A Dichotomy on the Complexity of Consistent Query Answering for Atoms with Simple Keys

Paraschos Koutris and Dan Suciu
{pkoutris,suciu}@cs.washington.edu
University of Washington

ABSTRACT

We study the problem of consistent query answering under primary key violations. In this setting, the relations in a database violate the key constraints and we are interested in maximal subsets of the database that satisfy the constraints, which we call repairs. For a boolean query Q, the problem CERTAINTY(Q) asks whether every such repair satisfies the query or not; the problem is known to be always in coNP for conjunctive queries. However, there are queries for which it can be solved in polynomial time. It has been conjectured that there exists a dichotomy on the complexity of CERTAINTY(Q) for conjunctive queries: it is either in PTIME or coNP-complete. In this paper, we prove that the conjecture is indeed true for the case of conjunctive queries without self-joins, where each atom has as a key either a single attribute (simple key) or all attributes of the atom.

Categories and Subject Descriptors

H.2.4 [**Database Management**]: Relational Databases

General Terms

Algorithms, Theory

Keywords

Repairs, Consistent query answering, Dichotomy

1. INTRODUCTION

Uncertainty in databases arises in several applications and domains (*e.g.* data integration, data exchange). An *uncertain* (or *inconsistent*) database is one that violates the integrity constraints of the database schema. In this work, we examine uncertainty under the framework of *consistent query answering*, established in [2].

In this framework, the presence of uncertainty generates many possible worlds, referred usually as *repairs*. For an inconsistent database I, a repair is a subset of I that minimally differs from I and also satisfies the integrity constraints. For a given query Q on database I, the set of *certain answers* contains all the answers that occur in every $Q(r)$, where r is a repair of I. The main research problem here is when the certain answers can be computed efficiently.

In this paper, we will restrict the problem such that the integrity constraints are only *key constraints*, and moreover, the queries are *boolean conjunctive queries*. In this case, a repair r of an inconsistent database I selects from each relation a maximal number of tuples such that no two tuples are key-equal. We further say that a boolean conjunctive query Q is *certain* if it evaluates to true for every such repair r. The decision problem CERTAINTY(Q) is now defined as follows: given an inconsistent database I, does $Q(r)$ evaluate to true for every repair r of I?

For this setting, it is known that CERTAINTY(Q) is always in coNP [3]. However, depending on the key constraints and the structure of the query Q, the complexity of the problem may vary. For example, for the query $Q_1 = R(\underline{x}, y), S(\underline{y}, z)$, CERTAINTY($Q_1$) is not only in P but, since one can show that CERTAINTY(Q_1) can be expressed as a first-order query over I [6], it is in AC^0. On the other hand, for $Q_2 = R(\underline{x}, y), S(\underline{z}, y)$, it has been proved in [6] that CERTAINTY(Q_2) is coNP-complete. Finally, for $Q_3 = R(\underline{x}, y), S(\underline{y}, x)$, one can show [14] that consistent query answering is in P, but the problem does not admit a first-order rewriting.

From the above examples, one can see that the complexity landscape is fairly intricate, even for the class of conjunctive queries. Although there has been progress in understanding the complexity for several classes of queries, the problem of deciding the complexity of CERTAINTY(Q) remains open. In fact, a long-standing conjecture claims the following dichotomy.

CONJECTURE 1.1. *Given a boolean conjunctive query Q, CERTAINTY(Q) is either in PTIME or is coNP-complete.*

The progress that has been made towards proving this conjecture has been limited. In particular, Kolaitis and Pema [8] have proved a dichotomy into PTIME and coNP-complete for the case where Q contains only two atoms and no self-joins (*i.e.* every relation name appears once). Wijsen [13] has given a necessary and sufficient condition for first-order rewriting for acyclic conjunctive queries without self-joins, and in a recent paper [15] further classifies several acyclic queries into PTIME and coNP-complete.

In this work, we significantly progress the status of the conjecture, by settling the dichotomy for a large class of

10.5441/002/icdt.2014.19

queries: boolean conjunctive queries w/o self-joins, where each atom has as primary key either a single attribute or all the attributes. Observe that this class contains all queries where atoms have arity at most 2; in particular, it also contains all three of the queries Q_1, Q_2, Q_3 previously discussed. Our results apply to a more general setting where one might have the external knowledge that some relations are *consistent* and others may be *inconsistent*. In contrast to previous approaches, our paper introduces consistent relations since in non-acyclic queries, certain patterns in the structure of the query cause a relation to behave as a consistent relation when checking for certainty. In particular, consider a query Q containing two atoms $R_1(\underline{x}, y), R_2(\underline{x}, y)$. If an instance contains the tuples $R_1(\underline{a}, b_1), R_2(\underline{a}, b_2)$ such that $b_1 \neq b_2$, we can remove the key-groups $R_1(\underline{a}, -), R_2(\underline{a}, -)$ without loss of generality in order to check for certainty[1]. Thus, the conjunction of R_1, R_2 behaves as a single consistent relation $R(\underline{x}, y)$. Our main result is

THEOREM 1.2. *For every boolean conjunctive query Q without self-joins consisting only of binary relations where exactly one attribute is the key, there exists a dichotomy of* CERTAINTY*(Q) into PTIME and coNP-complete.*

From here we derive:

COROLLARY 1.3. *For every boolean conjunctive query Q with relations of arbitrary arity, where either exactly one attribute is a key, or the key consists of all attributes, there exists a dichotomy of* CERTAINTY*(Q) into PTIME and coNP-complete.*

We prove Corollary 1.3 in the full version of this paper [9]; this paper consists of the proof of Theorem 1.2. The classification into PTIME and coNP-complete is based on analyzing the structure of a specific graph representation of the query along with the key constraints. The *query graph*, which we denote $G[Q]$, is a directed graph with vertices the variables in Q, and a directed edge (x, y) for every relation $R(\underline{x}, y)$.

Given the graph $G[Q]$, we give a necessary and sufficient condition for CERTAINTY(Q) to be computable in polynomial time. Consider two edges $e_R = (u_R, v_R), e_S = (u_S, v_S)$ in $G[Q]$ that correspond to two inconsistent relations R and S respectively. We say that e_R, e_S are *source-equivalent* if u_R, u_S belong to the same strongly connected component of $G[Q]$. We also say that e_R, e_S are *coupled* if (a) there exists an undirected path P_R from v_R to u_S such that no node in P_R is reachable from u_R through a directed path in $G - \{e_R\}$ and (b) there exists an undirected path P_S from v_S to u_R where no node in P_S is reachable from u_S through a directed path in $G - \{e_S\}$. Then:

THEOREM 1.4. *(1)* CERTAINTY*(Q) is coNP-complete if $G[Q]$ contains a pair of inconsistent edges that are coupled and not source-equivalent. Otherwise,* CERTAINTY*(Q) is in PTIME. (2) The problem: given a query Q decide whether* CERTAINTY*(Q) is coNP-complete or in PTIME is NLOGSPACE-complete.*

The following example illustrates the main theorem.

[1]Indeed, if we want to find a repair r that does not satisfy Q, we can always pick these two tuples to make sure that the value a will never contribute to an answer.

EXAMPLE 1.5. *Consider the following two queries:*

$$K_1 = R(\underline{x}, y), S(\underline{z}, w), T^c(\underline{y}, w)$$
$$K_2 = R(\underline{x}, y), S(\underline{z}, w), T^c(\underline{y}, w), U^c(\underline{x}, z)$$

Observe that the only difference between K_1, K_2 is the consistent relation U^c. Moreover, the edges e_R, e_S are not source-equivalent in both cases. In $G[K_1]$, the edges e_R, e_S are also coupled. Indeed, consider the path P_R that consists of the edges e_T, e_S and connects y with z. The nodes y, w, z of P_R are not reachable from x in the graphs $G[K_1] - \{e_R\}$. Similarly, the path P_S that consists of the edges e_T, e_R connects w with x and is not reached by any directed path starting from z in $G[K_1] - \{e_S\}$. Thus, CERTAINTY*(K_1) is coNP-complete.*

In contrast, the path P_R is reachable from x in $G[K_2]$: consider the path that consists of e_U. Since no other path connects e_R, e_S in $G[K_2]$, the edges e_R, e_S are not coupled. Thus, CERTAINTY*(K_2) is in PTIME.*

Note that if two edges e_R, e_S belong to two distinct weakly connected components, then they are trivially not coupled, which implies that Q is coNP-complete iff one of its weakly connected components is coNP complete.

In order to show Theorem 1.4, we develop new techniques for efficient computation of CERTAINTY(Q), as well as techniques for proving hardness. We start by introducing in Section 2 and Section 3 the basic notions and definitions. In Section 4, we present the case where $G[Q]$ is a strongly connected graph (*i.e.* there is a directed path from any node to any other node) and show that CERTAINTY(Q) is in PTIME. The algorithm for computing CERTAINTY(Q) in this case is based on a novel use of *or-sets* to represent efficiently answers to repairs. The polynomial time algorithm for CERTAINTY(Q) when $G[Q]$ satisfies the condition of Theorem 1.4 is presented in Section 3 and is based on a recursive decomposition of $G[Q]$. Finally, the hardness results are presented in Section 6, where we show that we can reduce the NP-hard problem MONOTONE-3SAT to any graph $G[Q]$ that does not satisfy the condition of Theorem 1.4.

2. PRELIMINARIES

A database schema is a finite set of relation names. Each relation R has a set of attributes $attr(R) = \{A_1, \ldots, A_k\}$, and a key, which is a subset of $attr(R)$. We typically write $R(\underline{x_1, \ldots, x_m}, y_1, \ldots, y_\ell)$ to denote that the attributes on positions $1, \ldots, m$ are the primary key. Each relation is of one of two types: consistent, or inconsistent. Sometimes we denote R^c or R^i to indicate that the type of the relation is consistent or inconsistent.

An instance I consists of a finite relation R^I for each relation name R, such that, if R is of consistent type, then R^I satisfies its key constraint. In other words, in an instance I we allow relations R^i to violate the key constraints but always require the relations R^c to satisfy the key constraints. Notice that, if the key of R consists of all attributes, then R^I always satisfies the key constraints, so we may assume w.l.o.g. that R is of consistent-type.

We denote a tuple by $R(\underline{a_1, \ldots, a_m}, b_1, \ldots, b_\ell)$. We define a *key-group* to be all the tuples of a relation with the same key, in notation $R(\underline{a_1, \ldots, a_m}, -)$.

Definition 2.1 (Repair). *An instance r is a* repair *for I if (a) r satisfies all key constraints and (b) r is a maximal subset of I that satisfies property (a).*

In this work, we study how to answer conjunctive queries on inconsistent instances:

Definition 2.2 (Consistent Query Answer). *Given an instance I, and a conjunctive query Q, we say that a tuple t is a* consistent answer *for Q if for every repair $r \subseteq I$, $t \in Q(r)$. If Q is a Boolean query, we say that Q is certain for I, denoted $I \models Q$, if for every repair r, $Q(r)$ is true.*

If Q is Boolean query, Certainty*(Q) denote the following decision problem: given an instance I, check if $I \models Q$.*

2.1 Frugal Repairs

Let Q be a Boolean conjunctive query Q. Denote Q^f the *full query* associated to Q, where all variables become head variables; therefore, for any repair r, $Q(r)$ is true iff $Q^f(r) \neq \emptyset$.

Definition 2.3 (Frugal Repair). *A repair r of I is* frugal *for Q if there exists no repair r' of I such that $Q^f(r') \subsetneq Q^f(r)$.*

Example 2.4. *Let $Q = R(\underline{x}, y), S(\underline{x}, y)$. In this case, the full query is $Q^f(x, y) = R(\underline{x}, y), S(\underline{x}, y)$. Also, consider the instance*

$$I = \{R(\underline{a_1}, b_1), R(\underline{a_1}, b_2), R(\underline{a_2}, b_3), S(\underline{a_1}, b_1), S(\underline{a_2}, b_3), R(\underline{a_3}, b_4), R(\underline{a_3}, b_5), S(\underline{a_3}, b_4), S(\underline{a_3}, b_5)\}$$

with the following repairs:

$$r_1 = \{R(\underline{a_1}, b_1), R(\underline{a_2}, b_3), S(\underline{a_1}, b_1), S(\underline{a_2}, b_3), R(\underline{a_3}, b_4), S(\underline{a_3}, b_4), S(\underline{a_3}, b_5)\}$$
$$r_2 = \{R(\underline{a_1}, b_2), R(\underline{a_2}, b_3), S(\underline{a_1}, b_1), S(\underline{a_2}, b_3), R(\underline{a_3}, b_4), S(\underline{a_3}, b_4), S(\underline{a_3}, b_5)\}$$
$$r_3 = \{R(\underline{a_1}, b_2), R(\underline{a_2}, b_3), S(\underline{a_1}, b_1), S(\underline{a_2}, b_3), R(\underline{a_3}, b_5), S(\underline{a_3}, b_4), S(\underline{a_3}, b_5)\}$$

Then, the answer sets are $Q^f(r_1) = \{(a_1, b_1), (a_2, b_3), (a_3, b_4)\}$, $Q^f(r_2) = \{(a_2, b_3), (a_3, b_4)\}$ and $Q^f(r_2) = \{(a_2, b_3), (a_3, b_5)\}$ respectively. Since $Q^f(r_2) \subsetneq Q^f(r_1)$, the repair r_1 is not frugal. On the other hand, both r_2 and r_3 are frugal.

Proposition 2.5. *$I \models Q$ if and only if every frugal repair of I for Q satisfies Q.*

Proof. One direction is straightforward: if some frugal repair does not satisfy Q, then Q is not certain for I. For the other direction, assume that Q is not certain for I. Then there exists a repair r s.t. $Q(r)$ is false, hence $Q^f(r) = \emptyset$: therefore r is a frugal repair, proving the claim. $\square$

The proposition also implies that we lose no generality if we study only frugal repairs in certain query answering. To check $I \models Q$ it suffices to check whether $Q^f(r) \neq \emptyset$ for every frugal repair. In some cases, it is even possible to compute $Q^f(r)$ by using a certain representation, as discussed next.

2.2 Representability

In general, the number of frugal repairs is exponential in the size of I. We describe here a compact representation method for the set of all answers $Q^f(r)$, where r ranges over all frugal repairs. We use the notation of or-sets adapted from [10]. An *or-set* is a set whose meaning is that one of its elements is selected nondeterministically. Following [10] we use angle brackets to denote or-sets. For example, $\langle 1, 2, 3\rangle$ denotes the or-set that is either 1 or 2 or 3; similarly $\langle \{1\}, \{1, 3\}\rangle$ means either the set $\{1\}$ or $\{1, 3\}$.

Let $\mathcal{F}_Q(I) = \langle r_1, r_2, \ldots\rangle$ be the or-set of all frugal repairs of I for Q, and let

$$\mathcal{M}_Q(I) = \langle Q^f(r) \mid r \in \mathcal{F}_Q(I)\rangle$$

be the or-set of all answers of Q^f on all frugal repairs. Notice that the type of $\mathcal{M}_Q(I)$ is $\langle\{T\}\rangle$, where $T = \times_{i=1}^k T_i$ is a product of atomic types. For a simple illustration, in Example 2.4, $\mathcal{M}_Q(I) = \langle\{(a_2, b_3), (a_3, b_4)\}, \{(a_2, b_3), (a_3, b_5)\}\rangle$, because r_2, r_3 are the only frugal repairs.

Give a type T, define the function $\alpha : \{\langle T\rangle\} \to \langle\{T\}\rangle$ [10]: $\alpha(\{A_1, \ldots, A_m\}) = \langle\{x_1, \ldots, x_m\} | x_1 \in A_1, \ldots, x_m \in A_m\rangle$. For example, $\alpha(\{\langle 1, 2\rangle, \langle 3, 4\rangle\}) = \langle\{1, 3\}, \{1, 4\}, \{2, 3\}, \{2, 4\}\rangle$ and $\alpha(\{\langle 1\rangle, \langle 1, 2, 3\rangle\}) = \langle\{1\}, \{1, 2\}, \{1, 3\}\rangle$.

Definition 2.6. *Let $T = \times_{i=1}^k T_i$. An or-set-of-sets S (of type $\langle\{T\}\rangle$) is* representable *if there exists a set-of-or-sets S_0 (of type $\{\langle T\rangle\}$) such that (a) $\alpha(S_0) = S$ and (b) for any distinct or-sets $A, B \in S_0$, the tuples in A and B use distinct constants in all coordinates: $\Pi_i(A) \cap \Pi_i(B) = \emptyset$, $\forall i = 1, k$.*

As an example, consider the or-sets

$$S = \langle\{(a_1, b_1), (a_2, b_3)\}, \{(a_1, b_2), (a_2, b_3)\}\rangle$$
$$S' = \langle\{(a_1, b_1), (a_2, b_3)\}, \{(a_1, b_2), (a_2, b_2)\}\rangle$$

S is representable, since we can find a compression $S_0 = \{\langle(a_1, b_1), (a_1, b_2)\rangle, \langle(a_2, b_3)\rangle\}$. Notice that a_1, b_1, b_2 appear only in the first or-set of S_0, whereas a_2, b_3 only in the second. On the other hand, it is easy to see that S' is not representable. We prove:

Proposition 2.7. *Let S or-set of sets of type $\langle\{\times_{i=1}^k T_i\}\rangle$, and suppose that its active domain has size n. If S is representable $S = \alpha(S_0)$, then its compression S_0 has size $O(n^k)$.*

Proof. If $S_0 = \{A_1, A_2, \ldots\}$, then every k-tuple consisting of constants from the active domain occurs in at most one or-set, thus the total size of S_0 is $O(n^k)$. $\square$

If $\mathcal{M}_Q(I)$ is representable, then we denote $A_Q(I)$ its compression; its size is at most polynomially large in I. In general, $\mathcal{M}_Q(I)$ may not be representable.

By the definition of frugality, if $s_1, s_2 \in \mathcal{M}_Q(I)$ then neither $s_1 \subsetneq s_2$ nor $s_2 \subsetneq s_1$ holds. This implies that, for any instance I, there are two cases. Either (1) $I \not\models Q$; in that case $\mathcal{M}_Q(I) = \langle\{\}\rangle$ is trivially representable as $A_Q(I) = \{\}$; or, (2) $I \models Q$, and in that case $\mathcal{M}_Q(I) = \langle A_1, A_2, \ldots\rangle$, where $A_i \neq \{\}$ for all i, may be exponentially large and not necessarily representable. For a simple illustration, in Example 2.4, $\mathcal{M}_Q(I)$ is representable, and its compression is $A_Q(I) = \{\langle(a_2, b_3)\rangle, \langle(a_3, b_4), (a_3, b_5)\rangle\}$.

If $A_Q(I)$ exists for every instance I and can be computed in polynomial time in the size of I, then Certainty(Q) is PTIME: to check $I \models Q$, simply compute $A_Q(I)$ and check $\neq \{\}$. The converse is not true, however: for example, consider the query $H = R(\underline{x}, y), S(\underline{y}, z)$, for which Certainty$(H)$ is in PTIME. However, for the instance $I' = \{R(\underline{a}, b), S(\underline{b}, c_1), S(\underline{b}, c_2)\}$, $\mathcal{M}_H(I') = \langle\{(a, b, c_1)\}, \{(a, b, c_2)\}\rangle$ is not representable.

2.3 Purified Instances

Let Q be a any boolean conjunctive query. An instance I is called *globally consistent* [1, pp.128], or *purified* [15], if for every relation R, $\Pi_{attr(R)}(Q^f(I)) = R^I$, where $\Pi_{attr(R)}$ denotes the projection on the attributes of relation R. In other words, no tuple in I is "dangling".

In the rest of the paper we will assume that the instance I is purified. This is without loss of generality, because if I is an arbitrary instance, then we can define a new instance $I^p \subseteq I$ such that $\mathcal{M}_Q(I) = \mathcal{M}_Q(I^p)$, and thus $I \models Q$ if and only if $I^p \models Q$.

LEMMA 2.8. *Given a query Q and an instance I, there exists a purified instance $I^p \subseteq I$ such that $\mathcal{M}_Q(I) = \mathcal{M}_Q(I^p)$.*

2.4 The Query Graph

In the rest of the paper we will restrict the discussion to the setting of Theorem 1.2, and consider only Boolean queries w/o self-joins consisting only of binary relations where exactly one attribute is the key; in [9] we prove Corollary 1.3, thus extending the dichotomy to more general queries.

Given a query Q, the *query graph* $G[Q]$ is a directed graph where the vertex set $V(G)$ consists of set of variables in Q, and edge set $E(G)$ contains for atom $R(\underline{u}, v)$ in Q an edge $e_R = (u, v)$ in $G[Q]$. Since Q has no self-joins each relation R defines a unique edge e_R, and we denote u_R and v_R its starting and ending node respectively. We say that the edge is consistent (inconsistent) if the type of R is consistent (inconsistent), and denote $E^i(G)$ ($E^c(G)$) the set of all consistent (inconsistent) edges. Thus $E(G) = E^i(G) \cup E^c(G)$.

A *directed path* P is an alternating sequence of vertices and edges $v_0, e_1, v_1, \ldots, e_\ell, v_\ell$ where $e_i = (v_{i-1}, v_i)$ for $i = 1, \ldots, \ell$ and $\ell \geqslant 0$. We write $P : x \to y$ for a directed path P where $v_0 = x$ to $v_\ell = y$, and every edge e_i is consistent; we write $P : x \rightsquigarrow y$ for any directed path P where $v_0 = x$ and $v_\ell = y$ that has any type of edges. An *undirected path* P is an alternating sequence of vertices and edges $v_0, e_1, v_1, \ldots, e_\ell, v_\ell$ where either $e_i = (v_{i-1}, v_i)$ or $e_i = (v_i, v_{i-1})$ for $i = 1, \ldots, \ell$ and $\ell \geqslant 0$; we write $P : x \leftrightarrow y$ for an undirected path where $v_0 = x$ and $v_\ell = y$ (that may also have any types of edges). A path P may contain a single vertex and no edges (when $\ell = 0$), in which case we can write $P : x \to x$. If $N \subseteq V(G)$, then $P \cap N$ denotes the set of vertices in P that occur in N. The notation $x \to y$ (or $x \rightsquigarrow y$, or $x \leftrightarrow y$) means "there exists a path $P : x \to y$" (or $P : x \rightsquigarrow y$, or $P : x \leftrightarrow y$).

Finally, since Q uniquely defines $G[Q]$ and vice versa, we will often use G to denote the the query Q (for example, we may say $G(r)$ instead of the boolean value $Q(r)$, for some repair r).

EXAMPLE 2.9. *Consider the following query:*

$$H = R_1(\underline{x}, y), R_2^c(\underline{y}, z), R_3(\underline{z}, x), V_1^c(\underline{u}, y), V_2^c(\underline{x}, v),$$
$$V_3^c(\underline{z}, v), S(\underline{u}, v), T(\underline{v}, w), U^c(\underline{u}, w)$$

The graph $G[H]$ is depicted in Figure 1. The curly edges denote inconsistent edges $E^i = \{R_1, R_3, S, T\}$, whereas the straight edges denote consistent ones. We also have $u \rightsquigarrow x$ (but not $u \to x$, since the only path from u to x contains inconsistent edges). Moreover, $y \to v$, since there is a directed path that goes from y to v through R_2, V_3. Finally, notice that, although $v \not\rightsquigarrow y$, $v \leftrightarrow y$.

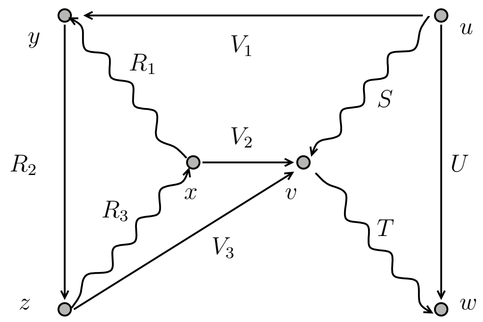

Figure 1: The query graph $G[H]$. The curly edges denote inconsistent relations, whereas the straight edges consistent relations.

2.5 The Instance Graph

Let Q be a Boolean conjunctive query without self-joins over binary relations with single-attribute keys. Let I be an instance for Q. We will assume w.l.o.g. that any two attributes that are not joined by Q have disjoint domains: otherwise, we simply rename the constants in one attribute. For example, if $Q = R(x, y), S(y, z), T(z, x)$ then we will assume that $\Pi_1(R^I) \cap \Pi_1(S^I) = \emptyset$, etc.

The *instance graph* is the following directed graph $F_Q(I)$. The nodes consists of all the constants occurring in I, and there is an edge (a, b) for every tuple $R^I(\underline{a}, b)$ in I. The size of the instance graph $F_Q(I)$ is the same as the size of the instance I.

3. THE DICHOTOMY THEOREM

We present here formally our dichotomy theorem, and start by introducing some definitions and notations. Let $u \in V(G)$ and $e_R \in E(G)$. Then,

$$u^{\oplus} = \{v \in V(G) \mid u \to v \text{ in } G\}$$
$$u^{+} = \{v \in V(G) \mid u \rightsquigarrow v \text{ in } G\}$$
$$u^{+,R} = \{v \in V(G) \mid u \rightsquigarrow v \text{ in } G - \{e_R\}\}$$

EXAMPLE 3.1. *Consider the graph $G[H]$ from Figure 1, which will be our running example. Then:*

$$x^{\oplus} = \{x, v\} \qquad x^{+,R_1} = \{x, v, w\} \qquad x^{+} = \{x, v, w, y, z\}$$

PROPOSITION 3.2. *If $R \in E^i$, $u_R^{\oplus} \subseteq u_R^{+,R} \subseteq u_R^{+}$.*

PROOF. Let $v \in u_R^{\oplus}$. Then, there exists a path $P : u_R \to v$ in G. Since P is consistent, it cannot contain the inconsistent edge e_R, and thus P exists in $G - \{e_R\}$ as well. Consequently, $v \in u_R^{+,R}$. The other inclusion is straightforward. □

Define the binary relation $R \lesssim S$ if $u_S \in u_R^{+}$. The relation $\lesssim$ is a *preorder* the set of edges, since it is reflexive and transitive. If $R \lesssim S$ and $S \lesssim R$ then we say that R, S are *source-equivalent* and denote $R \sim S$. Notice that $R \sim S$ iff their source nodes u_R, u_S belong in the same strongly connected component (SCC) of G; in particular, if R, S have the same source node, $u_R = u_S$, then $R \sim S$.

For $R \in E^i$, we define the following sets of *coupled* edges:

$$coupled^{\oplus}(R) = [R] \cup \{S \in E^i \mid \exists P : v_R \leftrightarrow u_S, P \cap u_R^{\oplus} = \emptyset\}$$
$$coupled^{+}(R) = [R] \cup \{S \in E^i \mid \exists P : v_R \leftrightarrow u_S, P \cap u_R^{+,R} = \emptyset\}$$

By definition, every edge S that is source-equivalent to R is coupled with R. In addition, $coupled^{\oplus}(R)$ ($coupled^{+}(R)$), includes all inconsistent edges S whose source node u_S is in the same weakly connected component as v_R, in the graph $G - u_R^{\oplus}$ ($G - u_R^{+,R}$ respectively). The notion of $coupled^{\oplus}$ is not necessary to express the dichotomy theorem, but it will be heavily used in the algorithm of Section 5.

EXAMPLE 3.3. *Let us compute the coupled edges in our running example, where $E^i = \{R_1, R_3, S, T\}$. We start by computing the node-closures of all the four source nodes:*

$$x^{\oplus} = \{x, v\},\ x^{+,R_1} = \{x, v, w\},\ z^{\oplus} = \{z, v\},\ z^{+,R_3} = \{z, v, w\},$$
$$u^{\oplus} = \{u, y, w\},\ u^{+,S} = \{u, y, w, x, v, w\},\ v^{\oplus} = \{v\},\ v^{+,T} = \{v\}$$

Next, we compute $coupled^{+}(e)$ for every inconsistent edge e. For example, the set $coupled^{+}(R_1)$ includes R_1 and R_3, because $R_1 \sim R_3$. In addition, after we remove $x^{+,R_1} = \{x, v, w\}$ from the graph, the destination node y of R_1 is still weakly connected to the source node u of S, thus $coupled^{+}(R_1)$ contains S; but y is no longer connected to the source node v of T, therefore $coupled^{+}(R_1)$ does not contain T. By similar reasoning:

$$\begin{aligned} coupled^{\oplus}(R_1) &= \{R_1, R_3, S\} & coupled^{+}(R_1) &= \{R_1, R_3, S\} \\ coupled^{\oplus}(R_3) &= \{R_1, R_3, S, T\} & coupled^{+}(R_3) &= \{R_3\} \\ coupled^{\oplus}(S) &= \{S\} & coupled^{+}(S) &= \{S\} \\ coupled^{\oplus}(T) &= \{R_1, R_3, S, T\} & coupled^{+}(T) &= \{R_1, R_3, S, T\} \end{aligned}$$

Proposition 3.2 implies:

COROLLARY 3.4. *If $R \in E^i$, $coupled^{\oplus}(R) \supseteq coupled^{+}(R)$.*

DEFINITION 3.5 (SPLITTABLE). *Two edges $R, S \in E^i$ are* coupled *if $R \in coupled^{+}(S)$ and $S \in coupled^{+}(R)$.*

The graph G is called unsplittable *if there exists two coupled edges R, S s.t. $R \not\sim S$. Otherwise, the graph is called* splittable.

The graph $G[H]$ from our running example is splittable, because the only pair of coupled edges are R_1, R_3, which are also source-equivalent. Indeed, any other pair is not coupled: R_1, S are not coupled because $R_1 \notin coupled^{+}(S)$; R_1, T are not coupled because $T \notin coupled^{+}(R_1)$; etc.

We can now state our dichotomy theorem, which we will prove in the rest of the paper.

THEOREM 3.6 (DICHOTOMY THEOREM). *(1) If $G[Q]$ is splittable, then* CERTAINTY*(Q) is in PTIME. (2) If $G[Q]$ is unsplittable, then* CERTAINTY*(Q) is coNP-complete.*

We end this section with a few observations. First, if Q consists of several weak connected components $Q_1, Q_2, \ldots$, in other words, Q_i, Q_j do not share any variables for all $i \neq j$, then Q is unsplittable iff some Q_i is unsplittable: this follows from the fact that $coupled^{+}(R)$ is included in the weakly connected component Q_i that contains R. In this case, Theorem 3.6 implies that CERTAINTY(Q) is coNP-complete iff CERTAINTY(Q_i) is coNP-complete for some i.

Second, if Q is strongly connected, then it is, by definition, splittable: in this case Theorem 3.6 says that CERTAINTY(Q) is in PTIME. In fact, the first step of our proof is to show that every strongly connected query is in PTIME.

$R(\underline{x}, y)$	$S(\underline{y}, z)$	$T(\underline{z}, x)$
(a_1, b_1)	(b_1, c_1)	(c_1, a_1)
(a_1, b_2)	(b_2, c_1)	
(a_2, b_2)	(b_2, c_2)	(c_2, a_2)
(a_3, b_3)	(b_3, c_3)	(c_3, a_3)
(a_3, b_4)	(b_4, c_4)	(c_4, a_3)
(a_4, b_4)	(b_4, c_3)	(c_3, a_4)

Figure 2: An inconsistent purified instance I for C_3.

Finally, we note that the property of being splittable or unsplittable may change arbitrarily, as we add more edges to the graph. For example, consider these three queries: $Q_1 = R(\underline{x}, y)$, $Q_2 = R(\underline{x}, y), S(\underline{z}, y)$, $Q_3 = R(\underline{x}, y), S(\underline{z}, y), T(\underline{z}, y)$, where all three relations R, S, T are inconsistent. Then Q_1, Q_3 are splittable, while Q_2 is unsplittable, and therefore, their complexities are PTIME, coNP-hard, PTIME. Indeed, in Q_2 we have $coupled^{+}(R) = coupled^{+}(S) = \{R, S\}$, therefore R, S are coupled and in-equivalent $R \not\sim S$, thus, Q_2 is unsplittable. On the other hand, in Q_3 we have[2] $coupled^{+}(S) = \{S, T\}$, $coupled^{+}(T) = \{S, T\}$, and therefore R, S are no longer coupled, nor are R, T: Q_3 is splittable.

4. STRONGLY CONNECTED GRAPHS

If $G[Q]$ is a strongly connected graph (SCG), then it is, by definition, splittable. Our first step is to prove Part (1) of Theorem 3.6 in the special case when $G[Q]$ is a strongly connected, by showing that CERTAINTY(Q) is in PTIME. We actually show an even stronger statement.

THEOREM 4.1. *If $G[Q]$ is strongly connected, $\mathcal{M}_Q(I)$ is representable and its compression $A_Q(I)$ can be computed in polynomial time in the size of I.*

As we discussed in Section 2, CERTAINTY(Q) is false if and only if $A_Q(I) = \{\}$; hence, as a corollary we obtain:

COROLLARY 4.2. *If $G[Q]$ is a strongly connected graph,* CERTAINTY*(Q) is in PTIME.*

We start in Subsection 4.1 by proving Theorem 4.1 in the special case when $G[Q]$ is a directed cycle; we prove the general case in Subsection 4.2.

4.1 A PTIME Algorithm for Cycles

For any $k \geqslant 2$, the cycle query C_k is defined as:

$$C_k = R_1(\underline{x_1}, x_2), R_2(\underline{x_2}, x_3), \ldots, R_k(\underline{x_k}, x_1)$$

Wijsen [15] describes a PTIME algorithm for computing CERTAINTY(C_2). We describe here a PTIME algorithm for computing $A_{C_k}(I)$ (and thus for computing CERTAINTY(C_k) for arbitrary $k \geqslant 2$ as well), called FRUGALC.

LEMMA 4.3. *Let I be a purified instance relative to C_k. Then, the instance graph $F_{C_k}(I)$ is a collection of disjoint SCCs such that every edge has both endpoints in the same SCC.*

[2]The difference between Q_2 and Q_3 is that in Q_2 we have $z^{+,S} = \{z\}$, while in Q_3 we have $z^{+,S} = \{x, y, z\}$.

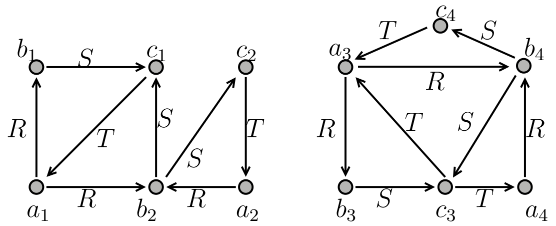

Figure 3: The graph $F_{C_3}(I)$ for the instance in Figure 2 has two SCC's, F_1 and F_2.

Proof. Let (u, v) be a directed edge in the graph. Since I is purified, (u, v) must belong in a cycle and thus there exists a directed path $v \to u$, implying that u, v are in the same SCC. $\Box$

Algorithm. Fix $k \geqslant 2$. The algorithm FrugalC takes as input a purified instance I and returns the compression $A_{C_k}(I)$ of $\mathcal{M}_{C_k}(I)$, in four steps:

1. Compute the SCCs of $F_{C_k}(I)$: $F_{C_k}(I) = F_1 \cup \ldots \cup F_m$, where each F_i is an SCC, and there are no edges between F_i, F_j for $i \neq j$.
2. Compute $S = \{i \mid F_i$ has no cycle of length $> k\}$.
3. For each $i \in S$, define the or-set: $A_i = \langle(a_1, \ldots, a_k) \mid a_1, \ldots, a_k$ cycle in $F_i\rangle$.
4. Return: $\{A_i \mid i \in S\}$.

Step 1 is clearly computable in PTIME. In Step 2, we remove all SCC's F_i that contain a cycle of length $> k$: to check that, enumerate over all simple paths of length $k+1$ in F_i (there are at most $O(n^{k+1})$), and for each path $u_0, u_1, u_2, \ldots, u_k$ check whether there exists a path from u_k to u_0 in $F_i - \{u_1, \ldots, u_{k-1}\}$. After Step 2, if $i \in S$, then every cycle in F_i has length k, and every edge is on a k-cycle (because I is purified). Step 3 constructs an or-set A_i consisting of all k-cycles of F_i (there are at most $O(n^k)$). The last step returns the set of all or-sets A_i: this is a correct representation (Definition 2.6) because no two or-sets A_i, A_j have any common constants (since they represent cycles from different SCC's). We will prove in the rest of the section that $A_{C_k}(I) = \{A_i \mid i \in S\}$, and therefore the algorithm correctly computes $A_{C_k}(I)$. Note that $I \models C_k$ iff $A_{C_k}(I) = \{\}$ iff $S = \varnothing$.

Example 4.4. *We illustrate the algorithm on the query $C_3 = R(\underline{x}, y), S(\underline{y}, z), T(\underline{z}, x)$. Consider the relations R, S, T of the instance I in Figure 2 and its graph $F_{C_3}(I) = F_1 \cup F_2$ shown in Figure 3. The SCC F_1 contains only cycles of length 3: (a_1, b_1, c_1), (a_1, b_2, c_1) and (a_2, b_2, c_2), whereas F_2 contains a cycle[3] of length 6: $(a_3, b_3, c_3, a_4, b_4, c_4)$. Therefore the algorithm returns a set consisting of a single or-set:*

$$A_{C_3}(I) = \{\langle(a_1, b_1, c_1), (a_1, b_2, c_1), (a_2, b_2, c_2)\rangle\}$$

It remains to show that the algorithm is correct, and this follows from two lemmas. Recall from Subsection 2.2 that $\mathcal{F}_{C_k}(I)$ denotes the or-set of frugal repairs of I for C_k. Assuming I is a purified instance, let $I = I_1 \cup I_2 \cup \ldots \cup I_m$, where each I_i corresponds to some SCC of $F_{C_k}(I)$.

[3]Notice that *every* edge in F_2 is on some cycle of length 3 (since I is purified), yet F_2 also contains a cycle of length 6.

Lemma 4.5. *For the frugal repairs of I, $\mathcal{F}_{C_k}(I) = \langle r_1 \cup \ldots \cup r_m | r_1 \in \mathcal{F}_{C_k}(I_1), \ldots, r_m \in \mathcal{F}_{C_k}(I_m)\rangle$*

In other words, the frugal repairs of I are obtained by choosing, independently, a frugal repair r_i for each SCC I_i, then taking their union.

Lemma 4.6. *Let I be a purified instance relative to C_k, such that $F_{C_k}(I)$ is strongly connected. Then:*

$$\mathcal{M}_{C_k}(I) = \begin{cases} \langle\{\}\rangle & \textit{if } I \textit{ has a cycle of length} > k, \\ \langle\{(a_1, \ldots, a_k)\} \mid a_1, \ldots, a_k \textit{ cycle in } F_{C_k}(I)\rangle & \textit{else} \end{cases}$$

The lemma says two things. On one hand, if I has a cycle of length $> k$, then $I \not\models C_k$. Consider the case when all cycles in I have length k. In general, if r is a minimal repair, then the full query $C_k^f(r)$ may return any nonempty set of k-cycles. The lemma states that if r is a frugal repair, then $C_k^f(r)$ returns *exactly one* k-cycle, and, moreover, that every k-cycle is returned on some frugal repair r.

We now apply the two lemmas to prove the correctness of the algorithm. Lemma 4.6 implies that, if I is strongly connected and has no cycle of length $> k$, $\mathcal{M}_{C_k}(I)$ is represented $A_{C_k}(I) = \{\langle(a_1, \ldots, a_k)|a_1, \ldots, a_k$ cycle in $F_{C_k}(I)\rangle\}$; and if I has a cycle of length $> k$ then $A_{C_k}(I) = \{\}$. Lemma 4.5 implies that, if I has m SCC's $I = I_1 \cup \ldots \cup I_m$, then $A_{C_k}(I) = A_{C_k}(I_1) \cup \ldots A_{C_k}(I_m)$. This completes the correctness proof of the the algorithm.

We conclude with an observation on FO-expressibility. Recall that [14] proves that the Certainty(C_2) is not first-order (FO)-expressible. The following proposition completes the complexity landscape for cycle queries.

Proposition 4.7. *For a cycle query C_k (where $k > 1$), Certainty(C_k) is FO-expressible if and only if C_k contains at most one inconsistent edge.*

4.2 A PTIME Algorithm for SCGs

We now present the general algorithm that computes the compression $A_Q(I)$ for any strongly connected query Q. The algorithm uses the following decomposition of the query graph $G[Q]$.

Let $G = G[Q]$ be a query graph and $G_0 \subseteq G$ be subgraph. A *chordal path* for G_0 is a simple, non-empty[4] path $P : u \rightsquigarrow v$ s.t. $G_0 \cap P = \{u, v\}$. If P consists of a single edge then we call it a *chord*. With some abuse, we apply the same terminology to queries: if the query Q can be written as Q_0, P, where Q_0 and P are sets of atoms s.t P is a simple path[5] from u to v, then we say that P is a chordal path for Q_0 if they share only the variables u, v.

Lemma 4.8 (Chordal Path Decomposition). *Let G be strongly connected. Then there exists a sequence $G_0 \subseteq \cdots \subseteq G_m = G$ of subgraphs of G such that*

1. *G_0 is a simple cycle*
2. *For every $i = 1, m$, $G_i = G_{i-1} \cup P_i$, where P_i is a chordal path of G_{i-1}.*

[4]Recall that, when $u = v$, then a simple, non-empty path from u to u is a cycle.

[5]Meaning that $P = R_1(\underline{u}, x_1), R_2(\underline{x_1}, x_2), \ldots, R_m(\underline{x_{m-1}}, v)$, all variables $u, x_1, \ldots, x_{m-1}$ are distinct, and all variables $x_1, \ldots, x_{m-1}, v$ are distinct.

EXAMPLE 4.9. *We will study the conjunctive query* $H_2 = R(\underline{x}, y), S(\underline{y}, z), T(\underline{z}, x), U(\underline{y}, t), V(\underline{t}, z)$. *The query admits the following decomposition:*

$$G_0 = G[Q_0] \qquad \text{where } Q_0 = R(\underline{x}, y), S(\underline{y}, z), T(\underline{z}, x)$$
$$G_1 = G_0 \cup P \qquad \text{where } P = U(\underline{y}, t), V(\underline{t}, z)$$

Our algorithm for computing CERTAINTY(Q) for an SCC Q uses a chordal path decomposition of Q and applies the following two procedures.

Procedure FrugalChord. Fix a query Q of the form $Q_0, R^c(\underline{u}, v)$, where $R^c(\underline{u}, v)$ is a chord for Q_0. The procedure FRUGALCHORD takes as input an instance I and the compact representation $A_{Q_0}(I)$, and returns the compact representation $A_Q(I)$. The procedure simply returns the set:

$$A_Q(I) = \{A \in A_{Q_0}(I) \mid \forall t \in A : (t[u], t[v]) \in R^c\} \quad (1)$$

In other words, the procedure computes a representation of Q on I by having access to a representation to Q_0 on I. Correctness follows from the following lemma, which is proven in [9].

LEMMA 4.10. *Let* $Q \equiv Q_0, R^c(\underline{u}, v)$ *such that* $R^c(\underline{u}, v)$ *is a chord of* Q_0. *If* $\mathcal{M}_{Q_0}(I)$ *is representable and its compression is* $A_{Q_0}(I)$*, then ,* $\mathcal{M}_{G_{i+1}}(I)$ *is also representable and its compression is given by Eq.(1).*

Procedure FrugalChordPath. Fix a query Q of the form Q_0, P, where P is a chordal path from u to v for Q_0. The procedure FRUGALCHORDPATH takes as input an instance I and the compact representation $A_{Q_0}(I)$, and returns the compact representation $A_Q(I)$, in six steps:

1. Assume $A_{Q_0}(I)$ has m or-sets, each with $n_1, \ldots, n_m$ elements:

   $$A_{Q_0}(I) = \{A_1, \ldots, A_m\},\ A_i = \langle t_{i1}, t_{i2}, \ldots, t_{in_i}\rangle \quad (2)$$

   Denote $n = \sum_i n_i$. Let a_i for $i = 1, m$ be m distinct constants, and let b_{ij} for $i = 1, m$, $j = 1, n_i$ be n distinct constants. Denote $\mathtt{tup}(b_{ij}) = t_{ij}$ the tuple encoded by b_{ij}.
2. Create four new relations:

   $$B^i = \{(\underline{a_i}, b_{ij}) \mid i = 1, m; j = 1, n_i\}$$
   $$B_1^c = \{(\underline{b_{ij}}, \pi_u(t_{ij})) \mid i = 1, m; j = 1, n_i\}$$
   $$B_2^c = \{(\underline{b_{ij}}, \pi_v(t_{ij})) \mid i = 1, m; j = 1, n_i\}$$
   $$B_0^c = \{(\underline{\pi_v(t_{ij})}, a_i) \mid i = 1, m; j = 1, n_i\}$$

   B^i is of inconsistent type (hence the superscript "i"), and B_1^c, B_2^c, B_0^c are of consistent type.
3. Assume the variables u, v are distinct, $u \neq v$: we discuss below the case $u = v$. Denote C_{k+3} and Q':

   $$C_{k+3} = B^i(\underline{a}, b), B_1^c(\underline{b}, u), R_1(\underline{u}, x_1), \ldots, R_k(\underline{x_{k-1}}, v), B_0^c(\underline{v}, a)$$
   $$Q' = C_{k+3}^f(a, b, u, x_1, \ldots, x_{k-1}, v), B_2^c(\underline{b}, v)$$

   where $R_1(\underline{u}, x_1), \ldots, R_k(\underline{x_{k-1}}, v)$ is the chordal path P, and a, b are new variables.
4. Use the algorithm FRUGALC to find the compact representation $A_{C_{k+3}}(I)$ for C_{k+3}.
5. Use the procedure FRUGALCHORD to find the compact representation of $A_{Q'}(I)$ for Q'.
6. Return the following set of or-sets:

   $$A_Q(I) = \{\langle(\mathtt{tup}(\pi_b(t)), \pi_{Vars(P)}(t)) \mid t \in A\rangle \mid A \in A_{Q'}(I)\} \quad (3)$$

We explain the algorithm next. In Step 1 we give fresh names to each or-set A_i in $A_{Q_0}(I)$, and to each tuple t_{ij} in each or-set in A_i: by Proposition 2.7, the number of constants needed is only polynomial in the size of the active domain of I. The crux of the algorithm is the table $B^i(\underline{a}, b)$ created in Step 2: its repairs correspond precisely to $\alpha(A_{Q_0}(I))$, up to renaming of constants. To see this notice that each repair of B^i has the form $\{(a_1, b_{1j_1}), \ldots, (a_m, b_{mj_m})\}$ for arbitrary choices $j_1 \in [n_1], \ldots, j_m \in [n_m]$. Therefore, the set of frugal repairs of B^i is $\alpha(S_0)$, where $S_0 = \{\langle(a_i, b_{ij}) | j = 1, n_i\rangle \mid i = 1, m\}$, which is precisely Eq.(2) up to renaming of the tuples by constants. The relation B_1^c decodes each constant b_{ij} by mapping it to the u-projection of t_{ij}; similarly for B_2^c. Clearly, both B_1^c, B_2^c are consistent, because every constant b_{ij} needs to be stored only once. The relation B_0^c is a reverse mapping, which associates to each value of v the name a_i of the unique or-set A_i that contains a tuple t_{ij} with that value in position v: the set A_i is uniquely defined because, by Definition 2.6, for any distinct sets A_{i_1}, A_{i_2} we have $\Pi_v(A_{i_1}) \cap \Pi_v(A_{i_2}) = \varnothing$.

Step 3 transforms Q into a cycle C_{k+3} plus a chord $B_2^c(\underline{b}, v)$, by simply replacing the entire subquery Q_0 with the single relation $B^i(\underline{a}, b)$ (which is correct, since $A_{Q_0}(I)$ is the same as the set of repairs of B^i) plus the decodings $B_1^c(\underline{b}, u)$, $B_2^c(\underline{b}, v)$: note that we only needed $B_0^c(\underline{v}, a)$ in order to close the cycle C_{k+3}. The next two steps compute the encodings $A_{C_{k+3}}(I)$ and $A_{Q'}(I)$ using the algorithm FRUGALC and FRUGALCHORD respectively. Finally, the last step converts back $A_{Q'}(I)$ into $A_Q(I)$ by expanding the constants b_{ij} into the tuples they encode, $t_{ij} = \mathtt{tup}(b_{ij})$. The algorithm has assumed $u \neq v$. If $u = v$ are the same variable, the C_{k+3} is no longer a cycle: in that case, we split u into two variables u, v and add two consistent relations $R^c(\underline{u}, v)$, $S^c(\underline{v}, u)$ to the query, and replace the last relation $R_k(\underline{x_{k-1}}, u)$ of P with $R_k(\underline{x_{k-1}}, v)$. The correctness of the algorithm follows from:

LEMMA 4.11. *Let Q be a query of the form Q_0, P where P is a chordal path from u to v for Q_0, and let I be an instance. Then, if $\mathcal{M}_{Q_0}(I)$ is representable and $A_{Q_0}(I)$ is its compact representation, then $\mathcal{M}_Q(I)$ is also representable and its compact representation is given by Eq.(3).*

Algorithm FrugalSCC. Let Q be a query that is strongly connected. The algorithm FRUGALSCC takes as input an instance I, and returns $A_Q(I)$, as follows. Let $Q_0, Q_1, \ldots, Q_m$ a chordal path decomposition for Q (Lemma 4.10). Start by computing $A_{Q_0}(I)$ using algorithm FRUGALC. Next, for each $i = 1, m$, use $A_{Q_{i-1}}(I)$ and the procedure FRUGALCHORDALPATH to compute $A_{Q_i}(I)$. Return $A_{Q_m}(I)$.

EXAMPLE 4.12. *Continuing Example 4.9, we will show how to compute $A_{H_2}(I_2)$ where I_2 is the instance shown in Figure 4. We write H_2 as $H_2 \equiv C_3, P$, where $C_3 = R(\underline{x}, y), S(\underline{y}, z), T(\underline{z}, x)$ and $P = U(\underline{y}, t), V(\underline{t}, z)$. We start by computing C_3 on I_2; one can check*[6] *that $A_{C_3}(I_2) =$*

[6]Every repair of I_2 contains exactly two cycles: (a_1, b_1, c_1) and one of (a_2, b_2, c_2) or (a_2, b_3, c_2).

$R(\underline{x}, y)$	$S(\underline{y}, z)$	$T(\underline{z}, x)$	$U(\underline{y}, t)$	$V(\underline{t}, z)$
(a_1, b_1)	(b_1, c_1)	(c_1, a_1)	(b_1, d)	(d, c_1)
(a_2, b_2)	(b_2, c_2)	(c_2, a_2)	(b_2, d)	(d, c_2)
(a_2, b_3)	(b_3, c_2)		(b_3, d)	

Figure 4: An inconsistent purified instance I_2 for H_2.

$B(\underline{a}, b)$	$B_1^c(\underline{b}, y)$	$B_2^c(\underline{b}, z)$	$B_0^c(\underline{z}, b)$
$(A_1, [a_1b_1c_1])$	$([a_1b_1c_1], b_1)$	$([a_1b_1c_1], c_1)$	(c_1, A_1)
$(A_2, [a_2b_2c_2])$	$([a_2b_2c_2], b_2)$	$([a_2b_2c_2], c_2)$	(c_2, A_2)
$(A_2, [a_2b_3c_2])$	$([a_2b_3c_2], b_3)$	$([a_2b_3c_2], c_2)$	

Figure 5: The resulting instance I' produced by the inductive step for H_2.

$\{A_1, A_2\}$ where $A_1 = \langle (a_1, b_1, c_1) \rangle$, $A_2 = \langle (a_2, b_2, c_2), (a_2, b_3, c_2) \rangle$. Encode the two sets with the new constants A_1, A_2, and encode the three tuples with three new constants $[a_1b_1c_1]$, $[a_2b_2c_2]$, $[a_2b_3c_2]$. The new relations we constructed $B^i(\underline{a}, b)$, $B_1^c(\underline{b}, y)$, $B_2^c(\underline{b}, z)$, $B_0^c(\underline{z}, b)$ are shown in Figure 5. Thus, we have to compute the following queries:

$$C_5 = B^i(\underline{a}, b), B_1^c(\underline{b}, y), U_1(\underline{y}, t), V(\underline{t}, z), B_0^c(\underline{z}, a)$$
$$Q' = C_5^f(a, b, y, t, z), B_2^c(\underline{b}, z)$$

on the instance I' in Figure 5. One can check that their answers are:

$$A_{C_5}(I') = \{\langle (A_1, [a_1b_1c_1], b_1, d, c_1), (A_2, [a_2b_2c_2], b_2, d, c_2), (A_2, [a_2b_3c_2], b_3, d, c_2) \rangle\}$$
$$A_{Q'}(I') = A_{C_5}(I')$$

Mapping this to the original query $H_2(x, y, z, t)$ by projecting out the A_i and merging the tuples, we obtain that

$$A_{H_2}(I_2) = \{\langle (a_1, b_1, c_1, d), (a_2, b_2, c_2, d), (a_2, b_3, c_2, d) \rangle\}$$

In particular, $I_2 \models H_2$, because $A_{H_2}(I_2)$ is nonempty.

5. THE PTIME ALGORITHM

In this section, we prove:

THEOREM 5.1. *If the graph $G[Q]$ is splittable, there exists a PTIME algorithm that solves* CERTAINTY*(Q).*

The polynomial time algorithm we present here is based on the fact that if $G[Q]$ is splittable, it has a very specific structure that allows us to break it into smaller pieces that we can solve independently; in other words, the problem is *self-reducible*. The graph object that allows this is called a *separator*, and we show in Subsection 5.4 that it always exists in $G[Q]$. Throughout this section, we will use the graph $G[H]$ of Figure 1 as a running example.

5.1 Separators

In this section, we define the notion of a *separator*, which is central to the construction of the polynomial time algorithm for deciding certainty on splittable graphs. We first need to set up some notation.

Recall that $\sim$ denotes a binary relation between edges $R, S \in E^i$: $R \sim S$ if R and S are source-equivalent. Consider the equivalence relation defined by $\sim$ on the set of inconsistent edges E^i, and denote $E^i/\sim$ the quotient set and $[R] \in E^i/\sim$ the equivalence class for an edge $R \in E^i$. For our example graph $G[H]$, we have $R_1 \sim R_3$ (because R_1, R_2, R_3 form a cycle), thus $[R_1] = \{R_1, R_3\}$. Also $S \lesssim [R_1]$, $S \lesssim T$, hence $E^i/\sim = \{[R_1], [S], [T]\}$.

For any $C \in E^i/\sim$, define

$$C^+ \stackrel{def}{\equiv} \bigcap_{R \in C} u_R^{+,R} \quad \text{and} \quad C^\oplus \stackrel{def}{\equiv} \bigcap_{R \in C} u_R^\oplus.$$

Similar to how we have defined *coupled*$^+(R)$, *coupled*$^\oplus(R)$ for any $R \in E^i$, we define *coupled*$^+(C)$, *coupled*$^\oplus(C)$ for $C \in E^i/\sim$:

$$\mathit{coupled}^+(C) \stackrel{def}{\equiv} \{C\} \cup \{C' \in E^i/\sim \mid \exists R \in C, S \in C' : \exists P : v_R \leftrightarrow u_S, P \cap C^+ = \varnothing\}$$
$$\mathit{coupled}^\oplus(C) \stackrel{def}{\equiv} \{C\} \cup \{C' \in E^i/\sim \mid \exists R \in C, S \in C' : \exists P : v_R \leftrightarrow u_S, P \cap C^\oplus = \varnothing\}$$

The definitions "lift" the notion of coupling from a single inconsistent edge to a set of inconsistent edges that forms an equivalence class. Continuing our example, we have:

$$\mathit{coupled}^+(\{R_1, R_3\}) = \{\{R_1, R_3\}, \{S\}\}$$
$$\mathit{coupled}^+(\{S\}) = \{\{S\}\}$$
$$\mathit{coupled}^+(\{T\}) = \{\{R_1, R_3\}, \{T\}, \{S\}\}$$

Moreover, for $G[H]$, the sets *coupled*$^+$, *coupled*$^\oplus$ coincide for every equivalence class.

For $C_1, C_2 \in E^i/\sim$, we define a binary relation $\leqslant^\oplus$: we write $C_1 \leqslant^\oplus C_2$ if there exists $S \in C_2$ such that $u_S \in C_1^\oplus$.

PROPOSITION 5.2. *$\leqslant^\oplus$ is antisymmetric and transitive.*

We can now define $C_1 <^\oplus C_2$ to be such that $C_1 \leqslant^\oplus C_2$ and $C_1 \neq C_2$. Then, following from Proposition 5.2, $<^\oplus$ is a *strict partial order*. We will be particularly interested in the maximal elements of this order, which we call *sinks*.

DEFINITION 5.3 (SINK). *$C \in E^i/\sim$ is a* sink *if it is a maximal element of $<^\oplus$.*

EXAMPLE 5.4. *Since $(u_{R_3} =) z \in u^\oplus (= u_S^\oplus)$, we have $\{S\} <^\oplus \{R_1, R_3\}$. Also, since $v \in u_{R_1}^\oplus \cap u_{R_3}^\oplus$, $\{R_1, R_3\} <^\oplus \{T\}$. By the transitivity of $<^\oplus$, we also obtain that $\{S\} <^\oplus \{T\}$. Hence, $\{T\}$ is the only sink of the graph $G[H]$.*

DEFINITION 5.5 (SEPARATOR). *A sink $C \in E^i/\sim$ is a* separator *if for every $C' \neq C$ such that $C' \in$ coupled$^\oplus(C)$, we have that $C' <^\oplus C$.*

In the specific case where $E^i/\sim$ contains a single sink C, since $<^\oplus$ is a strict partial order, for any $C' \in E^i/\sim, C' \neq C$, we have that $C' <^\oplus C$ and thus the single sink C is trivially a separator. Thus, for our example graph $G[H]$, $\{T\}$ is the only separator.

In order to prove the existence of a separator, it is not a sufficient condition that the graph is splittable. For example, consider the splittable query $Q = R^i(\underline{x}, y), S^i(\underline{x}, y), T^i(\underline{z}, y)$, which contains two sinks, $\{R, S\}$ and $\{T\}$. It is easy to see that $\{T\} \notin$ *coupled*$^\oplus(\{R, S\})$, and $\{R, S\} \notin$ *coupled*$^\oplus(\{T\})$; thus, $G[Q]$ has no separator. Instead, we show the existence of a separator for a graph that is splittable and *f-closed*.

DEFINITION 5.6 (F-CLOSED GRAPH). *G is* f-closed *if for every $R \in E^i(G)$, $v_R^\oplus \cap u_R^{+,R} \subseteq u_R^\oplus$.*

Indeed, $G[Q]$ is not f-closed, since $v_R^{\oplus} = \{y\}$, $u_R^{+,R} = \{y\}$ and $u_R^{\oplus} = \{x\}$. We will show in Subsection 5.3 that, given a splittable graph G and an instance I, we can always construct in polynomial time a splittable and f-closed graph G' and an instance I' such that $I \models G$ iff $I' \models G'$.

We show in Subsection 5.4 that, if G is splittable and f-closed, there exists a separator, and in fact the separator has an explicit construction:

THEOREM 5.7. *If G is a splittable and f-closed graph, then $C^{sep} = \arg\min_{sink\ C \in E^i/\sim} |coupled^{\oplus}(C)|$ is a separator.*

In other words, the sink C with the smallest $coupled^{\oplus}(C)$ is a separator (there can be many). In the next subsection, we use the existence of a separator to design a recursive polynomial time algorithm for splittable graphs.

5.2 The Recursive Algorithm

We present here an algorithm, RECURSIVESPLIT, that takes as input an instance I and a splittable and f-closed graph G and returns `True` if $I \models G$, otherwise `False`. The algorithm is recursive on the number of inconsistent relations, $|E^i(G)|$ of G. For the base case $E^i(G) = \varnothing$ (all relations are consistent), it is straightforward that RECURSIVESPLIT$(I, G) =$ `True` if and only if $G(I)$ is true.

We next show how to compute RECURSIVESPLIT(I, G) when $|E^i(G)| > 0$. Since G is a splittable and f-closed graph, Theorem 5.7 tells us that there exists a separator C. We partition the edges of E^i into a left ($\mathcal{L}$) and right ($\mathcal{R}$) set as follows:

$$\mathcal{L}^C = \{R \in E^i \mid [R] \in coupled^{\oplus}(C)\} \quad , \quad \mathcal{R}^C = E^i \backslash \mathcal{L}^C$$

Let S_C denote the unique SCC that contains all the sources for the edges in C. Recall from Section 4 that one can use the algorithm FRUGALSCC to compute the compression $A_{S_C}(I)$ of $\mathcal{M}_{S_C}(I)$ in polynomial time, since S_C is a strongly connected graph. Let $\mathcal{A}$ denote the set of all tuples that appear in some or-set of $A_{S_C}(I)$, and $\mathcal{B} = \Pi_{C^{\oplus}}(G^f(I))$. For some $\mathbf{a} \in \mathcal{A}$, we say that $\mathbf{a}$ is *aligned* with $\mathbf{b} \in \mathcal{B}$, denoted $\mathbf{a}\|\mathbf{b}$, if there exists a tuple $t \in G^f(I)$ such that $t[V(S_C)] = \mathbf{a}$ and $t[C^{\oplus}] = \mathbf{b}$. Also, define $algn(\mathbf{b}) = \{\mathbf{a} \in \mathcal{A} \mid \mathbf{a}\|\mathbf{b}\}$. Observe that $\mathbf{a}$ can be aligned with at most one $\mathbf{b}$, since there exists a consistent directed path from every node of $V(S_C)$ to every node of $C^{\oplus}$. Notice also that when $C^{\oplus} = \varnothing$, all the tuples in $\mathcal{A}$ are vacuously aligned with the empty tuple $()$.

For every $\mathbf{b} \in \mathcal{B}$, choose a tuple $t^{(\mathbf{b})} \in G^f(I)$ such that $t^{(\mathbf{b})}[C^{\oplus}] = \mathbf{b}$. For every tuple $\mathbf{a} \in \mathcal{A}$, we now define a subinstance $I[\mathbf{a}] \subseteq I$ such that:

$$R^{I[\mathbf{a}]} = \begin{cases} \{(t^{(\mathbf{b})}[u_R], t^{(\mathbf{b})}[v_R]) \mid \mathbf{b} : \mathbf{a}\|\mathbf{b}\} & \text{if } R \in \mathcal{R}^C, \\ \{(t[u_R], t[v_R]) \mid t \in G^f(I), t[V(S_C)] = \mathbf{a}\} & \text{if } R \in \mathcal{L}^C, \\ R^I & \text{otherwise.} \end{cases}$$

Notice that if some relation R belongs in S_C, then it must contain exactly one tuple, while if u_R belongs in $V(S_C)$, then $R^{I[\mathbf{a}]}$ contains exactly one key-group. On the other hand, the relations that do not belong in $\mathcal{L}^C$ contain only one tuple that contributes to $t^{(\mathbf{b})}$.

The first key idea behind the construction of subinstances is captured by the following lemma, which shows that certain subinstances are independent in the relations of $\mathcal{L}^C$.

LEMMA 5.8. *Let $\mathbf{a}_1, \mathbf{a}_2 \in \mathcal{A}$. The instances $I[\mathbf{a}_1], I[\mathbf{a}_2]$ share no key-groups in any relation $R \in \mathcal{L}^C$ if either of the following two conditions hold:*

1. *$\mathbf{a}_1, \mathbf{a}_2$ belong in different or-sets of $A_{S_C}(I)$.*
2. *$\mathbf{a}_1\|\mathbf{b}_1, \mathbf{a}_2\|\mathbf{b}_2$, and $\mathbf{b}_1 \neq \mathbf{b}_2$.*

The second key idea is that computing whether $I[\mathbf{a}] \models G$ can be reduced to a computation where G contains strictly less inconsistent relations. Indeed, recall that in $I[\mathbf{a}]$, every relation $R_i \in C$, $i = 1, \ldots, m$, contains exactly one key-group, $R_i(\underline{\mathbf{a}[u_{R_i}]}, -)$ (and if it both vertices of R are in S_C, it contains exactly one tuple). We can now apply a "brute force" approach and try all the possible combinations of choices for these key-groups, since they are polynomially many: each such combination will create a new instance where the relations in C will be consistent, and thus certainty for G can be computed in polynomial time by induction. It holds that $I[\mathbf{a}] \models G$ iff every new instance is certain for G. The procedure SIMPLIFY$(I[\mathbf{a}], G)$ describes the algorithm we just sketched.

Algorithm 1: SIMPLIFY$(I[\mathbf{a}], G)$

```
K = {(c_1, ..., c_m) | ∀i : R_i(a[u_{R_i}], c_i)}
G' ← G where all edges of C are of consistent type
∀c ∈ K:
I[a]^(c) ← (I[a] \ ⋃_{i=1}^m R_i(a[u_{R_i}], −)) ⋃_{i=1}^m R_i(a[u_{R_i}], c_i)
return  (∀c ∈ K: RecursiveSplit(I[a]^(c), G') = True)
```

Algorithm 2: RECURSIVESPLIT(I, G)

```
if E^i(G) = ∅ then return G(I)
Find a separator C of G
B ← Π_{C⊕}(G^f(I))
A_{S_C}(I) ← FrugalSCC(I, S_C)
for  b ∈ B do
    if ∃ or-set A ∈ A_{S_C}(I) s.t.
    ∀a ∈ A ∩ algn(b) ⇒ Simplify(I[a], G) = True then
        r[b] ← any repair of (⋃_{a∈algn(b)} I[a])
    else
        r[b] ← ∅
end
∀R ∈ E(G): R^{I'} = { R^I ∩ (⋃_{b∈B} r[b])   if R ∈ L^C,
                      { R^I                   otherwise.
G' ← G where all edges in L^C are of consistent type
return RecursiveSplit(I', G')
```

We can now analyze the algorithm RECURSIVESPLIT, and show that runs in polynomial time and is correct. For its running time, observe first that for the final recursive call on I, G', the graph G' has at most $|E^i(G)| - |\mathcal{L}^C| < |E^i(G)|$ inconsistent edges, so by the induction argument it can be computed in polynomial time. Second, the algorithm calls SIMPLIFY$(I[\mathbf{a}], G)$ at most $|\mathcal{A}|$ times, and we have shown that each such call can be computed in polynomial time.

We next argue that RECURSIVESPLIT correctly computes whether $I \models G$ or not. We prove in [9]:

LEMMA 5.9. *$\bigcup_{\mathbf{a} \in algn(\mathbf{b})} I[\mathbf{a}] \models G$ iff there exists an or-set $A \in A_{S_C}(I)$ such that for every $\mathbf{a} \in A \cap algn(\mathbf{b})$, $I[\mathbf{a}] \models G$.*

Given a repair r of I and a repair r' of $\bigcup_{\mathbf{a} \in algn(\mathbf{b})} I[\mathbf{a}]$, we define $merge^C(r, r')$ as a new repair r_m of I such that for any key-group $R(\underline{a}, -)$, if $R \notin \mathcal{L}^C$ or r' does not contain the key-group, r_m includes the choice of r; otherwise, it includes the choice of r'. In other words, to construct r_m we let r' overwrite r only in the relations of $\mathcal{L}^C$.

LEMMA 5.10. *For any frugal repair r of I:*

1. *If $\bigcup_{\mathbf{a} \in algn(\mathbf{b})} I[\mathbf{a}] \not\models G$ then $\mathbf{b} \notin \prod_{C^{\oplus}} G^f(r)$.*
2. *If $\bigcup_{\mathbf{a} \in algn(\mathbf{b})} I[\mathbf{a}] \models G$ then for any repair r' of the instance $\bigcup_{\mathbf{a} \in algn(\mathbf{b})} I[\mathbf{a}]$, $G(r) = G(merge^C(r, r'))$.*

To see why Lemma 5.9 and Lemma 5.10 imply the correctness of the algorithm, consider first the case where for some $\mathbf{b} \in \mathcal{B}$, for any or-set $A \in A_{S_C}(I)$, there exists some $\mathbf{a} \in A$ that is aligned with $\mathbf{b}$ such that $I[\mathbf{a}] \not\models G$. Then, Lemma 5.9 tells us that $\bigcup_{\mathbf{a} \in algn(\mathbf{b})} I[\mathbf{a}] \not\models G$ and thus, by Lemma 5.10(1), for every frugal repair r of I, $\mathbf{b} \notin \Pi_{C^{\oplus}} G^f(r)$. Hence, all the key-groups of the relations in $\mathcal{L}^C$ that appear in $I[\mathbf{a}]$, for any $\mathbf{a}$ aligned with $\mathbf{b}$, can be safely removed from the instance: this is exactly what setting $r[\mathbf{b}] = \varnothing$ achieves. On the other hand, assume that for some $\mathbf{b} \in \mathcal{B}$, there exists an or-set $A \in A_{S_C}(I)$, where for every $\mathbf{a} \in A \cap algn(\mathbf{b})$, $I[\mathbf{a}] \models G$. Then, Lemma 5.9 tells us that $\bigcup_{\mathbf{a} \in algn(\mathbf{b})} I[\mathbf{a}] \models G$, and by Lemma 5.10(2), whether the instance is certain or not is independent of the choice for the key-groups of $\mathcal{L}^C$ that are contained in $\bigcup_{\mathbf{a} \in algn(\mathbf{b})} I[\mathbf{a}]$.

5.3 f-closed Graphs

In this subsection, we show that we can always reduce in polynomial time G with instance I to an f-closed graph G' with instance I' such that $\mathcal{M}_G(I) = \mathcal{M}_{G'}(I')$. For this, we exploit the following technical lemma.

LEMMA 5.11. *Let $R \in E^i$ and $v \in u_R^{+,R} \cap v_R^{\oplus}$. Let P : $u_R, e_R, v_R, \ldots, v$ be the directed path from u_R to v with e_R as its first edge. If there exist $(a, b_1), (a, b_2) \in \Pi_{u_R, v}(P^f(I))$ such that $b_1 \neq b_2$, then no frugal repair of G contains a.*

Now, consider some instance I of G such that G is not f-closed. We present a polynomial time algorithm, F-CLOSURE, that reduces the graph to an f-closed graph, while keeping the representation $\mathcal{M}_G$ the same. Notice that the algorithm has no specific requirements on the structure of G.

Algorithm 3: F-CLOSURE(I, G)

$I_C \leftarrow I, \quad G_C \leftarrow G$
while $\exists R \in E^i(G_C)$, $v \in V(G_C)$ *such that* $v \in (u_R^{+,R} \cap v_R^{\oplus}) \setminus u_R^{\oplus}$ **do**
  $P = u_R, e_R, v_R, \ldots, v$
  $T = \Pi_{u_R, v}(P^f(I))$
  $I_C \leftarrow I_C \cup \{(a, b) \in T \mid \nexists (a, b') \in T$ where $b' \neq b\}$
  $G_C \leftarrow (V(G_C), E(G_C) \cup \{(u_R, v)\})$
end
return I_C, G_C

PROPOSITION 5.12. *Let I be an instance of graph G.* F-CLOSURE *returns an instance I_C of an f-closed graph G_C in polynomial time such that $\mathcal{M}_G(I) = \mathcal{M}_{G_C}(I_C)$.*

5.4 Proof Sketch of Separator Existence

We sketch here the proof for Theorem 5.7, which states that $C^{sep} = \arg\min_{\text{sink } C \in E^i/\sim} |coupled^{\oplus}(C)|$ is a separator.

Recall that we want to show that for any $C \in E^i/\sim$, where $C \neq C^{sep}$, either $C <^{\oplus} C^{sep}$ or $C \notin coupled^{\oplus}(C^{sep})$. We will show next that it suffices to consider only the sinks $C \in E^i/\sim$, and show that for any sink $C \neq C^{sep}$, $C \notin coupled^{\oplus}(C^{sep})$. Indeed, we show in [9] that for a sink C, the set $coupled^{\oplus}(C)$ is *upward closed*: if $C_0 \in coupled^{\oplus}(C)$ and $C_0 <^{\oplus} C_1$, then also $C_1 \in coupled^{\oplus}(C)$. Note that $coupled^{\oplus}(C)$ is not necessarily upward closed for any C that is not a sink.

LEMMA 5.13. *If C is a sink, $coupled^{\oplus}(C)$ is upward closed.*

Now, suppose that we have shown that for any sink $C \neq C^{sep}$, $C \notin coupled^{\oplus}(C^{sep})$, and consider any $C' \in E^i/\sim$, $C' \neq C$ that is not a sink. Then $C' <^{\oplus} C''$ for some $C'' \in E^i/\sim$ that is a sink; hence, $C'' \notin coupled^{\oplus}(C^{sep})$. However, since C^{sep} is a sink, we can apply Lemma 5.13 to conclude that $C' \notin coupled^{\oplus}(C^{sep})$.

The bulk of the proof consists of two technical results, which we prove in detail in the full version of this paper [9]. The first result tells us that for a sink C, the two types of coupling coincide: $coupled^{+}(C) = coupled^{\oplus}(C)$.

PROPOSITION 5.14. *Let G be a splittable and f-closed graph. For any sink $C \in E^i/\sim$, $C^+ = C^{\oplus}$.*

The second result tells us that for two distinct equivalence classes C_1, C_2 where $C_1 \in coupled^{+}(C_2)$, $coupled^{+}(C_1)$ is strictly contained in $coupled^{+}(C_2)$.

PROPOSITION 5.15. *Let G a splittable graph and $C_1, C_2 \in E^i/\sim$ such that $C_1 \neq C_2$. Then,*

1. *Either $C_1 \notin coupled^{+}(C_2)$ or $C_2 \notin coupled^{+}(C_1)$.*
2. *If $C_1 \in coupled^{+}(C_2)$, $coupled^{+}(C_1) \subset coupled^{+}(C_2)$.*

Now, consider a sink $C \neq C^{sep}$. If $C \in coupled^{+}(C^{sep})$, then by Proposition 5.15(2) and Proposition 5.14 it must be that $coupled^{\oplus}(C^{sep}) = coupled^{+}(C^{sep}) \supset coupled^{+}(C) = coupled^{\oplus}(C)$. However, this contradicts the minimality of $coupled^{\oplus}(C^{sep})$, and proves our theorem.

6. THE CONP-COMPLETE CASE

In this section, we prove part (2) of Theorem 3.6: if $G[Q]$ is unsplittable, then CERTAINTY(Q) is coNP-complete. We reduce CERTAINTY(Q) from MONOTONE-3SAT, which is a special case of 3SAT where each clause contains only positive or only negative literals. We say that a clause is positive (negative) if it contains only positive (negative) literals. MONOTONE-3SAT is known to be NP-complete [7].

Given an instance M of MONOTONE-3SAT, let us denote by Φ the set of all clauses, X the set of all variables, X^* the set of all literals and $\mathbb{B} = \{T, F\}$ (true, false). Moreover, let us define $\top = \Phi \times \mathbb{B} = \{(\phi, x^*) \mid x^* \in \phi, \phi \in \Phi\}$ and $\bot = \{()\}$. We order the set $\mathcal{L} = \{\bot, \mathbb{B}, X, \Phi, X^*, \top\}$ as shown in Figure 6: $\bot$ and $\top$ are the minimal and maximal elements, and $\mathbb{B} \leqslant \Phi$, $X \leqslant X^*$ and $\mathbb{B} \leqslant X^*$. The reader may check that $\mathcal{L}$ is a lattice. For example, $\Phi \wedge X^* = \mathbb{B}$ and $\mathbb{B} \vee X = X^*$.

DEFINITION 6.1 (VALID LABELING). *Let $R, S \in E^i$. A labeling $L : V(G) \rightarrow \mathcal{L}$ is* (R, S)-valid *if the following conditions hold:*

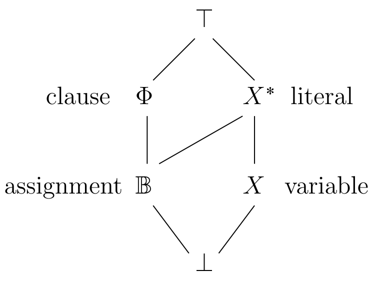

Figure 6: The lattice of the set of labels $\mathcal{L}$.

1. *$L(u_R) = \Phi$ and $L(v_R) \in \{\top, X, X^*\}$.*
2. *$L(u_S) = X$ and $L(v_S) \in \{\mathbb{B}, X^*\}$.*
3. *For every $T \in E^i \backslash \{R, S\}$, $L(u_T) \geqslant L(v_T)$.*
4. *$\exists P_R : v_R \leftrightarrow u_S$ such that $\forall v \in P_R, L(v) \geqslant X$.*
5. *$\exists P_S : v_S \leftrightarrow u_R$ such that $\forall v \in P_S, L(v) \geqslant \mathbb{B}$.*

PROPOSITION 6.2. *If $R, S \in E^i$ are coupled and $S \not\lesssim R$, then G admits a (R, S)-valid labeling.*

If the query Q has an unsplittable graph $G = G[Q]$, then there exists two coupled edges R, S s.t. $R \not\sim S$. This implies that we cannot have both $R \lesssim S$ and $S \lesssim R$, and the proposition tells us that G has an (R, S)-valid labeling. We will show later how to use this labeling to reduce M to CERTAINTY(Q). First, we prove the proposition.

PROOF. Since $S \in \textit{coupled}^+(R)$, there exists a path $P_R : v_R \leftrightarrow u_S$ s.t. $P_R \cap u_R^{+,R} = \varnothing$; similarly, there exists a path $P_S : v_S \leftrightarrow u_R$ s.t. $P_S \cap u_S^{+,S} = \varnothing$. Notice that, in particular, P_R contains the source and destination nodes v_R, u_S, and, similarly, P_S contains the nodes v_S, u_R, which implies:

$$v_R \notin u_R^{+,R} \qquad u_S \notin u_R^{+,R} \qquad v_S \notin u_S^{+,S} \qquad u_R \notin u_S^{+,S} \tag{4}$$

We define the label L as follows. Let $W = \{u_R, v_R, u_S, v_S\}$ and set the initial labels for the four nodes in W:

$$L_0(u_R) = \Phi, \quad L_0(v_R) = \top, \quad L_0(u_S) = X, \quad L_0(v_S) = X^*$$

For every $v \in V(G)$, let $\mathcal{W}^{-1}(v) = \{w \mid w \in W, v \in w^{+,R,S}\}$, where $w^{+,R,S}$ is the set of nodes reachable from w by a directed path that does not go through either R or S. In other words, $\mathcal{W}^{-1}(v)$ is the subset of the four distinguished nodes that can reach v without using R or S. Trivially, $w \in \mathcal{W}^{-1}(w)$, for every $w \in W$. Define the labeling L as follows:

$$\forall v \in V(G) : \quad L(v) = \bigwedge \{L(w) \mid w \in \mathcal{W}^{-1}(v)\}$$

We will show that this labeling is (R, S)-valid. We start by checking properties (1) and (2). Consider each of the four distinguished nodes in W:

u_R**:** The set $\mathcal{W}^{-1}(u_R)$ is either $\{u_R\}$ or $\{u_R, v_R\}$; indeed $v_S \notin \mathcal{W}^{-1}(u_R)$ because $S \not\lesssim R$, and $u_S \notin \mathcal{W}^{-1}(u_R)$ by Eq.(4). By definition, either $L(u_R) = \Phi$ or $L(u_R) = \Phi \wedge \top = \Phi$; in both cases $L(u_R) = \Phi$.

u_S**:** We have $\{u_S\} \subseteq \mathcal{W}^{-1}(u_S) \subseteq \{u_S, v_R, v_S\}$, because Eq.(4) implies $u_S \notin u_R^{+,R,S}$. This implies $X = L_0(u_S) \geqslant L(u_S) \geqslant L_0(u_S) \wedge L_0(v_R) \wedge L_0(v_S) = X \wedge \top \wedge X^* = X$, hence $L(u_S) = X$.

v_R**:** We have $\{v_R\} \subseteq \mathcal{W}^{-1}(v_R) \subseteq \{u_S, v_R, v_S\}$, because Eq.(4) implies $v_R \notin u_R^{+,R,S}$. Therefore, $\top \geqslant L(v_R) \geqslant X \wedge \top \wedge X^* = X$, implying $L(v_R) \in \{X, X^*, \top\}$.

v_S**:** We have $\{v_S\} \subseteq \mathcal{W}^{-1}(v_S) \subseteq \{u_R, v_R, v_S\}$, because Eq.(4) implies $v_S \notin u_S^{+,R,S}$. Therefore, $X^* \geqslant L(v_S) \geqslant \Phi \wedge \top \wedge X^* = \mathbb{B}$, implying $L(u_S) \in \{\mathbb{B}, X^*\}$.

To show property (3), consider an edge $e_T = (u_T, v_T)$, $T \neq R, S$. Then $\mathcal{W}^{-1}(u_T) \subseteq \mathcal{W}^{-1}(v_T)$ which implies $L(u_T) \geqslant L(v_T)$.

For (4), let P_R be the undirected path defined earlier s.t. $P_R \cap u_R^{+,R} = \varnothing$; we also have $P_R \cap u_R^{+,R,S} = \varnothing$. Let $v \in P_R$ be any node on this path. Then $u_R \notin \mathcal{W}^{-1}(v)$, which implies that $\mathcal{W}^{-1}(v) \subseteq \{v_R, u_S, v_S\}$, and therefore $L(v) \geqslant \top \wedge X \wedge X^* = X$.

Finally, for (4), let P_S be the undirected path defined earlier, s.t. $P_S \cap u_S^{+,S} = \varnothing$. For any node $v \in P_S$ we have $\mathcal{W}^{-1}(v) \subseteq \{u_R, v_R, v_S\}$, thus $L(v) \geqslant \Phi \wedge \top \wedge X^* = \mathbb{B}$. $\square$

Next, we show how to use a valid labeling to reduce the MONOTONE-3SAT Φ to CERTAINTY(Q).

The Functions $f_{L_1 L_2}$. For any pair of sets $L_1, L_2 \in \mathcal{L}$ such that $L_1 \geqslant L_2$, we define a function $f_{L_1 L_2} : L_1 \to L_2$, as follows. First, for the seven pairs L_1, L_2 where L_1 covers[7] L_2, we define $f_{L_1 L_2}$ directly:

$(\Phi, \mathbb{B})$: $f_{\Phi,\mathbb{B}}(\phi) = T$ if ϕ is a positive clause, else F
(X^*, X) : $f_{X^*,X}(x^+) = f_{X^*,X}(x^-) = x$
$(X^*, \mathbb{B})$: $f_{X^*,\mathbb{B}}(x^+) = T$ and $f_{X^*,\mathbb{B}}(x^-) = F$
$(\top, \Phi)$: $f_{\top,\Phi}((\phi, x^*)) = \phi$
$(\top, X^*)$: $f_{\top,X^*}((\phi, x^*)) = x^*$
$(\mathbb{B}, \bot), (X, \bot)$: $f_{\mathbb{B},\bot}(b) = f_{X,\bot}(x) = ()$

Next, we define $f_{LL} = id_L$ (the identity on L) and $f_{L_1 L_3} = f_{L_2 L_3} \circ f_{L_1 L_2}$ for all $L_1 \geqslant L_2 \geqslant L_3$. Readers familiar with category theory will notice that we have transformed the lattice $\mathcal{L}$ into a category.

Instance Construction. We define the instance I, by defining a binary relation T^I for every relation name T. Let $L_1 = L(u_T)$, $L_2 = L(v_T)$. We distinguish two cases, depending on whether T is R, S or not.

If $T \neq R$, $T \neq S$, then we know that $L_1 \geqslant L_2$. Define $T^I = \{(a, b) \mid a \in L_1, b = f_{L_1 L_2}(a) \in L_2\}$. Notice that the first attribute of T^I is a key (because $f_{L_1 L_2}$ is a function), and therefore T^I always satisfies the key constraint.

If $T = R$ or $T = S$, then $L_1 \not\geqslant L_2$. In this case we construct R^I and S^I to be a certain set of pairs (a, b), $a \in L_1, b \in L_2$, where b is obtained from a by either going "back" (b) in the lattice, or going "back and forth" (b-f), depending on the combination of L_1, L_2 given by Definition 6.1:

$(\Phi, \top)$: $R^I = \{(a, b) \mid b \in f^{-1}_{\top,\Phi}(a)\}$ (b)
(Φ, X^*) : $R^I = \{(a, b) \mid \exists c \in f^{-1}_{\top,\Phi}(a) : f_{\top,X^*}(c) = b\}$ (b-f)
(Φ, X) : $R^I = \{(a, b) \mid \exists c \in f^{-1}_{\top,\Phi}(a) : f_{\top,X}(c) = b\}$ (b)
(X, X^*) : $S^I = \{(a, b) \mid b \in f^{-1}_{X^*,X}(a)\}$ (b)
$(X, \mathbb{B})$: $S^I = \{(a, b) \mid \exists c \in f^{-1}_{X^*,X}(a) : f_{X^*,\mathbb{B}}(c) = b\}$ (b-f)

Notice that in all cases, R^I and S^I are inconsistent. For example, in the first case, a repair of R^I chooses for each clause $\phi \in \Phi$ a value (ϕ, b) with $b \in \mathbb{B}$.

[7]In a lattice, L_1 *covers* L_2 if $L_1 > L_2$ and there is no L_3 s.t. $L_1 > L_3 > L_2$.

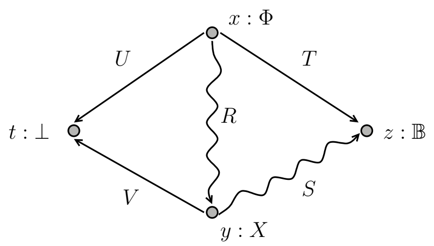

Figure 7: A query graph with a (R, S)-valid labeling.

EXAMPLE 6.3. *Consider the formula* $Y = \phi_1 \wedge \phi_2$*, where* $\phi_1 = (x^+ \vee y^+ \vee z^+)$ *and* $\phi_2 = (z^- \vee w^- \vee t^-)$*. If the inconsistent relation* R *is labeled with* (Φ, X)*, it will be populated by the tuples* $(\phi_1, x), (\phi_1, y), (\phi_1, z)$ *and* $(\phi_2, z), (\phi_2, w), (\phi_2, t)$*. On the other hand, a consistent relation* $T \neq R, S$ *that is labeled with* $(\Phi, \mathbb{B})$ *will contain the tuples* $(\phi_1, T), (\phi_2, F)$*.*

Thus, given a valid labeling we can create a database instance using the above construction. We prove in [9]:

PROPOSITION 6.4. *Let* I *be the instance that corresponds to a* (R, S)*-valid labeling according to an instance* M *of* MONOTONE-3SAT*. Then,* $I \not\models Q$ *if and only if* M *has a satisfying assignment.*

EXAMPLE 6.5. *Consider the query of Figure 7. Notice that* $R \lesssim S$*. Also,* $u_R = x$*,* $v_R = u_S = y$ *and* $v_S = z$*. Since* $L^+(x) = \{L_0(u_R)\} = \{\Phi\}$*,* $L(x) = \Phi$*. Also,* $L^+(y) = \{L_0(v_R), L_0(u_S)\} = \{\top, X\}$*, hence* $L(y) = \top \wedge X = X$*. For variable* z*,* $L^+(z) = \{L_0(v_S), L_0(u_R)\} = \{\Phi, X^*\}$ *and* $L(z) = \Phi \wedge X^* = \mathbb{B}$*.* $L^+(t) = \{L_0(u_R), L_0(v_R), L_0(u_S)\} = \{\Phi, \top, X\}$ *and hence* $L(t) = \Phi \wedge \top \wedge X = \bot$*.*

7. RELATED WORK

The consistent query answering framework was first proposed by Arenas et al. in [2]. Fuxman and Miller [6] focused on primary key constraints, with the goal of specifying conjunctive queries where CERTAINTY(Q) is first-order expressible, *i.e.* can be represented as a boolean first-order query over the inconsistent database. They presented a class of acyclic conjunctive queries w/o self-joins, called C_{forest}, that allows such first-order rewriting. Further, Fuxman et al. [5] designed and built a system that supported the query rewriting functionality for consistent query answering.

In a series of papers [12, 14], Wijsen improved on the results for first-order expressibility. The author presented a necessary and sufficient syntactic condition for the first-order expressibility for acyclic conjunctive queries without self-joins. In a later paper, Wijsen [13] gave a polynomial time algorithm for the query $Q_2 = R(\underline{x}, y), S(\underline{y}, x)$, which is known to be not first-order expressible. Q_2 is the first query that was proven to be tractable even though it does not admit a first-order rewriting. Kolaitis and Pema [8] proved a dichotomy for the complexity of CERTAINTY(Q) when the query has only two atoms and no self-joins into polynomial time and coNP-complete. Finally, Wijsen [15] recently classified several acyclic queries into PTIME and coNP-complete, without however showing the complete dichotomy for acyclic queries without self-joins.

A relevant problem to consistent query answering is the counting version of the problem: given a query and an inconsistent database, count the number of repairs that satisfy the query. Maslowski and Wijsen [11] showed that this problem admits a dichotomy in P and #P-complete for conjunctive queries without self-joins.

Finally, we should mention that the problem of consistent query answering is closely related to probabilistic databases, in particular *disjoint-independent* probabilistic databases [4]. Wijsen in [15] discusses the precise connection between the complexity of evaluating a query Q on probabilistic databases and CERTAINTY(Q).

8. CONCLUSION

In this paper, we make significant progress towards proving a dichotomy on the complexity of CERTAINTY(Q), studying the case where Q is a Conjunctive Query without self-joins consisting of atoms with simple keys or keys containing all attributes. It remains a fascinating open question whether a dichotomy exists for general conjunctive queries, even in the case where there are no self-joins.

Acknowledgments. This work is supported in part by the NSF through NSF grant IIS-0915054 and IIS-1115188.

9. REFERENCES

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases.* Addison-Wesley, 1995.

[2] M. Arenas, L. E. Bertossi, and J. Chomicki. Consistent query answers in inconsistent databases. In *PODS*, pages 68–79. ACM Press, 1999.

[3] J. Chomicki and J. Marcinkowski. Minimal-change integrity maintenance using tuple deletions. *Inf. Comput.*, 197(1-2):90–121, 2005.

[4] N. N. Dalvi and D. Suciu. Management of probabilistic data: foundations and challenges. In *PODS*, pages 1–12. ACM, 2007.

[5] A. Fuxman, D. Fuxman, and R. J. Miller. Conquer: A system for efficient querying over inconsistent databases. In *VLDB*, pages 1354–1357. ACM, 2005.

[6] A. Fuxman and R. J. Miller. First-order query rewriting for inconsistent databases. *J. Comput. Syst. Sci.*, 73(4):610–635, 2007.

[7] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman & Co., New York, NY, USA, 1979.

[8] P. G. Kolaitis and E. Pema. A dichotomy in the complexity of consistent query answering for queries with two atoms. *Inf. Process. Lett.*, 112(3):77–85, 2012.

[9] P. Koutris and D. Suciu. A dichotomy on the complexity of consistent query answering for atoms with simple keys. *CoRR*, abs/1212.6636, 2012.

[10] L. Libkin and L. Wong. Semantic representations and query languages for or-sets. In *PODS*, pages 37–48. ACM Press, 1993.

[11] D. Maslowski and J. Wijsen. On counting database repairs. In *LID*. ACM, 2011.

[12] J. Wijsen. Consistent query answering under primary keys: a characterization of tractable queries. In *ICDT*, volume 361 of *ACM International Conference Proceeding Series.* ACM, 2009.

[13] J. Wijsen. On the first-order expressibility of computing certain answers to conjunctive queries over uncertain databases. In *PODS.* ACM, 2010.

[14] J. Wijsen. A remark on the complexity of consistent conjunctive query answering under primary key violations. *Inf. Process. Lett.*, 110(21), 2010.

[15] J. Wijsen. Charting the tractability frontier of certain conjunctive query answering. In *PODS.* ACM, 2013.