}$	–	$\frac{1}{2}$	–	–	–	–	–	–
t_3	1	$\frac{1}{3}$	$\frac{1}{2}$	$\frac{1}{2}$	–	1	–	–	–	–
t_4	0	–	–	–	–	–	1	1	$\frac{1}{2}$	–
t_5	1	$\frac{1}{3}$	$\frac{1}{2}$	$\frac{1}{2}$	1	1	–	–	–	–

(b)

Fig. 6: Sum example. (a): Relation R with tuples from tuple domain $\boldsymbol{T}$. (b): Responsibility ρ_i of t_i for WHY SO? (SUM($\boldsymbol{x}^0$) $\geq c$) and WHY NO? (SUM($\boldsymbol{x}^0$) $\ngeq c$).

Figure 6b shows responsibility for different values of constant c in Example 5 and illustrates that responsibility for SUM is not monotone. In order to compute responsibility for a tuple t_i, one must find the smallest set of tuples that, when inverted (i.e. either inserted or deleted) make tuple t_i counterfactual for the condition. Determining the causes of an aggregate is in general NP-complete. We refer the reader to our technical report [20] for further theoretical analysis of aggregate causality and more examples.

4.3 Causes beyond tuples

Provenance and non-answers commonly focus on tuples as discrete units that have contribution to a query result. Our causality framework is not restricted to tuples, but can model any element that could be considered contributory to a result. To showcase this flexibility, we pick an example from Chapman and Jagadish [4] that models *operations* in workflows as possible answers to "Why not?" questions.

Example 6 (Book Shopper [4], Ex. 1). *A shopper knows that all "window display books" at Ye Olde Booke Shoppe are around $20, and wishes to make a cheap purchase. She issues the query: Show me all window books. Suppose the result from this query is (Euripides, "Medea"). Why is (Hrotsvit, "Basilius") not in the result set? Is it not a book in the book store? Does it cost more than $20? Is there a bug in the query-database interface such that the query was not correctly translated?*

Chapman and Jagadish consider a discrete component of a workflow, called *manipulation*, as an explanation of a "Why not?" query. The workflow describing the query of the example is shown in Fig. 7b. Roughly, a manipulation is considered *picky* for a non-result if it prunes the tuple. For example, manipulation 1 of Fig. 7b is picky for "Odyssey", as it costs more than $20. Equivalently, a manipulation is *frontier picky* for a set of non-results, if it is the last in the workflow to reject tuples from the set. In this framework, the cause of a non-answer will be a frontier picky manipulation.

In Example 6, tuple t =(Hrotsvit, "Basilius") passes the price test, but is cut by manipulation 2 as it doesn't satisfy the seasonal criteria. The causal network representing this example is presented in Fig. 7c. Input nodes model the events: M_1: manipulation 1 is not potentially picky with respect to t, and M_2: manipulation 2 is not potentially picky with respect to t. At the end, the tuple appears only if neither manipulation is picky: $M_1 \wedge M_2$. Intermediate node Y_1 encodes the precedence of the manipulations in the workflow. A tuple will be stopped at point Y_1 of the workflow if M_2 is picky but M_1 was not: $M_1 \wedge \bar{M}_2$. It will pass this point if the opposite holds, so $Y_1 = \overline{M_1 \wedge \bar{M}_2} = \bar{M}_1 \vee M_2$, and $Y = M_1 \wedge Y_1$.

Author	Title	Price	Publisher
	Epic of Gilgamesh	$150	Hesperus
Euripides	Medea	$16	Free Press
Homer	Iliad	$18	Penguin
Homer	Odyssey	$49	Vintage
Hrotsvit	Basilius	$20	Harper
Longfellow	Wreck of the Hesperus	$89	Penguin
Shakespeare	Coriolanus	$70	Penguin
Sophocles	Antigone	$48	Free Press
Virgil	Aeneid	$92	Vintage

(a)

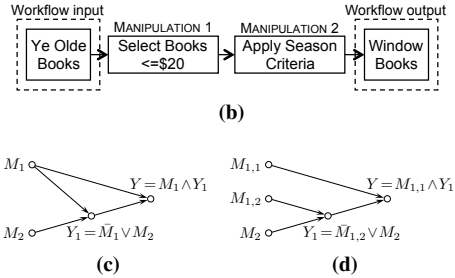

Fig. 7: (a) Books in "Ye Olde Booke Shoppe" [4]. (b) Variation of the query workflow from [4]. The causal network of Example 6 (c), and its DN with respect to M_2 (d).

Applying the FC framework for $M_1 = 1$ (M_1 is not picky), and $M_2 = 0$ (M_2 is picky), correctly yields that M_2 is the only cause: $\boldsymbol{S} = \emptyset$, $\Delta I_\Phi(M_2) \neq 0$. If both manipulations were potentially picky ($M_1 = 0$ and $M_2 = 0$), the FC definition again correctly picks M_1 as the only cause with support $\boldsymbol{S} = \{M_2\}$ (even though M_2 is potentially picky, the tuple never gets to it), which agrees with the WHY NOT? framework that selects as explanation the last manipulation that rejected the tuple.

5 Related Work

Our work is mainly related and unifies ideas from three main areas: research on *causality*, *provenance*, and *missing query result explanations*.

Causality. Causality is an active research area mainly in logic and philosophy with their own dedicated workshops (see e.g. [1]). The most prevalent definitions of causality are based on the idea of *counterfactual causes*, i.e. causes are explained in terms of counterfactual conditionals of the form *If X had not occurred, Y would not have occurred*. This idea of counterfactual causality can be traced back to Hume [22]. The best known counterfactual analysis of causation in modern times is due to Lewis [16]. In a databases setting, Miklau and Suciu [23] define *critical tuples* as those which can become counterfactual under some value assignment of variables. Halpern and Pearl [12] (HP in short) define a variation they call *actual causality*. Roughly speaking, the idea is that X is a cause of Y if Y counterfactually depends on X under "some" *permissive contingency*, where "some" is elaborately defined. Later, Chockler and Halpern [6] define the degree of *responsibility* as a gradual way to assign causality. Eiter and Lukasiewicz [9] show that the problem of detecting whether $X = x^0$ is an actual cause of an event is Σ_2^P-complete for general acyclic models and NP-complete for binary acyclic models. They also give an alleged proof showing that actual causality is always reducible to primitive events. However, Halpern [11] later gives an example for non-primitive actual causes, showing this proof to ignore some cases under the original definition. Chockler et al. [7] later apply causality and responsibility to binary Boolean networks, giving a modified definition of cause.

A general overview of various applications of causality in a database context is given in [19]. The complexity of computing causality and responsibility is studied in [21] for the case of conjunctive queries, leading to a strong dichotomy result.

Provenance. Approaches for defining data provenance can be mainly divided into three categories: *how*, *why*, and *where* provenance ([3,5,8,10]). In particular for the "why so" case, we observe a close connection between provenance and causality, where it is often the case that tuples in the provenance for the result of a positive query result are causes.

While none of the work on provenance mentions or makes direct connections to causality, those connections can be found. The work by Buneman et al. [3] makes a distinction between why and where provenance that can be connected to causality as follows: *why provenance* returns all tuples that can be considered causes for a particular result, and *where provenance* returns attributes along a particular causal path. Green et al. [10] present a generalization for all types of provenance as semirings; finding functional causes in a Boolean tree, if taken in a provenance context, yields degree-one polynomials for provenance semirings. View data lineage, as presented by Cui et al. [8] also addresses aggregates but lacks a notion of graded contribution.

In contrast, our approach can rank tuples according to their *responsibility*, hence our approach allows to determine a gradual contribution with counterfactual tuples ranked first. Also, in contrast to our paper, most of the work on provenance has little or no connection to the philosophical groundwork on causality. We take this work and significantly adapt it so that it can be applied to databases.

Missing query results. Very recent work has focused on the question "why no", i.e. why is a certain tuple *not* in the result set? The work by Huang et al. [15] presents provenance for potential answers and never answers. In the case that no insertions or modifications can yield the desired result - usually for privacy or security reasons - the system declares that particular tuple a never answer. Both Huang's work and Artemis [13] handle potential answers by providing tuple insertions or modifications that would yield the missing tuples. Alternatively, Chapman and Jagadish [4] focus on which *manipulation* in the query plan eliminated a specific tuple, while Tran and Chan [26] show how the query can be modified in order to include missing results in the answer. Lim et al. [18] adopt a third, explanation-based, approach. This approach aims to answer questions such as *why, why not, how to,* and *what if* for context-aware applications, but does not address a database setting.

Our work, unifies the above approaches in the sense that we model both, *tuples* or *manipulations*, as possible causes for missing query answers. Also, our approach unifies the problem of explaining missing query answers (*why* is a tuple not in the query result) with work on provenance (*why* is a tuple in the query result).

Other. Minsky and Papert initiated the study of the computational properties of Boolean functions using their representation by polynomials and call this the *arithmetic* instead of the logical form [24, p.27]. This method was later successfully used in complexity theory and became known as *arithmetization* [2].

6 Conclusions and Future Work

In this paper, we defined *functional causes*, a rigorous and extensible definition of causality encoding the semantics of *causal structures* with the help of powerful *potential functions*. Through theoretical analysis of its properties, we demonstrated that our definition provides a more powerful and robust way to reason about causes than other established notions of causality. Albeit NP-hard in the general case, common categories of causal networks that correspond to interesting database examples (e.g. safe queries) prove to be tractable. We presented several database examples that portrayed the applicability of our framework in the context of provenance, explanation of non-answers, as well as aggregates. We demonstrated how to determine causes of query results for `SUM` and `COUNT` aggregates, and how these can be ranked according to the causality metric of *responsibility*.

Providing support for causal queries allows users to better understand the reasons behind their observations, and is an important tool for identifying potential errors in uncertain or untrusted data. Overall, with this work we establish the theoretical foundations of causality theory in the database context, which we view as a unified framework that deals with query result explanations.

Acknowledgements. This work was partially supported by NSF grants IIS-0911036, IIS-0915054, and IIS-0713576. We would like to thank Christoph Koch for valuable insights, and Chris Ré for helpful discussions in early stages of this project.

References

1. International multidisciplinary workshop on causality. IRIT, Toulouse, June 2009.
2. L. Babai and L. Fortnow. Arithmetization: A new method in structural complexity theory. *Computational Complexity*, 1:41–66, 1991.
3. P. Buneman, S. Khanna, and W. C. Tan. Why and where: A characterization of data provenance. In *ICDT*, 2001.
4. A. Chapman and H. V. Jagadish. Why not? In *SIGMOD*, 2009.
5. J. Cheney, L. Chiticariu, and W. C. Tan. Provenance in databases: Why, how, and where. *Foundations and Trends in Databases*, 1(4):379–474, 2009.
6. H. Chockler and J. Y. Halpern. Responsibility and blame: A structural-model approach. *J. Artif. Intell. Res. (JAIR)*, 22:93–115, 2004.
7. H. Chockler, J. Y. Halpern, and O. Kupferman. What causes a system to satisfy a specification? *ACM Trans. Comput. Log.*, 9(3), 2008.
8. Y. Cui, J. Widom, and J. L. Wiener. Tracing the lineage of view data in a warehousing environment. *ACM Trans. Database Syst.*, 25(2):179–227, 2000.
9. T. Eiter and T. Lukasiewicz. Complexity results for structure-based causality. *Artif. Intell.*, 142(1):53–89, 2002. (Conference version in *IJCAI*, 2002).
10. T. J. Green, G. Karvounarakis, and V. Tannen. Provenance semirings. In *PODS*, 2007.
11. J. Y. Halpern. Defaults and normality in causal structures. In *KR*, 2008.
12. J. Y. Halpern and J. Pearl. Causes and explanations: A structural-model approach. Part I: Causes. *Brit. J. Phil. Sci.*, 56:843–887, 2005. (Conference version in *UAI*, 2001).
13. M. Herschel, M. A. Hernández, and W. C. Tan. Artemis: A system for analyzing missing answers. *PVLDB*, 2(2):1550–1553, 2009.
14. C. Hitchcock. The intransitivity of causation revealed in equations and graphs. *The Journal of Philosophy*, 98(6):273–299, 2001.
15. J. Huang, T. Chen, A. Doan, and J. F. Naughton. On the provenance of non-answers to queries over extracted data. *PVLDB*, 1(1):736–747, 2008.
16. D. Lewis. Causation. *The Journal of Philosophy*, 70(17):556–567, 1973.
17. D. Lewis. Causation as influence. *The Journal of Philosophy*, 97(4):182–197, 2000.
18. B. Y. Lim, A. K. Dey, and D. Avrahami. *Why* and *why not* explanations improve the intelligibility of context-aware intelligent systems. In *CHI*, 2009.
19. A. Meliou, W. Gatterbauer, J. Halpern, C. Koch, K. F. Moore, and D. Suciu. Causality in databases. *IEEE Data Engineering Bulletin special issue on Provenance*, Sept. 2010.
(to appear, see `http://db.cs.washington.edu/causality/`).
20. A. Meliou, W. Gatterbauer, K. F. Moore, and D. Suciu. Why so? or why no? functional causality for explaining query answers. *CoRR*, abs/0912.5340, 2009. (see `http://db.cs.washington.edu/causality/`).
21. A. Meliou, W. Gatterbauer, K. F. Moore, and D. Suciu. The causality and responsibility of query answers and non-answers. In *PVLDB*, 2011. (to appear, see `http://db.cs.washington.edu/causality/`).
22. P. Menzies. Counterfactual theories of causation. Stanford Encylopedia of Philosophy, 2008.
23. G. Miklau and D. Suciu. A formal analysis of information disclosure in data exchange. In *SIGMOD*, 2004.
24. M. L. Minsky and S. Papert. *Perceptrons - expanded edition: An introduction to computational geometry*. MIT Press, 1987.
25. D. Olteanu and J. Huang. Secondary-storage confidence computation for conjunctive queries with inequalities. In *SIGMOD*, 2009.
26. Q. T. Tran and C.-Y. Chan. How to conquer why-not questions. In *SIGMOD*, 2010.