From Shapley Value to Model Counting and Back

AHMET KARA, University of Zurich, Switzerland
DAN OLTEANU, University of Zurich, Switzerland
DAN SUCIU, University of Washington, USA

In this paper we investigate the problem of quantifying the contribution of each variable to the satisfying assignments of a Boolean function based on the Shapley value.

Our main result is a polynomial-time equivalence between computing Shapley values and model counting for any class of Boolean functions that are closed under substitutions of variables with disjunctions of fresh variables. This result settles an open problem raised in prior work, which sought to connect the Shapley value computation to probabilistic query evaluation.

We show two applications of our result. First, the Shapley values can be computed in polynomial time over deterministic and decomposable circuits, since they are closed under OR-substitutions. Second, there is a polynomial-time equivalence between computing the Shapley value for the tuples contributing to the answer of a Boolean conjunctive query and counting the models in the lineage of the query. This equivalence allows us to immediately recover the dichotomy for Shapley value computation in case of self-join-free Boolean conjunctive queries; in particular, the hardness for non-hierarchical queries can now be shown using a simple reduction from the #P-hard problem of model counting for lineage in positive bipartite disjunctive normal form.

CCS Concepts: • **Theory of computation → Data provenance**; **Logic and databases**; *Problems, reductions and completeness.*

Additional Key Words and Phrases: explanations, query lineage, polynomial time reductions

ACM Reference Format:
Ahmet Kara, Dan Olteanu, and Dan Suciu. 2024. From Shapley Value to Model Counting and Back. *Proc. ACM Manag. Data* 2, 2 (PODS), Article 79 (May 2024), 23 pages. https://doi.org/10.1145/3651142

1 INTRODUCTION

The Shapley value quantifies the fair contribution of a player to a wealth function that is shared by a set of players in a cooperative game [29, 31]. For this reason, it has been used in a variety of applications ranging from bioinformatics to network analysis and machine learning: measuring the centrality and power of genes [24] and the influence in social networks [25]; sharing profit between Internet providers [21, 22]; finding key players in networks [33]; feature selection, explainability, multi-agent reinforcement learning, ensemble pruning, and data valuation [23, 30].

In this paper we investigate the problem of computing the Shapley value for variables in Boolean functions. The Shapley values quantify the contribution of each variable to the satisfying assignments of the Boolean function. Understanding the importance of variables to the outcome of a Boolean function has numerous applications [16, 17]. The nature of the Shapley values for the variables in Boolean functions can also serve as complexity-theoretic assumption for tractability

Authors' addresses: Ahmet Kara, ahmet.kara@uzh.ch, University of Zurich, Department of Informatics, Andreasstrasse 15, Zurich, 8050, Switzerland; Dan Olteanu, dan.olteanu@uzh.ch, University of Zurich, Department of Informatics, Andreasstrasse 15, Zurich, 8050, Switzerland; Dan Suciu, suciu@cs.washington.edu, University of Washington, Department of Computer Science & Engineering, Box 352350, Seattle, WA, 98195-2350, USA.

ACM 2836-6573/2024/5-ART79
https://doi.org/10.1145/3651142

in generalized constraint satisfaction problems with order predicates [4, 18]. When focusing on functions representing the lineage of Boolean conjunctive queries in relational databases [15], the Shapley values are used to support explanations for query answers. In this setting, the tuples in the input database are the players that contribute to the answer of a given query and the Shapley value assigns a score to each input tuple based on its contribution to the query answer. Recent works in database theory and systems [10, 12, 20, 28] have made great progress towards charting the tractability frontier of computing the Shapley values of database tuples and proposed algorithms for exact and approximate computation. We next highlight two key results from prior work.

First, for every Boolean query Q and database $\boldsymbol{D}$, the problem of computing the Shapley value of any tuple in $\boldsymbol{D}$ reduces in polynomial time to the problem of computing Q over a probabilistic version of $\boldsymbol{D}$, where each tuple becomes an independent random variable [12]. This connection to probabilistic query evaluation (PQE) allows to transfer well-established results from PQE to Shapley value computation. In particular, the tractability of PQE for safe queries [32] implies the tractability of Shapley value computation for safe queries. Furthermore, knowledge compilation techniques developed for PQE can be adjusted for Shapley value computation. It is stated as open problem whether PQE also reduces in polynomial time to Shapley value computation, effectively establishing a polynomial-time equivalence between the two problems [12].

Second, the dichotomy for conjunctive queries without self-joins over probabilistic databases [6] also holds for Shapley value computation [20]: For any self-join-free Boolean conjunctive query Q, the problem of Shapley value computation is in FP if Q is hierarchical and is $\mathrm{FP}^{\#\mathrm{P}}$-hard otherwise.

The main result in this paper is a polynomial-time equivalence between the Shapley value computation and model counting for any class of Boolean functions that are closed under substitutions of variables with (possibly empty) disjunctions of fresh variables. This equivalence connects the Shapley value computation to a fundamental and well-established problem [14] with many applications from artificial intelligence to formal verification. This result settles the open problem raised in prior work [12], albeit not using PQE but model counting under OR-substitutions.

We also show two applications of our result. In Section 4 we first show that deterministic and decomposable circuits are closed under OR-substitutions, where we allow further polynomial-time transformations. Since model counting is tractable for such circuits [8], it follows from our main result that Shapley value computation is also tractable for such circuits. Deterministic and decomposable circuits are extensively investigated in knowledge compilation [8, 9], prime examples are the ordered binary decision diagrams (OBDDs) and the deterministic decomposable negation normal forms (d-DNNFs).

Our second application is in databases. In Section 5 we show a polynomial-time equivalence between computing the Shapley value for the tuples contributing to the answer of a Boolean conjunctive query Q and counting the models in the *lineage* of Q. When lifted to the level of the query, the OR-substitutions can be expressed by *stretching* the query, a rewriting which introduces fresh variables in relations. This equivalence allows us to immediately recover the dichotomy for Shapley value computation in case of self-join-free Boolean conjunctive queries [20]; in particular, the hardness for non-hierarchical queries can now be shown using a simple reduction from the #P-hard problem of model counting for lineage in positive bipartite formulas in disjunctive normal form [27], as previously used to show $\mathrm{FP}^{\#\mathrm{P}}$-hardness of PQE [6].

Shapley value versus SHAP score. Recent works [2, 3, 11] consider the notion of SHAP score, which is based on, yet different from, the Shapley value and used for providing explanations in machine learning. For a given classification model M, entity $\boldsymbol{e}$, and feature x, the SHAP score intuitively represents the importance of the feature value $\boldsymbol{e}(x)$ to the classification result $M(\boldsymbol{e})$. In its general formulation, it takes as input a Boolean function F encoding a Boolean classifier and a

probability distribution on the set of truth assignments. The probability distribution is assumed to be a *product distribution*, also called a *fully factorized distribution*, and the wealth function of the SHAP score is an *expectation*. In this setting, it was shown that computing the SHAP score is polynomial-time equivalent to *weighted* model counting for the function F [11]. These prior works [1, 3] also show that the SHAP score can be computed in polynomial time in case the Boolean function F is given by a tractable (deterministic and decomposable) circuit. Tractability of such circuits is the main study in knowledge compilation [8, 9].

In contrast, we study the Shapley value where the wealth function is just the Boolean function F, without any probability distribution. This appears unrelated to the SHAP score, in particular it is not equivalent to setting all probabilities to 1/2. While there exist fully-polynomial randomized approximation schemes (FPRAS) for model counting [19] and the Shapley value in the database context [20], there is no such FPRAS for the SHAP score even in case of positive bipartite DNF functions [2]. Our polynomial-time equivalence is technically more challenging than for the SHAP score discussed in prior work [11], because we no longer have the ability to use an oracle with varying probability functions (or, equivalently, weight functions). Instead, our proof of equivalence relies on the ability to substitute a Boolean variable with a disjunction of fresh variables.

2 PRELIMINARIES

We use $\mathbb{N}$ to denote the set of natural numbers including 0. For $n \in \mathbb{N}$, we denote by $[n] \stackrel{\text{def}}{=} \{1, 2, \ldots, n\}$. In case $n = 0$, then $[n] = \emptyset$.

Boolean Functions. Let $\boldsymbol{X}$ be the set of $n \in \mathbb{N}$ Boolean variables $X_1, \ldots, X_n$. Where convenient, we may denote a variable X_i by its index i. A *Boolean function* over $n \in \mathbb{N}$ variables is a function $F : \{0,1\}^n \to \{0,1\}$. We denote by $\boldsymbol{BF}$ the set of all Boolean functions.

Representations of Boolean Functions. In this paper, we consider two syntactic representations for Boolean functions: propositional formulas and Boolean circuits, which we introduce next.

A *propositional formula* φ over a set $\boldsymbol{X}$ of variables is a constant 0 or 1, a variable $X_i \in \boldsymbol{X}$, the negation $\neg\varphi$ of a formula φ, the logical conjunction $\varphi_1 \wedge \varphi_2$, or the logical disjunction $\varphi_1 \vee \varphi_2$ of formulas φ_1 and φ_2. The size $|\varphi|$ of the formula φ is the number of the occurrences of its constants, variables, and logical operators $\neg$, $\wedge$, and $\vee$.

A *Boolean circuit* G over a set $\boldsymbol{X}$ of variables is a directed acyclic graph where each node is one of the following *gates*: a constant gate labeled with either 0 or 1; a variable gate labeled with a variable from $\boldsymbol{X}$; or a logic gate labeled with a Boolean operator $\wedge$ (and), $\vee$ (or), or $\neg$ (not). The constant and variable gates have no incoming edges. The logic gates $\wedge$ and $\vee$ may have two or more incoming edges, and the logic gate $\neg$ has one incoming edge. There is one gate, called the output gate, that has no outgoing edge. The size $|G|$ of a circuit G is the number of its edges. The main difference between formulas and circuits is that in circuits, we can name formulas and re-use them at the place of variables.

Two formulas (circuits) are isomorphic if they are equal up to renaming of variables; we consider such formulas (circuits) identical. For instance, the formulas $X_1 \wedge (X_2 \vee \neg X_3)$ and $Y_1 \wedge (Y_2 \vee \neg Y_3)$ are isomorphic.

The same Boolean function may admit different formula and circuit representations. In this section and Section 5, we use the representation of functions as propositional formulas. For example, we write $F = X_1 \wedge (X_2 \vee \neg X_3)$ to describe a function F over the three variables X_1, X_2, and X_3. In Section 4, we use the circuit representation of functions. The results in Section 3 hold for any representation of Boolean functions.

Substitutions. Given $n \in \mathbb{N}$, a *substitution* is a function $\theta : [n] \to \boldsymbol{BF}$. We often denote the substitution θ by the set $\{X_1 := \theta(1), \ldots, X_n := \theta(n)\}$. The result of applying the substitution θ to a Boolean function F is denoted by $F[\theta]$. We may define a substitution only on a subset of the variables and assume implicitly that the other variables are mapped to themselves. For example, for the above function F and substitution $\theta = \{X_2 := Z_1 \vee Z_2\}$, we have $F[\theta] = X_1 \wedge (Z_1 \vee Z_2 \vee \neg X_3)$.

Definition 2.1. A Boolean function F over n variables *admits an OR-substitution into* a Boolean function G, denoted by $F \overset{\text{OR}}{\to} G$, if $G = F[\theta]$ with $\theta = \{X_i := Z_i^1 \vee \ldots \vee Z_i^{m_i} \mid i \in [n]\}$ for $m_1, \ldots, m_n \in \mathbb{N}$ and fresh variables $Z_i^1, \ldots, Z_i^{m_i}$. Notice that G has $\sum_{i=1}^n m_i$ variables. If $m_i = 0$, then θ maps X_i to 0.

Given a class C of Boolean functions, we define $\widetilde{C} = \{G \mid \exists F \in C \text{ such that } F \overset{\text{OR}}{\to} G\}$. We say that C *OR-substitutes into* $\widetilde{C}$.

Notice that $C \subseteq \widetilde{C}$, because we can substitute each X_i with a single variable Z_i and obtain an isomorphic function.

Valuations. Valuations are special substitutions where variables are mapped to constants. Given a valuation $\theta : [n] \to \{0, 1\}$, we denote by $F[\theta]$ the Boolean value of F. We say that θ is a *model* of F if $F[\theta] = 1$. It is often convenient to denote the valuation θ by the set $T \overset{\text{def}}{=} \{i \in [n] \mid \theta(i) = 1\}$, in which case we write $F[T]$ for $F[\theta]$. The size of a model θ is thus the number of variables it sets to 1, i.e., $|T|$. For instance, consider the valuation $T = \{1\}$, which for the example function $F = X_1 \wedge (X_2 \vee \neg X_3)$ maps X_1 to 1 and the other two variables X_2 and X_3 to 0. Then, $F[\{X_1\}] = 1$, so T is a model of F of size 1. Two functions F_1 and F_2 are *equivalent*, denoted by $F_1 \equiv F_2$, if $F_1[\theta] = F_2[\theta]$ for all valuations θ.

Model Counting. Consider a Boolean function F over n variables. The *model count* $\#F$ is the number of models of F:

$$\#F \overset{\text{def}}{=} \sum_{T \subseteq [n]} F[T]$$

Given $0 \leq k \leq n$, the *k-model count* $\#_k F$ is the number of models of F of size k:

$$\#_k F \overset{\text{def}}{=} \sum_{T \subseteq \binom{[n]}{k}} F[T]$$

where $\binom{[n]}{k}$ represents the subsets of $[n]$ of size k. We denote the vector of k-model counts by:

$$\#_{0,\ldots,n} F \overset{\text{def}}{=} (\#_0 F, \#_1 F, \ldots, \#_n F)$$

Shapley value. Given a Boolean function F over n variables, the *Shapley value* of a variable X_i for $i \in [n]$ is defined as:

$$\mathsf{Shap}(F, X_i) \overset{\text{def}}{=} \frac{1}{n!} \sum_{\Pi \in S_n} \left(F[\Pi^{<i} \cup \{i\}] - F[\Pi^{<i}] \right) \tag{1}$$

where S_n is the symmetric group, i.e., the set of permutations of $[n]$, and $\Pi^{<i}$ is the set of indices j that come before i in the permutation Π. If i is at the first position of Π, then $\Pi^{<i}$ is the empty set.

Example 2.2. Consider again the function $F = X_1 \wedge (X_2 \vee \neg X_3)$. The only models of the function are $\{X_1\}$, $\{X_1, X_2\}$, and $\{X_1, X_2, X_3\}$. Hence, $\#F = 3$, $\#_0 F = 0$, and $\#_1 F = \#_2 F = \#_3 F = 1$. The table below shows for each possible permutation $\Pi \in S_3$, the difference $F[\Pi^{<i} \cup \{i\}] - F[\Pi^{<i}]$ for

$i \in [3]$. For instance, in case $\Pi = (2,1,3)$ we have $\Pi^{<1} \cup \{1\} = \{1,2\}$ and $\Pi^{<1} = \{2\}$. Hence, $F[\Pi^{<1} \cup \{1\}] - F[\Pi^{<1}] = 1 - 0 = 1$.

	$F[\Pi^{<i} \cup \{i\}] - F[\Pi^{<i}]$		
Π	$i=1$	$i=2$	$i=3$
$(1,2,3)$	1	0	0
$(1,3,2)$	1	1	-1
$(2,1,3)$	1	0	0
$(2,3,1)$	1	0	0
$(3,1,2)$	0	1	0
$(3,2,1)$	1	0	0

To obtain the Shapley value of variable X_i, we sum up the values in the column for i and divide by $3! = 6$. We obtain $\mathsf{Shap}(F, X_1) = \frac{5}{6}$, $\mathsf{Shap}(F, X_2) = \frac{2}{6}$, $\mathsf{Shap}(F, X_3) = -\frac{1}{6}$. Note that the Shapley value of X_3 is negative because it appears negatively in the function.

Next, we give an alternative formulation of the Shapley value that uses model counting.

Proposition 2.3 ([20] page 11, adapted). *The* Shapley value *of a variable X_i of a Boolean function F is:*

$$\mathsf{Shap}(F, X_i) = \sum_{k=0}^{n-1} c_k \left(\#_k F[X_i := 1] - \#_k F[X_i := 0]\right) \tag{2}$$

where $c_k = \frac{k!(n-k-1)!}{n!}$.

The above formulation does not consider $\#_n F$, since X_i is set to either 1 or 0 and F has therefore $n-1$ remaining variables.

Example 2.4. We compute the Shapley value of X_1 in $F = X_1 \wedge (X_2 \vee \neg X_3)$ using Eq. (2). We have $F[X_1 := 0] = 0 \wedge (X_2 \vee \neg X_3)$. Since this function cannot evaluate to 1, we have $\#_0 F[X_1 := 0] = \#_1 F[X_1 := 0] = \#_2 F[X_1 := 0] = 0$. It holds $F[X_1 := 1] = 1 \wedge (X_2 \vee \neg X_3)$. The function $F[X_1 := 1]$ has the models $\emptyset$, $\{X_2\}$, and $\{X_2, X_3\}$. Hence, $\#_0 F[X_1 := 1] = \#_1 F[X_1 := 1] = \#_2 F[X_1 := 1] = 1$. We have $c_0 = \frac{0!(3-0-1)!}{6} = \frac{2}{6}$, $c_1 = \frac{1!(3-1-1)!}{6} = \frac{1}{6}$, and $c_2 = \frac{2!(3-2-1)!}{6} = \frac{2}{6}$. Following Eq. (2), we obtain $\mathsf{Shap}(F, X_1) = \frac{2}{6} + \frac{1}{6} + \frac{2}{6} = \frac{5}{6}$, which is the Shapley value of X_1 as computed in Example 2.2.

The following proposition follows immediately from the definition of the Shapley value:

Proposition 2.5. *For any Boolean function F, it holds*

$$\sum_{i \in [n]} \mathsf{Shap}(F, X_i) = F[\mathbf{1}] - F[\mathbf{0}]$$

where $\mathbf{1}$ *is the valuation that maps all variables to* 1*, and* $\mathbf{0}$ *the valuation that maps all variables to* 0.

In the original setting, $\sum_{i \in [n]} \mathsf{Shap}(F, X_i) = F[\mathbf{1}]$. This does not hold in our case, since $F[\mathbf{0}]$ may not necessarily be 0 as F may have both positive and negative literals. That is, in our setting the *efficiency* property ($F[\mathbf{0}] = 0$) of the Shapley value [29] does not hold; it holds for functions where all literals are positive.

Example 2.6. For the function $F = X_1 \wedge (X_2 \vee \neg X_3)$, we have $F[\mathbf{1}] = 1$ and $F[\mathbf{0}] = 0$, since $\mathbf{1}$ is a model of F but $\mathbf{0}$ is not. By Proposition 2.5, the Shapley values of the variables of F must sum up to 1. This is indeed the case, since we have $\mathsf{Shap}(X_1) = \frac{5}{6}$, $\mathsf{Shap}(X_2) = \frac{2}{6}$, and $\mathsf{Shap}(X_3) = -\frac{1}{6}$ (see Example 2.2).

We write $\mathsf{Shap}(F)$ to denote the vector of the Shapley values of all variables in F:

$$\mathsf{Shap}(F) \stackrel{\text{def}}{=} (\mathsf{Shap}(F, X_1), \ldots, \mathsf{Shap}(F, X_n))$$

Polynomial-time Reductions and Transformations. A *polynomial-time reduction* (also called a *Cook reduction*) from a problem A to a problem B, denoted by $A \leq^P B$, is a polynomial-time algorithm for the problem A with access to an oracle for the problem B. If $B \leq^P A$ also holds, then we write $A \equiv^P B$ and say that the two problems are polynomial-time equivalent. A *polynomial-time transformation* from a class of functions C_1 to another class of functions C_2, denoted by $C_1 \lesssim^P C_2$, is an algorithm T that takes time polynomial in the representation size of functions and such that: $\forall F_1 \in C_1, \exists F_2 \in C_2 : F_2 = T(F_1)$ and $F_1 \equiv F_2$. If $C_2 \lesssim^P C_1$ also holds, then we write $C_1 \approx^P C_2$ and say that C_1 and C_2 have a bidirectional polynomial-time transformation.

3 POLYNOMIAL-TIME REDUCTIONS FOR PROBLEMS OVER BOOLEAN FUNCTIONS

We consider three problems: model counting, fixed-size model counting, and Shapley value computation. They are all parameterized by a class C of Boolean functions. We show reductions between these problems that take time polynomial in the representation size of the functions *under the assumption* that the OR-substitutions can be computed in time polynomial in the sizes of the function and of the substitution. In subsequent sections we show two well-known examples where this assumption is met: for deterministic and decomposable circuits (Section 4) and for query lineage (Section 5).

Given a function $F \in C$ over n variables, the model counting problem asks for the number of models of F:

Problem:	$\#C$
Description:	*Model Counting*
Input:	$F \in C$
Compute:	$\#F$

There is extensive literature on the model counting problem $\#C$ [14]. We use two examples later in this paper. If C is the class of positive, bipartite functions in disjunctive normal form, i.e., functions of the form $F = \bigvee_{(i,j)\in E}(X_i \wedge Y_j)$ where E is a set of pairs $E \subseteq [n] \times [n]$, then $\#C$ is #P-hard [27]. If C is the class of deterministic and decomposable Boolean circuits, then $\#C$ is in FP [8, 9].

The fixed-size model counting problem asks for the number of models of F of size k, for any $0 \leq k \leq n$:

Problem:	$\#_* C$
Description:	*Fixed-Size Model Counting*
Input:	$F \in C$ over n variables
Compute:	$\#_{0,\ldots,n} F$

The Shapley value computation problem asks for the Shapley value of each variable in F:

Problem:	$\mathsf{Shap}(C)$
Description:	*Shapley Value Computation*
Input:	$F \in C$
Compute:	$\mathsf{Shap}(F)$

Our main result gives polynomial-time reductions between the above three problems:

Theorem 3.1. *Given a class C of Boolean functions, it holds:*

- $\mathsf{Shap}(C) \leq^P \#_*\widetilde{C}$
- $\#_*C \leq^P \#\widetilde{C}$
- $\#C \leq^P \mathsf{Shap}(\widetilde{C})$.

In case C OR-substitutes to itself, i.e., $C = \widetilde{C}$, the problems $\#C$, $\#_*C$, and $\mathsf{Shap}(C)$ become polynomial-time equivalent:

Corollary 3.1 (Theorem 3.1). *Given a class C of Boolean functions with $C = \widetilde{C}$, it holds:*

$$\mathsf{Shap}(C) \equiv^P \#_*C \equiv^P \#C$$

This result connects model counting to Shapley value computation. Whenever model counting is tractable for a class C of Boolean functions that is closed under OR-substitutions, then the Shapley value computation is also tractable. We give here an immediate example; Sections 4 and 5 provide two further examples.

The class C of *positive β-acyclic CNF* functions is trivially closed under OR-substitutions[1]. Furthermore, $\#C$ is in FP [5][2]. Corollary 3.1 then implies that $\mathsf{Shap}(C)$ is also in FP.

There are two immediate generalizations of Theorem 3.1. First, we may allow for polynomial-time transformations to accommodate the OR-substitutions. That is, the polynomial-time equivalence between the two problems holds whenever $C \approx^P \widetilde{C}$ holds and not only when $C = \widetilde{C}$ holds. Second, we may use substitutions beyond the OR-substitution considered here, such as AND-substitutions (more details are given at the end of Section 3).

3.1 Proof of Theorem 3.1

We separate the theorem into three lemmas:

Lemma 3.2. $\mathsf{Shap}(C) \leq^P \#_*\widetilde{C}$

Lemma 3.3. $\#_*C \leq^P \#\widetilde{C}$

Lemma 3.4. $\#C \leq^P \mathsf{Shap}(\widetilde{C})$.

Proof of Lemma 3.2. Let $F \in C$ be a Boolean function. Our goal is to compute $\mathsf{Shap}(F)$ in polynomial time, given an oracle for $\#_*\widetilde{C}$. We use Eq. (2) for the Shapley value and the following equality:

$$\#_{k+1}F = \#_kF[X_i := 1] + \#_{k+1}F[X_i := 0]$$

Then, Eq. (2) becomes:

$$\mathsf{Shap}(F, X_i) = \sum_{k=0}^{n-1} c_k \left(\#_{k+1}F - \#_{k+1}F[X_i := 0] - \#_kF[X_i := 0]\right)$$

Consider the function $\widetilde{F}$ that results from F by replacing each variable by a fresh variable and the function $\widetilde{F'}$ that results from F by replacing X_i by the empty disjunction and each other variable by a fresh variable. Clearly, F admits OR-substitutions into $\widetilde{F}$ and $\widetilde{F'}$, hence, $\widetilde{F}, \widetilde{F'} \in \widetilde{C}$. The functions $\widetilde{F}$ and $\widetilde{F'}$ are isomorphic (i.e., identical up to renaming of the variables) to F and respectively $F[X_i := 0]$, so model counting and fixed-size model counting is the same for F and $\widetilde{F}$, and also for

[1]The hypergraph of a CNF function has one node per variable and one hyperedge per clause. It is β-acyclic if there is no cycle in the hypergraph, nor in any sub-hypergraph. Substituting a variable by a disjunction of fresh variables preserves the structure of the CNF and of its hypergraph, except for replacing one node by several nodes that all occur in the same hyperedges as the replaced node.

[2]Tractability holds even when removing the restriction on the functions being positive.

$F[X_i := 0]$ and $\widetilde{F'}$. We thus have access to an oracle to compute the quantities $\#_{k+1}F$, $\#_{k+1}F[X_i := 0]$, and $\#_k F[X_i := 0]$. This means that we can compute $\mathsf{Shap}(F, X_i)$ in polynomial time. □

Proof of Lemma 3.3. Let $F \in C$ be a Boolean function over the variables $\boldsymbol{X} = \{X_1, \ldots, X_n\}$. Our goal is to compute $\#_{0,\ldots,n}(F)$ in polynomial time, given an oracle for $\#\widetilde{C}$. For a valuation $\theta : \boldsymbol{X} \to \{0,1\}$, we write $|\theta|$ for the number of variables X_i s.t. $\theta(X_i) = 1$. It follows:

$$\#_k F = \sum_{\theta : |\theta| = k} F[\theta]$$

for $0 \le k \le n$. For each $\ell \in \mathbb{N}$, define:

$$F^{(\ell)} \stackrel{\text{def}}{=} F[X_1 := \bigvee_{j=1}^{\ell} Z_1^j, \ldots, X_n := \bigvee_{j=1}^{\ell} Z_n^j]$$

where each Z_i^j with $i \in [n]$ and $j \in [\ell]$ is a fresh variable. It holds $F^{(\ell)} \in \widetilde{C}$. Therefore, we have access to an oracle for computing $\#F^{(\ell)}$. We claim:

Claim 3.5. *For each $\ell \in \mathbb{N}$, it holds:*

$$\#F^{(\ell)} = \sum_{k=0}^{n} (2^\ell - 1)^k \#_k F \tag{3}$$

Claim 3.5 implies Lemma 3.3 as follows. We use Eq. (3) for $\ell \in [n+1]$ to form a system of $n+1$ linear equations with the $n+1$ unknowns $\#_0 F, \ldots, \#_n F$. The matrix of this system is a Vandermonde $(n+1)$-by-$(n+1)$ matrix, which is non-singular so we can compute its inverse [13]. Hence, we can solve the linear system

$$\underbrace{\begin{pmatrix} \#F^{(1)} \\ \vdots \\ \#F^{(n+1)} \end{pmatrix}}_{\text{known}} = \underbrace{\begin{pmatrix} 1 & (2^1-1)^1 & \cdots & (2^1-1)^n \\ & \vdots & \vdots & \vdots \\ 1 & (2^{n+1}-1)^1 & \cdots & (2^{n+1}-1)^n \end{pmatrix}}_{\text{Vandermonde}} \underbrace{\begin{pmatrix} \#_0 F \\ \vdots \\ \#_n F \end{pmatrix}}_{\text{unknown}}$$

and determine the values of $\#_0 F, \ldots, \#_n F$ in polynomial time.

It remains to prove Claim 3.5. Let $\boldsymbol{Z} = \{Z_i^j \mid i \in [n] \text{ and } j \in [\ell]\}$ be the set of variables of $F^{(\ell)}$. By definition:

$$\#F^{(\ell)} = \sum_{\varphi : \boldsymbol{Z} \to \{0,1\}} F^{(\ell)}[\varphi] \tag{4}$$

For each valuation $\varphi : \boldsymbol{Z} \to \{0,1\}$ we define the *induced valuation* $\theta_\varphi : \boldsymbol{X} \to \{0,1\}$ by setting $\theta_\varphi(X_i) = \varphi(\bigvee_{j=1}^{\ell} Z_i^j)$. In other words, $\theta_\varphi(X_i) = 1$ iff φ evaluates $Z_i^1 \vee \cdots \vee Z_i^\ell$ to 1. Notice that:

$$\forall \varphi : \boldsymbol{Z} \to \{0,1\}, \quad F^{(\ell)}[\varphi] = F[\theta_\varphi] \tag{5}$$

$$\forall \theta : \boldsymbol{X} \to \{0,1\}, \quad |\{\varphi \mid \theta_\varphi = \theta\}| = (2^\ell - 1)^{|\theta|} \tag{6}$$

We group each valuation φ in Eq. (4) by its induced valuation θ_φ:

$$\begin{aligned}
\#F^{(\ell)} &= \sum_{\theta:\boldsymbol{X}\to\{0,1\}} \sum_{\varphi:\theta_\varphi=\theta} F^{(\ell)}[\varphi] \\
&= \sum_{\theta:\boldsymbol{X}\to\{0,1\}} \sum_{\varphi:\theta_\varphi=\theta} F[\theta] && \text{(by Eq. (5))} \\
&= \sum_{\theta:\boldsymbol{X}\to\{0,1\}} (2^\ell-1)^{|\theta|} F[\theta] && \text{(by Eq. (6))} \\
&= \sum_{k=0}^{n} \sum_{\theta:\boldsymbol{X}\to\{0,1\}:|\theta|=k} (2^\ell-1)^k F[\theta] \\
&= \sum_{k=0}^{n} (2^\ell-1)^k \#_k F
\end{aligned}$$

This completes the proof of Claim 3.5, which implies Lemma 3.3. □

Proof of Lemma 3.4. Let $F \in C$ be a Boolean function. Our goal is to compute $\#F$ in polynomial time given an oracle to $\mathsf{Shap}(\widetilde{C})$.

Suppose F has n variables $\boldsymbol{X} = \{X_1, \ldots, X_n\}$. We fix $\ell \in \mathbb{N}$. For each variable X_i, let $F^{(\ell,i)}$ be the function obtained from F by substituting X_i with a fresh variable Z_i and every other variable X_p with a disjunction of fresh variables $X_p := Z_p^1 \vee \cdots \vee Z_p^\ell$. The function F admits OR-substitutions into $F^{(\ell,i)}$, hence, $F^{(\ell,i)} \in \widetilde{C}$. Using the oracle for $\mathsf{Shap}(\widetilde{C})$, we compute $\mathsf{Shap}(F^{(\ell,i)}, Z_i)$. Then, using Eq. (2) for the Shapley value and Eq. (3), we obtain:

$$\begin{aligned}
\mathsf{Shap}(F^{(\ell,i)}, Z_i) &= \sum_{k=0}^{n-1} c_k \left(\#_k F^{(\ell,i)}[Z_i := 1] - \#_k F^{(\ell,i)}[Z_i := 0] \right) \\
&= \sum_{k=0}^{n-1} (2^\ell-1)^k c_k \left(\#_k F[X_i := 1] - \#_k F[X_i := 0] \right)
\end{aligned}$$

Keeping i fixed, we let ℓ iterate over $[n]$ to form a system of n equations with n unknowns $\Gamma_k F \stackrel{\text{def}}{=} c_k \left(\#_k F[X_i := 1] - \#_k F[X_i := 0] \right)$, $k \in \{0, \ldots, n-1\}$.

$$\underbrace{\begin{pmatrix} \mathsf{Shap}(F^{(1,i)}, Z_i) \\ \vdots \\ \mathsf{Shap}(F^{(n,i)}, Z_i) \end{pmatrix}}_{\text{known}} = \underbrace{\begin{pmatrix} (2^1-1)^0 & \cdots & (2^1-1)^{n-1} \\ \vdots & \vdots & \vdots \\ (2^n-1)^0 & \cdots & (2^n-1)^{n-1} \end{pmatrix}}_{\text{Vandermonde}} \underbrace{\begin{pmatrix} \Gamma_0 F \\ \vdots \\ \Gamma_{n-1} F \end{pmatrix}}_{\text{unknown}}$$

The matrix of the equation system is a Vandermonde matrix, hence, nonsingular. We solve the system, and, since the constants c_k are known and computable in polynomial time, we obtain all differences $\#_k F[X_i := 1] - \#_k F[X_i := 0]$. We next show how to compute $\#F$ using these differences. Let us keep k fixed and sum these differences for $i \in [n]$. We claim:

Claim 3.6. *For any $k \in \{0, \ldots, n-1\}$, it holds:*

$$\sum_{i=1}^{n} \left(\#_k F[X_i := 1] - \#_k F[X_i := 0] \right) = (k+1)\#_{k+1} F - (n-k)\#_k F$$

Claim 3.6 follows from the following two equalities:

$$\sum_{i=1}^{n} \#_k F[X_i := 1] = (k+1)\#_{k+1} F \tag{7}$$

$$\sum_{i=1}^{n} \#_k F[X_i := 0] = (n-k)\#_k F \tag{8}$$

Equality (7) holds as follows:

$$\begin{aligned}
\sum_{i=1}^{n} \#_k F[X_i := 1] &= \sum_{i=1}^{n} \sum_{\theta: \boldsymbol{X} - \{X_i\} \to \{0,1\}; |\theta| = k} F[\{X_i := 1\} \cup \theta] \\
&= \sum_{i=1}^{n} \sum_{\varphi: \boldsymbol{X} \to \{0,1\}; |\varphi| = k+1; \varphi(X_i) = 1} F[\varphi] \\
&\overset{(*)}{=} (k+1) \sum_{\psi: \boldsymbol{X} \to \{0,1\}; |\psi| = k+1} F[\psi] \\
&= (k+1)\#_{k+1} F
\end{aligned}$$

Equality $(*)$ holds because each valuation φ, which maps X_i and k other variables to 1 and the remaining $n-k-1$ variables to 0, is considered $k+1$ times when iterating over all $i \in [n]$. More precisely, let T be the set of the indices of the $k+1$ variables set to 1 in φ. Then out of the n iterations in the outer sum, the valuation φ is only considered for $i \in T$.

Equality (8) above follows from a similar argument.

$$\begin{aligned}
\sum_{i=1}^{n} \#_k F[X_i := 0] &= \sum_{i=1}^{n} \sum_{\theta: \boldsymbol{X} - \{X_i\} \to \{0,1\}; |\theta| = k} F[\{X_i := 0\} \cup \theta] \\
&= \sum_{i=1}^{n} \sum_{\varphi: \boldsymbol{X} \to \{0,1\}; |\varphi| = k; \varphi(X_i) = 0} F[\varphi] \\
&\overset{(**)}{=} (n-k) \sum_{\psi: \boldsymbol{X} \to \{0,1\}; |\psi| = k} F[\psi] \\
&= (n-k)\#_k F
\end{aligned}$$

Equality $(**)$ holds because each valuation φ, which maps X_i to 0, k other variables to 1, and the remaining $n-k-1$ variables to 0, is considered $n-k$ times when iterating over all $i \in [n]$. More precisely, let T be the set of the indices of the k variables set to 1 in φ. Then out of the n iterations in the outer sum, the valuation φ is only considered for $i \in [n] \setminus T$, as for $i \in T$ the considered valuations have variable X_i set to 0.

This completes the proof of Claim 3.6. Thus, we have computed all n differences $(k+1)\#_{k+1} F - (n-k)\#_k F$. The final step is the following. Start by observing that $\#_0 F = F[\mathbf{0}]$, where $\mathbf{0}$ is the valuation that sets all variables to 0. Then, proceed inductively, computing $\#_k F$ for $k = \{1, \ldots, n\}$, using Claim 3.6, where we have already computed the left-hand side. □

AND-substitutions. Theorem 3.1 also holds for AND-substitutions:

$$F^{(\ell)} \overset{\text{def}}{=} F[X_1 := \bigwedge_{j=1}^{\ell} Z_1^j, \ldots, X_n := \bigwedge_{j=1}^{\ell} Z_n^j]$$

where each Z_i^j with $i \in [n]$ and $j \in [\ell]$ is a fresh variable. To accommodate AND-substitutions, Claim 3.5 changes as follows:

Claim 3.7. *For each $\ell \in \mathbb{N}$, it holds:*

$$\#F^{(\ell)} = \sum_{k=0}^{n} (2^\ell - 1)^{n-k} \#_k F$$

4 FROM FUNCTIONS TO CIRCUITS

In general, Boolean functions do not admit polynomial-time satisfiability and model counting. Knowledge compilation is an approach that turns Boolean functions into equivalent representations that admit polynomial-time computation for a large number of tasks including model counting [8, 9]. The price to pay is a possibly exponential time in the number of variables of the function to compute such an equivalent yet tractable representation. The tractability of well-known circuits, such as OBDDs and d-DNNFs, relies on two key properties: determinism and decomposability.

We next recall the notion of a deterministic and decomposable circuit and then show that such circuits can efficiently accommodate OR-substitutions. This implies that the Shapley value can be computed in time polynomial in the size of such tractable circuits.

4.1 Deterministic and Decomposable Circuits

Given a circuit G, a gate g in G defines the circuit G_g that is G where all gates that have no directed path to g are removed. An $\vee$-gate g is *deterministic* if for every pair (g_1, g_2) of distinct input gates of g, their circuits G_{g_1} and G_{g_2} are disjoint: There is no valuation θ such that $G_{g_1}[\theta] = G_{g_2}[\theta] = 1$. An $\wedge$-gate g is *decomposable* if for every pair (g_1, g_2) of distinct input gates of g, their circuits G_{g_1} and G_{g_2} have no variable in common. A circuit is deterministic if all its $\vee$-gates are deterministic and is decomposable if all its $\wedge$-gates are decomposable.

Example 4.1. Consider the circuit $(\neg X_1 \wedge X_2) \vee (X_1 \wedge X_3)$. It is deterministic as its only $\vee$-gate is deterministic: There is no valuation that maps both $\neg X_1 \wedge X_2$ and $X_1 \wedge X_3$ to 1, since the two functions are mutually exclusive. It is also decomposable since for both $\wedge$-gates have input gates whose circuits do not share variables.

4.2 Circuits under OR-substitutions

Our main insight in this section is that the deterministic and decomposable circuits can efficiently accommodate OR-substitutions. Let $\mathcal{G}$ be the class of deterministic and decomposable circuits and $\widetilde{\mathcal{G}}$ be the class of circuits in $\mathcal{G}$ where some variables are OR-substituted.

Lemma 4.2. $\widetilde{\mathcal{G}} \lesssim^P \mathcal{G}$.

More precisely, we can show the following for any deterministic and decomposable circuit G, a variable X that occurs k times in G, and distinct variables $Z_1, \ldots, Z_n$ that do not occur in G: A deterministic and decomposable circuit that represents G under the OR-substitution $X \overset{\text{OR}}{\to} \bigvee_{i=1}^{\ell} Z_i$ can be computed in $O(|G| + k\ell)$ time. This proves that the assumption made at the beginning of Section 3 holds for such circuits.

Proof. While the circuit $G_\vee(Z_1, \ldots, Z_\ell) = Z_1 \vee \cdots \vee Z_\ell$ that replaces X is not deterministic, it can be turned into an equivalent deterministic and decomposable circuit of size $O(\ell)$:

$$G_\vee(Z_i, \ldots, Z_\ell) = Z_i \vee (\neg Z_i \wedge (G_\vee(Z_{i+1}, \ldots, Z_\ell))), \text{ for } i \in [\ell - 1]$$
$$G_\vee(Z_\ell) = Z_\ell$$

Its negation $\neg G_\vee(Z_1, \ldots, Z_\ell)$ can be equivalently expressed as $\neg Z_1 \wedge \cdots \wedge \neg Z_\ell$, which is both deterministic and decomposable, since Z_1 to Z_ℓ are distinct variables. Furthermore, substituting X by $G_\vee$ and $\neg X$ by $\neg G_\vee$ does not violate the decomposability and determinism of the gates that are reached from X and $\neg X$. □

The next theorem states that the Shapley value can be computed in polynomial time on deterministic and decomposable circuits. It is an immediate corollary of three results: (1) the well-known result on tractability of model counting for $\mathcal{G}$ [8]; (2) Lemma 4.2 stating that OR-substitutions can be assimilated by any circuit in $\mathcal{G}$ in polynomial time; and (3) Theorem 3.1 conditioning the tractability of Shap on the tractability of model counting for functions under OR-substitutions.

Theorem 4.1. $\mathsf{Shap}(\mathcal{G})$ *is in FP.*

5 FROM FUNCTIONS TO QUERIES

We now lift our investigation of the Shapley value computation problem from Boolean functions to (first-order) conjunctive queries. This is an application of our main result in Theorem 3.1, enabled by the observation that the *lineage or provenance polynomial* [15] of a query is a Boolean function.

One challenge is to understand what is the counterpart of OR-substitutions at the query level. For this purpose, we introduce the notion of *stretching* of a query and show that the lineage of the stretching of a CQ Q is equivalent to the lineage of Q under OR-substitutions. Furthermore, the two lineages can be transformed into one another in polynomial time. One caveat specific to this section is that the problems and reductions used in the results below use data complexity[3].

The main result of this section is the recovery of the dichotomy for Shapley value computation [20] using immediate derivations based on our main theorem and classical results for model counting.

5.1 Conjunctive Queries and Lineage

We consider databases where some relations are *endogenous* while all others are *exogenous*. While we are interested in the contribution of the tuples from endogenous relations to the answer of a query, we disregard the contribution of the tuples from exogenous relations. Whenever we need to distinguish between the two kinds of relations, we annotate an endogenous relation R as R^n and an exogenous relation R as R^x.

A Boolean Conjunctive Query (CQ) is:

$$Q = \exists \boldsymbol{x} \bigwedge_{j \in [m]} R_j(\boldsymbol{y}_j) \tag{9}$$

where $\boldsymbol{x}$ is the tuple of all variables in Q, $R_j(\boldsymbol{y}_j)$ are the atoms of Q where R_j is either an endogenous or an exogenous relation, and $\boldsymbol{y}_j \subseteq \boldsymbol{x}$ for $j \in [m]$. The size of Q, denoted by $|Q|$, is the number m of its atoms. We denote by $at(x)$ the atoms with variable x, i.e., $at(x) = \{R_j(\boldsymbol{y}_j) | j \in [m], x \in \boldsymbol{y}_j\}$. To distinguish between variables in queries from those in Boolean functions, we write the former in lowercase and the latter in uppercase.

A CQ Q is *hierarchical* if for any two query variables x and y, one of the the following conditions hold: $at(x) \cap at(y) = \emptyset$, $at(x) \subseteq at(y)$, or $at(y) \subseteq at(x)$. A CQ Q is *self-join-free* if there are no two atoms for the same relation.

For each database instance $\boldsymbol{D}$, the *lineage* $F_{Q,\boldsymbol{D}}$ of a CQ Q over $\boldsymbol{D}$ is a Boolean function represented as a positive propositional formula in disjunctive normal form (DNF) over the variables $v(t)$ associated to the tuples t in $\boldsymbol{D}$. Each clause in the lineage is a conjunction of m variables, where m

[3]Under data complexity, the query is fixed and has constant size. The complexity $O(|D|^{|Q|})$ is thus polynomial time, since the exponent $|Q|$ is the constant query size.

is the number of relation atoms in Q. We define lineage recursively on the structure of a CQ ($\boldsymbol{D}$ is implicit and dropped from the subscript):

$$F_{Q_1 \vee Q_2} = F_{Q_1} \vee F_{Q_2} \qquad F_{Q_1 \wedge Q_2} = F_{Q_1} \wedge F_{Q_2}$$

$$F_{\exists x Q} = \bigvee_{a \in \text{adom}(D)} F_{Q[a/x]}$$

$$F_{R^n(t)} = \begin{cases} v(t) & \text{if } t \in R \\ 0 & \text{otherwise} \end{cases} \qquad F_{R^x(t)} = \begin{cases} 1 & \text{if } t \in R \\ 0 & \text{otherwise} \end{cases}$$

The lineage of a conjunction (disjunction) of two subqueries is the conjunction (disjunction) of their lineages. In case of an existential quantifier $\exists x$, we construct the disjunction of the lineages of all residual queries obtained by replacing the query variable x by each value in the active domain (adom) of the database $\boldsymbol{D}$. Once all variables in an atom $R(t)$ are replaced by constants, we check whether the tuple t of these constants is in the relation R. If it is not, then it does not contribute to the lineage (it is 0, or false). If it is, then we distinguish two cases. If R is endogenous, then the Boolean variable $v(t)$ associated with the tuple t is added to the lineage. If R is exogenous, then we add instead 1 (or true) to signal that the variable $v(t)$ is not relevant for Shapley value computation.

The query Q defines a class of Boolean functions that consists of the lineages of Q over all databases $\boldsymbol{D}$:

$$C_Q \stackrel{\text{def}}{=} \{F_{Q,D} \mid \boldsymbol{D} \text{ is a database instance}\}$$

5.2 Stretching Databases and Queries

The following transformation is central to this section:

Definition 5.1. Given an endogenous relation $R^n(y_1, \ldots, y_k)$ with attributes $y_1, \ldots, y_k$, its *stretching* is the relation $\widetilde{R}^n(y_0, y_1, \ldots, y_k)$. That is, we add one new attribute on the first position.

Given a CQ, where $\forall j \in [m] : \boldsymbol{a}_j \subseteq \boldsymbol{a}$ and $\forall j \in [p] : \boldsymbol{b}_j \subseteq \boldsymbol{b}$:

$$Q = \exists \boldsymbol{a}\, \exists \boldsymbol{b} \bigwedge_{j \in [m]} R_j^n(\boldsymbol{a}_j) \wedge \bigwedge_{j \in [p]} S_j^x(\boldsymbol{b}_j)$$

its *stretching* is the CQ

$$\widetilde{Q} = \exists \boldsymbol{a}\, \exists z_1 \ldots \exists z_m\, \exists \boldsymbol{b} \bigwedge_{j \in [m]} R_j^n(z_j, \boldsymbol{a}_j) \wedge \bigwedge_{j \in [p]} S_j^x(\boldsymbol{b}_j)$$

where $z_1, \ldots, z_m$ are fresh existential variables, one for every atom of an endogenous relation.

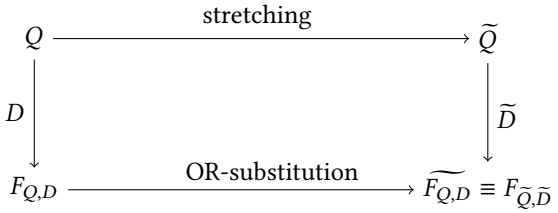
Example 5.2. The stretching of the non-hierarchical query

$$Q = \exists x \exists y\, R^n(x) \wedge S^x(x, y) \wedge T^n(y) \tag{10}$$

is

$$\widetilde{Q} = \exists x \exists y \exists z_1 \exists z_2\, R^n(z_1, x) \wedge S^x(x, y) \wedge T^n(z_2, y) \tag{11}$$

The relationship between a CQ Q, its stretching $\widetilde{Q}$, their lineages $F_{Q,D}$ and $F_{\widetilde{Q},\widetilde{D}}$ over databases D and $\widetilde{D}$, and the function $\widetilde{F_{Q,D}}$ obtained from $F_{Q,D}$ by OR-substitution, is depicted below:



In the bottom right node, $\widetilde{F_{Q,D}}$ and $F_{\widetilde{Q},\widetilde{D}}$ are equivalent and transformable into each other in polynomial time. The stretching at the query level captures the OR-substitutions at the lineage level. That is, the lineage of Q under OR-substitutions can be recovered via a polynomial-time transformation from the lineage of the stretching of Q and vice versa. This shows that the assumption made at the beginning of Section 3 holds for lineage: We can construct in polynomial time[4] a lineage for the stretched query from the lineage of the query under OR-substitutions.

Lemma 5.3. *$\widetilde{C_Q} \approx^P C_{\widetilde{Q}}$ holds for any CQ Q and its stretching $\widetilde{Q}$.*

Example 5.4. Consider the query $Q = \exists x R_1^n(x), R_2^n(x)$ and its stretching $\widetilde{Q} = \exists x \exists z_1 \exists z_2 R_1^n(z_1, x), R_2^n(z_2, x)$. We depict below a database $\boldsymbol{D}$ consisting of the relations R_1 and R_2 and a database $\widetilde{\boldsymbol{D}}$ consisting of the stretched relations. The variables Y_i and Z_i^j are associated to the database tuples.

$$\begin{array}{lll} \boldsymbol{D}: & R_1^n(x) & R_2^n(x) \\ & \underline{\quad x \quad} & \underline{\quad x \quad} \\ & Y_1 : a_1 & Y_3 : a_1 \\ & Y_2 : a_2 & Y_4 : a_2 \end{array}$$

$$\begin{array}{lll} \widetilde{\boldsymbol{D}}: & R_1^n(z_1, x) & R_2^n(z_2, x) \\ & \underline{\quad z_1 \quad x \quad} & \underline{\quad z_2 \quad x \quad} \\ & Z_1^1 : b_1^1 \quad a_1 & Z_3^1 : c_1^1 \quad a_1 \\ & \cdots \quad \cdots \quad \cdots & \cdots \quad \cdots \quad \cdots \\ & Z_1^m : b_1^m \quad a_1 & Z_3^p : c_1^p \quad a_1 \\ & Z_2^1 : b_2^1 \quad a_2 & Z_4^1 : c_2^1 \quad a_2 \\ & \cdots \quad \cdots \quad \cdots & \cdots \quad \cdots \quad \cdots \\ & Z_2^n : b_2^n \quad a_2 & Z_4^q : c_2^q \quad a_2 \end{array}$$

The lineage of Q over $\boldsymbol{D}$ is $F_{Q,D} = (Y_1 \wedge Y_3) \vee (Y_2 \wedge Y_4)$, hence $F_{Q,D} \in C_Q$. The lineage of $\widetilde{Q}$ over $\widetilde{\boldsymbol{D}}$ is $F_{\widetilde{Q},\widetilde{D}} = \bigvee_{i\in[m],j\in[p]}(Z_1^i \wedge Z_3^j) \vee \bigvee_{i\in[n],j\in[q]}(Z_2^i \wedge Z_4^j)$, hence $F_{\widetilde{Q},\widetilde{D}} \in C_{\widetilde{Q}}$. Under the OR-substitution $\theta = \{Y_1 := \bigvee_{i=1}^m Z_1^i, Y_2 := \bigvee_{i=1}^n Z_2^i, Y_3 := \bigvee_{i=1}^p Z_3^i, Y_4 := \bigvee_{i=1}^q Z_4^i\}$, we get $F_{Q,D}[\theta] = ((\bigvee_{i=1}^m Z_1^i) \wedge (\bigvee_{i=1}^p Z_3^i)) \vee ((\bigvee_{i=1}^n Z_2^i) \wedge (\bigvee_{i=1}^q Z_4^i))$. It holds $F_{Q,D}[\theta] \in \widetilde{C_Q}$. Observe that $F_{Q,D}[\theta] \equiv F_{\widetilde{Q},\widetilde{D}}$ and can be transformed into one another in quadratic time using the distributivity law for $\wedge$ over $\vee$ (the time is exponential in the number of endogenous relations).

Lemma 5.3 immediately implies the following polynomial-time equivalences between the three problems introduced in Section 3, now over classes of query lineage:

Corollary 5.5 (of Lemma 5.3). *For any CQ Q and its stretching $\widetilde{Q}$, the following polynomial-time equivalences hold:*

- $\mathsf{Shap}(\widetilde{C_Q}) \equiv^P \mathsf{Shap}(C_{\widetilde{Q}})$
- $\#\widetilde{C_Q} \equiv^P \#C_{\widetilde{Q}}$
- $\#_*\widetilde{C_Q} \equiv^P \#_*C_{\widetilde{Q}}$

[4]This is in polynomial time data complexity, so possibly exponential in the query size or equivalently in the arity of the clauses in the lineage.

For instance, if we want to compute $\mathsf{Shap}(\widetilde{F})$ for $\widetilde{F} \in \widetilde{C_Q}$, i.e., for Q's lineage under OR-substitutions, and have an oracle for $\mathsf{Shap}(C_{\widetilde{Q}})$, i.e., for computing the Shapley values for the lineage of Q's stretching $\widetilde{Q}$, we can first transform $\widetilde{F}$ in polynomial time into an equivalent function $F \in C_{\widetilde{Q}}$ and then compute $\mathsf{Shap}(F)$ using the oracle. Since $\widetilde{F} \equiv F$, we have $\mathsf{Shap}(\widetilde{F}) = \mathsf{Shap}(F)$.

Query stretching preserves the hierarchical property:

Lemma 5.6. *A CQ Q is hierarchical iff its stretching $\widetilde{Q}$ is hierarchical.*

5.3 Dichotomy for Self-Join-Free CQs

We prove the following dichotomy using our polynomial-time equivalences and lineage transformations:

Theorem 5.1 ([20]). *Let Q be a self-join-free CQ. If Q is hierarchical, then $\mathsf{Shap}(C_Q)$ is in FP, otherwise it is FP$^{\#P}$-hard.*

The hardness result holds for specific classes of databases, where we can choose conveniently the endogenous and exogenous relations, whereas the tractability result holds for any database. We first focus on hardness and later on tractability.

Hardness. We show that for any non-hierarchical CQ Q, there are specific classes of databases for which $\mathsf{Shap}(C_Q)$ is FP$^{\#\mathrm{P}}$-hard. We first show the hardness for the smallest non-hierarchical CQ and then generalize to arbitrary non-hierarchical CQs.

Let us consider the smallest non-hierarchical CQ Q in Eq. (10) and its stretching in Eq. (11), where we choose conveniently the relations R and T to be endogenous, while the relation S be exogenous. The class C_Q consists of all positive bipartite functions in disjunctive normal form: $\bigvee_{(i,j)\in S} X_i \wedge Y_j$, where X_i annotates tuple $R(i)$ and Y_j annotates tuple $T(j)$. Any such function can be obtained by appropriately picking R and T for the sets of variables X_i and Y_j, and S to encode its clauses. We next use a prior result on the #P-hardness for model counting for this class of functions [27]:

$$
\begin{aligned}
\#C_Q \leq^P \mathsf{Shap}(\widetilde{C_Q}) \qquad & \text{(by Theorem 3.1)} \\
\Rightarrow \#C_Q \leq^P \mathsf{Shap}(C_{\widetilde{Q}}) \qquad & \text{(by Corollary 5.5)} \\
\Rightarrow \#C_Q \leq^P \mathsf{Shap}(C_Q) \qquad & \text{(by Claim 5.2 below)} \\
\Rightarrow \mathsf{Shap}(C_Q) \text{ is FP}^{\#\mathrm{P}}\text{-hard} \qquad & (\#C_Q \text{ is \#P-hard [27]})
\end{aligned}
$$

Claim 5.2. *$C_{\widetilde{Q}} = C_Q$ holds for the non-hierarchical query Q in Eq. (10) and its stretching $\widetilde{Q}$ in Eq. (11).*

The proof of Claim 5.2 is in Appendix B.1.

The generalization to arbitrary non-hierarchical CQs is as in prior work [6, 20]. We reduce the computation of Q in Eq.(10) over any database $\boldsymbol{D}$ to the computation of any non-hierarchical query Q' over a specifically-designed database $\boldsymbol{D'}$ constructed from $\boldsymbol{D}$.

By definition, the non-hierarchical query Q has two variables x and y such that $at(x) \cap at(y) \neq \emptyset$, $at(x) \not\subseteq at(y)$, and $at(y) \not\subseteq at(x)$. We construct $\boldsymbol{D'}$ as follows. We pick two distinct atoms in Q, call them $\hat{R}$ and $\hat{T}$, such that: $\hat{R}$ has x and not y, and $\hat{T}$ has y and not x. We make the relations of these two atoms endogenous and all other relations exogenous. The x-column in $\hat{R}$ and the y-column in $\hat{T}$ are copies of the corresponding columns in R and T. The x and y columns in the other relations in $\boldsymbol{D'}$ are copies of the corresponding columns in S. The values for all other variables are set to a fixed dummy value. Then, the lineage of Q and Q' over $\boldsymbol{D}$ and respectively $\boldsymbol{D'}$ is the same: $F_{Q,D} = F_{Q',D'}$. The hardness of $\#C_Q$ thus transfers to $\#C_{Q'}$.

Tractability. We show that $\mathsf{Shap}(C_Q)$ is in FP for any hierarchical CQ Q. We use that $\#C_Q$ is tractable for any hierarchical Q [26]:

Claim 5.3. *For any hierarchical CQ Q, $\#_*C_Q$ is in FP.*

Proof.

$$
\begin{aligned}
& Q \text{ is hierarchical} \\
\Rightarrow\ & \widetilde{Q} \text{ is hierarchical} && \text{(by Lemma 5.6)} \\
\Rightarrow\ & \#C_{\widetilde{Q}} \text{ is in FP} && \text{(by [26])} \\
\Rightarrow\ & \#\widetilde{C_Q} \text{ is in FP} && \text{(by Corollary 5.5)} \\
\Rightarrow\ & \#_*C_Q \text{ is in FP} && \text{(by Theorem 3.1)}
\end{aligned}
$$

□

Tractability of $\mathsf{Shap}(C_Q)$ is now an immediate implication:

$$
\begin{aligned}
& Q \text{ is hierarchical} \\
\Rightarrow\ & \widetilde{Q} \text{ is hierarchical} && \text{(by Lemma 5.6)} \\
\Rightarrow\ & \#_*C_{\widetilde{Q}} \text{ is in FP} && \text{(by Claim 5.3)} \\
\Rightarrow\ & \#_*\widetilde{C_Q} \text{ is in FP} && \text{(by Corollary 5.5)} \\
\Rightarrow\ & \mathsf{Shap}(C_Q) \text{ is in FP} && \text{(by Theorem 3.1)}
\end{aligned}
$$

Discussion. The above hardness proof is significantly simpler than the original one [20], which solves several instances of computing the number of independent sets of a given bipartite graph and assembles them in a full-rank set of linear equations. In fact, the original proof questions[5] whether a simple proof based on the hardness of model counting for positive bipartite DNF, as used to show the hardness of the non-hierarchical queries over probabilistic databases and also used in our proof above, is even possible. Our result settles this question in the affirmative.

6 CONCLUSION AND FUTURE WORK

In this paper we give a polynomial-time equivalence between computing Shapley values and model counting for any class of Boolean functions that are closed under substitutions of variables with disjunctions of fresh variables. This result settles an open problem raised in prior work. We also show two direct applications of our result: tractability of Shapley value computation for deterministic and decomposable circuits and the dichotomy for Shapley value computation in case of self-join-free Boolean conjunctive queries. We conjecture that our work can be instrumental to show that the dichotomy for unions of conjunctive queries in probabilistic databases [7] also applies to Shapley value computation.

ACKNOWLEDGMENTS

The authors would like to acknowledge grants from the UZH Global Strategy and Partnerships Funding Scheme, NSF-BSF 2109922, and NSF IIS 2314527. Part of this work was conducted while the authors participated in the Simons Program on Logic and Algorithms in Databases and AI. Ahmet Kara and Dan Olteanu would like to acknowledge Daniel Deutch, who introduced them to the topic of Shapley values in databases.

[5]First paragraph in the proof of Proposition 4.6 [20]: "It is not at all clear to us how such an approach can work in our case and, indeed, our proof is more involved".

REFERENCES

[1] Marcelo Arenas, Pablo Barceló, Leopoldo E. Bertossi, and Mikaël Monet. 2021. The Tractability of SHAP-Score-Based Explanations for Classification over Deterministic and Decomposable Boolean Circuits. In *AAAI*. 6670–6678.

[2] Marcelo Arenas, Pablo Barceló, Leopoldo E. Bertossi, and Mikaël Monet. 2023. On the Complexity of SHAP-Score-Based Explanations: Tractability via Knowledge Compilation and Non-Approximability Results. *J. Mach. Learn. Res.* 24 (2023), 63:1–63:58. http://jmlr.org/papers/v24/21-0389.html

[3] Marcelo Arenas, Pablo Barceló, Miguel A. Romero Orth, and Bernardo Subercaseaux. 2022. On Computing Probabilistic Explanations for Decision Trees. In *NeurIPS*.

[4] Joshua Brakensiek, Venkatesan Guruswami, and Sai Sandeep. 2023. Conditional Dichotomy of Boolean Ordered Promise CSPs. *TheoretiCS* 2 (2023). https://doi.org/10.46298/theoretics.23.2

[5] Johann Brault-Baron, Florent Capelli, and Stefan Mengel. 2015. Understanding Model Counting for Beta-Acyclic CNF-Formulas. In *STACS*, Vol. 30. 143–156. https://doi.org/10.4230/LIPIcs.STACS.2015.143

[6] Nilesh N. Dalvi and Dan Suciu. 2004. Efficient Query Evaluation on Probabilistic Databases. In *VLDB*. 864–875. https://doi.org/10.1016/B978-012088469-8.50076-0

[7] Nilesh N. Dalvi and Dan Suciu. 2012. The Dichotomy of Probabilistic Inference for Unions of Conjunctive Queries. *J. ACM* 59, 6 (2012), 30:1–30:87. https://doi.org/10.1145/2395116.2395119

[8] Adnan Darwiche and Pierre Marquis. 2002. A Knowledge Compilation Map. *J. Artif. Intell. Res.* 17 (2002), 229–264. https://doi.org/10.1613/jair.989

[9] Adnan Darwiche, Pierre Marquis, Dan Suciu, and Stefan Szeider. 2017. Recent Trends in Knowledge Compilation (Dagstuhl Seminar 17381). *Dagstuhl Reports* 7, 9 (2017), 62–85. https://doi.org/10.4230/DagRep.7.9.62

[10] Susan B. Davidson, Daniel Deutch, Nave Frost, Benny Kimelfeld, Omer Koren, and Mikaël Monet. 2022. ShapGraph: An Holistic View of Explanations through Provenance Graphs and Shapley Values. In *SIGMOD*. 2373–2376. https://doi.org/10.1145/3514221.3520172

[11] Guy Van den Broeck, Anton Lykov, Maximilian Schleich, and Dan Suciu. 2022. On the Tractability of SHAP Explanations. *J. Artif. Intell. Res.* 74 (2022), 851–886. https://doi.org/10.1613/jair.1.13283

[12] Daniel Deutch, Nave Frost, Benny Kimelfeld, and Mikaël Monet. 2022. Computing the Shapley Value of Facts in Query Answering. In *SIGMOD*. 1570–1583. https://doi.org/10.1145/3514221.3517912

[13] Gene H. Golub and Charles F. Van Loan. 1996. *Matrix Computations, Third Edition.* Johns Hopkins University Press.

[14] Carla P. Gomes, Ashish Sabharwal, and Bart Selman. 2021. Model Counting. In *Handbook of Satisfiability - Second Edition*. Vol. 336. 993–1014. https://doi.org/10.3233/FAIA201009

[15] Todd J. Green, Gregory Karvounarakis, and Val Tannen. 2007. Provenance Semirings. In *PODS*. 31–40. https://doi.org/10.1145/1265530.1265535

[16] Peter L. Hammer, Alexander Kogan, and Uriel G. Rothblum. 2000. Evaluation, Strength, and Relevance of Variables of Boolean Functions. *SIAM J. Discret. Math.* 13, 3 (2000), 302–312. https://doi.org/10.1137/S089548019732787X

[17] Hans Harder, Simon Jantsch, Christel Baier, and Clemens Dubslaff. 2023. A Unifying Formal Approach to Importance Values in Boolean Functions. *CoRR* abs/2305.08103 (2023). https://doi.org/10.48550/arXiv.2305.08103

[18] Gil Kalai. 2004. Social indeterminacy. *Econometrica* 72, 5 (2004), 1565–1581.

[19] Richard M. Karp, Michael Luby, and Neal Madras. 1989. Monte-Carlo Approximation Algorithms for Enumeration Problems. *J. Algorithms* 10, 3 (1989), 429–448. https://doi.org/10.1016/0196-6774(89)90038-2

[20] Ester Livshits, Leopoldo E. Bertossi, Benny Kimelfeld, and Moshe Sebag. 2021. The Shapley Value of Tuples in Query Answering. *Log. Methods Comput. Sci.* 17, 3 (2021). https://doi.org/10.46298/lmcs-17(3:22)2021

[21] Richard T. B. Ma, Dah-Ming Chiu, John Chi-Shing Lui, Vishal Misra, and Dan Rubenstein. 2010. Internet Economics: The Use of Shapley Value for ISP Settlement. *IEEE/ACM Trans. Netw.* 18, 3 (2010), 775–787. https://doi.org/10.1109/TNET.2010.2049205

[22] Richard T. B. Ma, Dah-Ming Chiu, John C. S. Lui, Vishal Misra, and Dan Rubenstein. 2008. Interconnecting Eyeballs to Content: A Shapley Value Perspective on ISP Peering and Settlement. In *SIGCOMM*. 61–66. https://doi.org/10.1145/1403027.1403041

[23] Dang Minh, H. Xiang Wang, Y. Fen Li, and Tan N. Nguyen. 2022. Explainable Artificial Intelligence: A Comprehensive Review. *Artif. Intell. Rev.* 55, 5 (2022), 3503–3568. https://doi.org/10.1007/s10462-021-10088-y

[24] Stefano Moretti, Vito Fragnelli, Fioravante Patrone, and Stefano Bonassi. 2010. Using Coalitional Games on Biological Networks to Measure Centrality and Power of Genes. *Bioinform.* 26, 21 (2010), 2721–2730. https://doi.org/10.1093/bioinformatics/btq508

[25] Ramasuri Narayanam and Yadati Narahari. 2011. A Shapley Value-Based Approach to Discover Influential Nodes in Social Networks. *IEEE Trans Autom. Sci. Eng.* 8, 1 (2011), 130–147. https://doi.org/10.1109/TASE.2010.2052042

[26] Dan Olteanu and Jiewen Huang. 2008. Using OBDDs for Efficient Query Evaluation on Probabilistic Databases. In *SUM 2008*. 326–340. https://doi.org/10.1007/978-3-540-87993-0_26

[27] J. Scott Provan and Michael O. Ball. 1983. The Complexity of Counting Cuts and of Computing the Probability that a Graph is Connected. *SIAM J. Comput.* 12, 4 (1983), 777–788. https://doi.org/10.1137/0212053
[28] Alon Reshef, Benny Kimelfeld, and Ester Livshits. 2020. The Impact of Negation on the Complexity of the Shapley Value in Conjunctive Queries. In *PODS*. 285–297. https://doi.org/10.1145/3375395.3387664
[29] Alvin E Roth. 1988. *The Shapley Value: Essays in Honor of Lloyd S. Shapley.* Cambridge University Press.
[30] Benedek Rozemberczki, Lauren Watson, Péter Bayer, Hao-Tsung Yang, Oliver Kiss, Sebastian Nilsson, and Rik Sarkar. 2022. The Shapley Value in Machine Learning. In *IJCAI*. 5572–5579. https://doi.org/10.24963/ijcai.2022/778
[31] L. S. Shapley. 1953. *A Value for n-Person Games.* Princeton University Press, 307–318. https://doi.org/doi:10.1515/9781400881970-018
[32] Dan Suciu, Dan Olteanu, Christopher Ré, and Christoph Koch. 2011. *Probabilistic Databases.* Morgan & Claypool Publishers. https://doi.org/10.2200/S00362ED1V01Y201105DTM016
[33] Tjeerd van Campen, Herbert Hamers, Bart Husslage, and Roy Lindelauf. 2018. A New Approximation Method for the Shapley Value Applied to the WTC 9/11 Terrorist Attack. *Soc. Netw. Anal. Min.* 8, 1 (2018), 3:1–3:12. https://doi.org/10.1007/s13278-017-0480-z

A MISSING DETAILS IN SECTION 2

A.1 Proof of Proposition 2.3

Proposition 2.3 ([20] page 11, adapted). *The* Shapley value *of a variable X_i of a Boolean function F is:*

$$\mathsf{Shap}(F, X_i) = \sum_{k=0}^{n-1} c_k \left(\#_k F[X_i := 1] - \#_k F[X_i := 0]\right)$$

where $c_k = \frac{k!(n-k-1)!}{n!}$.

We show this proposition as follows:

$$\begin{aligned}
&\mathsf{Shap}(F, X_i) \\
&\overset{(a)}{=} \frac{1}{n!} \sum_{\Pi \in S_n} \left(F[\Pi^{<i} \cup \{i\}] - F[\Pi^{<i}]\right) \\
&\overset{(b)}{=} \frac{1}{n!} \sum_{T \subseteq [n]-\{i\}} |T|!(n-|T|-1)!\,(F[T \cup \{i\}] - F[T]) \\
&\overset{(c)}{=} \frac{1}{n!} \sum_{k=0}^{n-1} |k|!(n-|k|-1)!\,(\#_k F[X_i := 1] - \#_k F[X_i := 0]) \\
&\overset{(d)}{=} \sum_{k=0}^{n-1} c_k \left(\#_k F[X_i := 1] - \#_k F[X_i := 0]\right)
\end{aligned}$$

Equality (a) holds by definition. We obtain Equality (b) by grouping the sum by possible sets $T \subseteq [n] - \{i\}$ and scaling the result of $F[T \cup \{i\}] - F[T]$ by the number of permutations of the set $\{1, \ldots, n\}$ that start with the values in T followed by i. Observe that $|T|!(n-|T|-1)!$ is the number of permutations of the set $\{1, \ldots, n\}$ that start with the values in T followed by i. To obtain Equality (c), we iterate over the sizes of possible sets $T \subseteq [n] - \{i\}$ and observe that the number of sets $T \subseteq [n] - \{i\}$ of size k such that $F[T \cup \{i\}] = 1$ is exactly $\#_k F[X_i := 1]$; similarly, the number of sets T of size k such that $F[T] = 1$ is $\#_k F[X_i := 0]$. We obtain Equality (d) by moving $\frac{1}{n!}$ inside the sum and replacing $\frac{k!(n-k-1)!}{n!}$ by c_k.

A.2 Proof of Proposition 2.5

Proposition 2.5. *For any Boolean function F, it holds*

$$\sum_{i\in[n]} \mathsf{Shap}(F, X_i) = F[\mathbf{1}] - F[\mathbf{0}]$$

where $\mathbf{1}$ *is the valuation that maps all variables to* 1*, and* $\mathbf{0}$ *the valuation that maps all variables to* 0*.*

We show this proposition as follows:

$$\begin{aligned}
\sum_{i=1}^{n} \mathsf{Shap}(F, X_i) &\overset{(a)}{=} \sum_{i=1}^{n}\sum_{k=0}^{n-1} c_k \left(\#_k F[X_i := 1] - \#_k F[X_i := 0]\right) \\
&= \sum_{k=0}^{n-1}\sum_{i=1}^{n} c_k \left(\#_k F[X_i := 1] - \#_k F[X_i := 0]\right) \\
&\overset{(b)}{=} \sum_{k=0}^{n-1} \left(c_k(k+1)\#_{k+1}F - c_k(n-k)\#_k F\right) \\
&= c_0 \cdot \#_1 F - \underline{c_0 \cdot n \cdot \#_0 F} + \\
&\quad c_1 \cdot 2 \cdot \#_2 F - c_1 \cdot (n-1) \cdot \#_1 F \; + \cdots + \\
&\quad \overline{c_{n-1} \cdot n \cdot \#_n F} - c_{n-1} \cdot \#_{n-1} F \\
&\overset{(c)}{=} \overline{c_{n-1} \cdot n \cdot \#_n F} - \underline{c_0 \cdot n \cdot \#_0 F} + \\
&\quad \sum_{k=0}^{n-2} \left(c_k(k+1)\#_{k+1}F - c_{k+1}(n-k-1)\#_{k+1}F\right) \\
&\overset{(d)}{=} c_{n-1} \cdot n \cdot \#_n F - c_0 \cdot n \cdot \#_0 F \\
&\overset{(e)}{=} F[\mathbf{1}] - F[\mathbf{0}]
\end{aligned}$$

Equality (a) uses the Shapley value characterization given in Proposition 2.3. Equality (b) follows from the two Equalities (7) and (8) in Section 3.1. We obtain Equality (c) by regrouping the terms on the left-hand side: We keep $c_{n-1} \cdot n \cdot \#_n F - c_0 \cdot n \cdot \#_0 F$ outside the scope of the sum and pair the terms $c_k(k+1)\#_{k+1}F$ and $c_{k+1}(n-k-1)\#_{k+1}F$ for $0 \le k \le n-2$ within the scope of the sum. Equality (d) holds, since for each k, the two terms within the scope of the sum cancel each other. This cancelling is due to the following equalities: $c_k(k+1) = \frac{k!(n-k-1)!}{n!}(k+1) = \frac{(k+1)!(n-k-1)!}{n!} = \frac{(k+1)!(n-k-2)!}{n!}(n-k-1) = c_{k+1}(n-k-1)$. Equality (e) follows from the equalities $c_{n-1} \cdot n = c_0 \cdot n = \frac{(n-1)!}{n!}n = 1$ and the observation that F can have at most one model of size n and at most one model of size 0.

B MISSING DETAILS IN SECTION 5

We introduce notation used in the following. Given a relation R over some attributes $(y_1, \ldots, y_n)$, we write $(y_1 : a_1, \ldots, y_n : a_n)$ to denote a tuple in R where the y_i-value is a_i for $i \in [n]$.

B.1 Proof of Claim 5.2

Claim 5.2. $C_{\widetilde{Q}} = C_Q$ *holds for the non-hierarchical query* Q *in Eq.* (10) *and its stretching* $\widetilde{Q}$ *in Eq.* (11)*.*

We first illustrate how we can construct databases to show that each lineage in C_Q is also a lineage in $C_{\widetilde{Q}}$ and vice versa.

Example B.1. Consider the following database $\boldsymbol{D}$, where the variables Y_i preceding the tuples in endogenous relations are associated to the tuples.

$\boldsymbol{D}$:

	$R^n(x)$
	x
Y_1 :	a_1
Y_2 :	a_2

$S^x(x,y)$	
x	y
a_1	b_1
a_2	b_2

$T^n(y)$	
	y
Y_3 :	b_1
Y_4 :	b_2

The lineage of Q over $\boldsymbol{D}$ is $F_{Q,\boldsymbol{D}} = (Y_1 \wedge Y_3) \vee (Y_2 \wedge Y_4)$. Hence, it holds $F_{Q,\boldsymbol{D}} \in C_Q$. Now, we construct from $\boldsymbol{D}$ a database $\widetilde{\boldsymbol{D}}$ such that $F_{Q,\boldsymbol{D}}$ is the lineage of $\widetilde{Q}$ over $\widetilde{\boldsymbol{D}}$. The idea is to assign to the fresh attributes added due to stretching a dummy value d:

$\widetilde{\boldsymbol{D}}$:

	$\widetilde{R}^n(z_1,x)$	
	z_1	x
Y_1 :	d	a_1
Y_2 :	d	a_1

$S^x(x,y)$	
x	y
a_1	b_1
a_2	b_2

$\widetilde{T}^n(z_2,y)$	
z_2	y
d	b_1
d	b_2

Now, consider the following database $\widetilde{\boldsymbol{D}}'$ with stretched relations:

$\widetilde{\boldsymbol{D}}'$:

	$\widetilde{R}^n(z_1,x)$	
	z_1	x
Y_1 :	d_1	a
Y_2 :	d_2	a

$S^x(x,y)$	
x	y
a	b

$\widetilde{T}^n(z_2,y)$	
z_2	y
d_1	b
d_2	b

The lineage of $\widetilde{Q}$ over $\widetilde{\boldsymbol{D}}'$ is $F_{\widetilde{Q},\widetilde{\boldsymbol{D}}'} = (Y_1 \wedge Y_3) \vee (Y_1 \wedge Y_4) \vee (Y_2 \wedge Y_3) \vee (Y_2 \wedge Y_4)$. It holds $F_{\widetilde{Q},\widetilde{\boldsymbol{D}}'} \in C_{\widetilde{Q}}$. We construct now from $\widetilde{\boldsymbol{D}}'$ a database $\boldsymbol{D}'$ such that $F_{\widetilde{Q},\widetilde{\boldsymbol{D}}'}$ is a lineage of Q over $\boldsymbol{D}'$. The idea is to represent tuples over (z_1, x) and (z_2, y) as single (composite) values over x and respectively y and construct S such that the combinations of (z_1, x) and (z_2, y) remain the same as in $\widetilde{\boldsymbol{D}}'$:

$\boldsymbol{D}'$:

	$R^n(x)$
	x
Y_1 :	(d_1, a)
Y_2 :	(d_2, a)

$S^x(x,y)$	
x	y
(d_1, a)	(d_1, b)
(d_1, a)	(d_2, b)
(d_2, a)	(d_1, b)
(d_2, a)	(d_2, b)

$T^n(y)$	
	y
Y_3 :	(d_1, b)
Y_4 :	(d_2, b)

Next, we prove Claim 5.2 formally. Consider the non-hierarchical CQ $Q = \exists x \exists y\, R^n(x) \wedge S^x(x,y) \wedge T^n(y)$ in Eq. (10) and its stretching $\widetilde{Q} = \exists x \exists y \exists z_1 \exists z_2\, R^n(z_1, x) \wedge S^x(x,y) \wedge T^n(z_2, y)$ in Eq. (11). We first show that $C_Q \subseteq C_{\widetilde{Q}}$ and then we show $C_{\widetilde{Q}} \subseteq C_Q$.

B.1.1 $C_Q \subseteq C_{\widetilde{Q}}$. Consider the lineage $F_{Q,\boldsymbol{D}} \in C_Q$ for a database $\boldsymbol{D} = \{R^n, S^x, T^n\}$. We show that $F_{Q,\boldsymbol{D}} \in C_{\widetilde{Q}}$. We construct from $\boldsymbol{D}$ a database $\widetilde{\boldsymbol{D}} = \{\widetilde{R}^n, S^x, \widetilde{T}^n\}$ as follows. Assume that R^n is defined over the attribute x, S^x is defined over the attributes (x, y), and T^n is defined over the attribute y. Relation S^x remains unchanged. We transform relation R^n into the relation $\widetilde{R}^n$ over the attributes (z_1, x) for a new attribute z_1. The relation $\widetilde{R}^n$ consists of the tuples $\{(z_1 : d, x : a) | (x : a) \in R^n\}$, where d is a fresh dummy value. If a variable in $F_{Q,\boldsymbol{D}}$ is associated with the tuple $(x : a)$ in R^n, we associate the same variable with the tuple $(z_1 : d, x : a)$ in $\widetilde{R}^n$. Similarly, we transform relation T^n into the relation $\widetilde{T}^n$ over the attributes (z_2, y) for a new attribute z_2. The relation $\widetilde{T}^n$ consists of the tuples $\{(z_2 : d, y : b) | (y : b) \in T^n\}$. If a variable in $F_{Q,\boldsymbol{D}}$ is associated with the tuple $(y : b)$ in T^n, we associate the same variable with the tuple $(z_2 : d, y : b)$ in $\widetilde{T}^n$. Observe that $F_{Q,\boldsymbol{D}}$ is the lineage of $\widetilde{Q}$ over $\widetilde{\boldsymbol{D}}$, which means that $F_{Q,\boldsymbol{D}} \in C_{\widetilde{Q}}$.

B.1.2 $C_{\widetilde{Q}} \subseteq C_Q$. This direction is analogous to the one shown in the previous section. Consider the lineage $F_{\widetilde{Q},\widetilde{D}} \in C_{\widetilde{Q}}$ for some database $\widetilde{\boldsymbol{D}} = \{\widetilde{R}^n, S^x, \widetilde{T}^n\}$. We show that $F_{\widetilde{Q},\widetilde{D}} \in C_Q$. We start with constructing a database $\boldsymbol{D} = \{R^n, S^x_{\text{new}}, T^n\}$ from $\widetilde{\boldsymbol{D}}$. Observe that in contrast to the construction in Section B.1.1, we change also the relation S^x. Assume that $\widetilde{R}^n$, S^x, and $\widetilde{T}^n$ in $\widetilde{\boldsymbol{D}}$ are defined over the attributes (z_1, x), (x, y), and respectively (z_2, y). We denote the value domains of the attributes z_1, x, z_2, and y by $\text{Dom}(z_1)$, $\text{Dom}(x)$, $\text{Dom}(z_2)$, and respectively $\text{Dom}(y)$. We construct from $\widetilde{R}^n$ the relation R^n over the attribute x' with domain $\text{Dom}(x') = \text{Dom}(z_1) \times \text{Dom}(x)$. We define $R^n = \{(x' : (a', a)) | (z_1 : a', x : a) \in \widetilde{R}^n\}$. If a variable in $F_{\widetilde{Q},\widetilde{D}}$ is associated with the tuple $(z_1 : a', x : a)$ in $\widetilde{R}^n$, we associate it with the tuple $(x' : (a', a))$ in R^n. Analogously, we construct from $\widetilde{T}^n$ the relation T^n over the attribute y' with domain $\text{Dom}(y') = \text{Dom}(z_2) \times \text{Dom}(y)$. We set $T^n = \{(y' : (b', b)) | (z_2 : b', y : b) \in \widetilde{T}^n\}$. If a variable is associated with the tuple $(z_2 : b', y : b)$ in $\widetilde{T}^n$, we associate it with the value $(y : (b', b))$ in T^n. Finally, we construct from relation S^x the relation S^x_{new} over the attributes (x', y') such that $S^x_{\text{new}} = \{(x' : (a', a), y' : (b', b)) | (z_1 : a', x : a) \in \widetilde{R}^n, (x : a, y : b) \in S^x, \text{ and } (z_2 : b', y : b) \in \widetilde{T}^n\}$. Observe that $F_{\widetilde{Q},\widetilde{D}}$ is the lineage of Q over $\boldsymbol{D}$. This means that $F_{\widetilde{Q},\widetilde{D}} \in C_Q$.

B.2 Proof of Lemma 5.3

Lemma 5.3. *$\widetilde{C_Q} \approx^P C_{\widetilde{Q}}$ holds for any CQ Q and its stretching $\widetilde{Q}$.*

The high-level idea of the bidirectional transformation is as follows: Consider the lineage $F_{Q,D}$ of Q over a database $\boldsymbol{D}$ and a variable X associated with a tuple $\boldsymbol{t} = (\boldsymbol{x} : \boldsymbol{a})$ in an endogenous relation R^n. Assume that $\widetilde{F}_{Q,D}$ results from $F_{Q,D}$ by substituting X with the disjunction $Z_1 \vee \cdots \vee Z_\ell$. Now, consider the database $\widetilde{\boldsymbol{D}}$ that results from $\boldsymbol{D}$ by stretching $R^n(\boldsymbol{x})$ into $\widetilde{R}^n(z, \boldsymbol{x})$ and replacing $\boldsymbol{t} = (\boldsymbol{x} : \boldsymbol{a})$ with ℓ new tuples $\boldsymbol{t}_1 = (z : a_1, \boldsymbol{x} : \boldsymbol{a}), \ldots, \boldsymbol{t}_\ell = (z : a_\ell, \boldsymbol{x} : \boldsymbol{a})$ where $a_1, \ldots, a_\ell$ are fresh values. Then, $\widetilde{F}_{Q,D}$ is equivalent to the lineage $F_{\widetilde{Q},\widetilde{D}}$ of $\widetilde{Q}$ over $\widetilde{\boldsymbol{D}}$ and can be obtained from it in polynomial time (data complexity).

We now explain the transformations in more detail. Consider a CQ Q and its stretching $\widetilde{Q}$. In Section B.2.1 we show that $C_{\widetilde{Q}} \lesssim^P \widetilde{C_Q}$ and in Section B.2.2 we show that $\widetilde{C_Q} \lesssim^P C_{\widetilde{Q}}$.

B.2.1 $C_{\widetilde{Q}} \lesssim^P \widetilde{C_Q}$. We describe a polynomial-time algorithm A that transforms any function $F_{\widetilde{Q},\widetilde{D}} \in C_{\widetilde{Q}}$ into an equivalent function from $\widetilde{C_Q}$, for some database $\widetilde{\boldsymbol{D}}$. The algorithm A first constructs from $\widetilde{\boldsymbol{D}}$ a database $\boldsymbol{D}$, where the attributes added by stretching are discarded. Then, it transforms $F_{\widetilde{Q},\widetilde{D}}$ into an equivalent function $\widetilde{F}$ in polynomial time such that $F_{Q,D} \overset{OR}{\to} \widetilde{F}$, which means $\widetilde{F} \in \widetilde{C_Q}$. In the following, we first describe the construction of $\boldsymbol{D}$, then we give the definition of $\widetilde{F}$, and finally explain the transformation from $F_{\widetilde{Q},D}$ into $\widetilde{F}$.

Construction of $\boldsymbol{D}$. The exogenous relations in $\widetilde{\boldsymbol{D}}$ remain unchanged. The algorithm replaces each endogenous relation $\widetilde{R}^n$ in $\widetilde{\boldsymbol{D}}$ with an endogenous relation R^n constructed as follows. Let $(z, \boldsymbol{y})$ be the attributes of $\widetilde{R}^n$ where z is the attribute added due to stretching. We set $R^n = \pi_{\boldsymbol{y}}\widetilde{R}$, i.e., R^n is the projection of $\widetilde{R}$ onto $\boldsymbol{y}$. Given a value tuple $\boldsymbol{t}$ over the variables $\boldsymbol{y}$, let $\boldsymbol{Z}$ be the set of variables associated to the tuples in $\widetilde{R}$ whose projection onto $\boldsymbol{y}$ is $\boldsymbol{t}$. The algorithm associates the fresh variable $X_{\boldsymbol{Z}}$ to the tuple $\boldsymbol{t}$ in R. The construction time is linear in the size of $\widetilde{\boldsymbol{D}}$.

Definition of $\widetilde{F}$. Let us denote the set of variables in $F_{Q,D}$ by $\boldsymbol{X}$. We define the substitution $\theta = \{X_{\boldsymbol{Z}} := \bigvee_{Z \in \boldsymbol{Z}} Z | X_{\boldsymbol{Z}} \in \boldsymbol{X}\}$ and set $\widetilde{F} = F_{Q,D}[\theta]$. It follows $F_{Q,D} \overset{OR}{\to} \widetilde{F}$.

Transformation of $F_{\widetilde{Q},\widetilde{D}}$ into $\widetilde{F}$. The algorithm first constructs from $\boldsymbol{D}$ a database $\boldsymbol{D}'$ where each lineage variable X_Z is replaced by the disjunction $\bigvee_{Z\in\boldsymbol{Z}} Z$. It then computes the lineage $F_{Q,D'}$ of Q over $\boldsymbol{D}'$. By construction, it holds $F_{Q,D'} = \widetilde{F}$ and $F_{Q,D'} \equiv F_{\widetilde{Q},\widetilde{D}}$. The construction of $F_{Q,D'}$ requires the computation of the join of the relations in $\boldsymbol{D}'$, which can be done in time polynomial in the size of $\boldsymbol{D}'$ (hence, polynomial in the size of $\widetilde{\boldsymbol{D}}$) using any conventional join algorithm.

We conclude that the overall transformation from $F_{\widetilde{Q},\widetilde{D}}$ into $\widetilde{F}$ takes time polynomial in the size of $F_{\widetilde{Q},\widetilde{D}}$ and $\widetilde{\boldsymbol{D}}$.

B.2.2 $\widetilde{C_Q} \lesssim^P C_{\widetilde{Q}}$. We give a polynomial-time algorithm B that transforms functions in $\widetilde{C_Q}$ into equivalent functions in $C_{\widetilde{Q}}$. Let $\widetilde{F} \in \widetilde{C_Q}$. Hence, there is a database $\boldsymbol{D}$ and an OR-substitution θ such that $F_{Q,D}[\theta] = \widetilde{F}$. We first explain how algorithm B transforms $\widetilde{F}$ in polynomial time into an equivalent function $\widetilde{F}'$ in DNF. Then, we show that $\widetilde{F}' \in C_{\widetilde{Q}}$, which concludes the proof.

Transformation of $\widetilde{F}$ into $\widetilde{F}'$ in DNF. Assume that $F_{Q,D} = C_1 \vee \cdots \vee C_p$ where each C_i is the conjunction of the variables in some set $\boldsymbol{X}_i$. We set $\boldsymbol{X} = \bigcup_{i=1}^{p} \boldsymbol{X}_i$. Assume that θ is defined as $\{X := \bigvee_{Z\in\boldsymbol{Z}_X} Z | X \in \boldsymbol{X}\}$, where for each $X \in \boldsymbol{X}$, $\boldsymbol{Z}_X$ is a set of fresh variables. This means that $\widetilde{F} = C'_1 \vee \cdots \vee C'_p$, where

$$C'_i = \bigwedge_{X\in\boldsymbol{X}_i} \bigvee_{Z\in\boldsymbol{Z}_X} Z.$$

Algorithm B transform each such C'_i into a disjunction C''_i of conjunctions. Assume that $\boldsymbol{X}_i = X_1 \wedge \cdots \wedge X_m$. Then,

$$C''_i = \bigvee_{Z_1\in\boldsymbol{X}_1,\ldots,Z_m\in\boldsymbol{X}_m} \bigwedge_{j=1}^{m} Z_j.$$

The algorithms sets $\widetilde{F}' = C''_1 \vee \cdots \vee C''_p$. The equivalence follows from the distributivity of $\vee$ over $\wedge$. The transformation can be done in time polynomial in the size of $\widetilde{F}$.

$\widetilde{F}'$ is included in $C_{\widetilde{Q}}$. We turn $\boldsymbol{D}$ into a database $\widetilde{\boldsymbol{D}}$ such that the lineage of $\widetilde{Q}$ over $\widetilde{\boldsymbol{D}}$ is equal to $\widetilde{F}'$. The exogenous relations in $\boldsymbol{D}$ remain unchanged. For each endogenous relation R^n over the attributes $\boldsymbol{y}$, we construct a relation $\widetilde{R}^n$ over $(z, \boldsymbol{y})$. For each value tuple $\boldsymbol{t}$ in R^n associated with the lineage variable X, we add the following new tuples to $\widetilde{R}^n$. Let $\theta(X) = Z_1 \vee \cdots \vee Z_\ell$. We add to $\widetilde{R}^n$ the tuples $\boldsymbol{t}_1, \ldots, \boldsymbol{t}_\ell$, where each $\boldsymbol{t}_i$ results from $\boldsymbol{t}$ by adding a fresh value for attribute z. We associate the tuples $\boldsymbol{t}_1, \ldots, \boldsymbol{t}_\ell$ in $\widetilde{R}^n$ with the lineage variables $Z_1, \ldots, Z_\ell$, respectively. It follows from the construction of $\widetilde{\boldsymbol{D}}$ that the lineage $F_{\widetilde{Q},\widetilde{D}}$ is equal to $\widetilde{F}'$. Hence, $\widetilde{F}' \in C_{\widetilde{Q}}$.

B.3 Proof of Lemma 5.6

Lemma 5.6. *A CQ Q is hierarchical iff its stretching $\widetilde{Q}$ is hierarchical.*

The main idea is that the class of hierarchical queries is closed under adding or removing variables that are contained in a single atom. To prove this formally, we first recall that for any two variables x and y in a hierarchical query, one of the following three properties (called *hierarchical properties* in the following) must hold: $at(x) \cap at(y) = \emptyset$, $at(x) \subseteq at(y)$, or $at(y) \subseteq at(x)$. We show each direction of Lemma 5.6 separately.

"$\Rightarrow$"-direction: Assume that Q is hierarchical. If $\widetilde{Q}$ does not contain any fresh variable, then it is obviously hierarchical. So, let x be a fresh variable in $\widetilde{Q}$ that does not appear in Q and let y be an arbitrary variable in $\widetilde{Q}$. By the definition of stretching, we have $|at(x)| = 1$. For the sake of

contradiction, assume that the hierarchical properties do not hold for x and y in $\widetilde{Q}$. This means that $at(x) \cap at(y) \neq \emptyset$, yet $at(x) \not\subseteq at(y)$ and $at(y) \not\subseteq at(x)$. So x appears in at least two atoms and the same for y. This contradicts our assumption that $|at(x)| = 1$. Hence, $\widetilde{Q}$ must be hierarchical.

"$\Leftarrow$"-direction: Assume that $\widetilde{Q}$ is hierarchical. Consider two distinct variables x and y in Q. By the definition of stretching, these variables must also appear in $\widetilde{Q}$. Since $\widetilde{Q}$ is hierarchical, one of the three hierarchical properties must hold for x and y in $\widetilde{Q}$. Since stretching only extends *existing* atoms, $at(x)$ and $at(y)$ are the same for both Q and $\widetilde{Q}$. This means that one of the three hierarchical properties must also hold for x and y in Q. This implies that Q is hierarchical.

Received June 2023; revised August 2023; accepted September 2023