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 \( \text{coNP} \) 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 \( \text{PTIME} \) or \( \text{coNP-complete} \). 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 \( \text{coNP} \) [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 \( Q_1 = R(x, y), S(y, z) \), CERTAINTY\((Q_1)\) is not only in \( \text{P} \) but, since one can show that CERTAINTY\((Q_1)\) can be expressed as a first-order query over \( I \) [6], it is in \( \text{AC}^0 \). On the other hand, for \( Q_2 = R(x, y), S(x, y) \), it has been proved in [6] that CERTAINTY\((Q_2)\) is \( \text{coNP-complete} \). Finally, for \( Q_3 = R(x, y), S(y, x) \), 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 \( \text{PTIME} \) or is \( \text{coNP-complete} \).

The progress that has been made towards proving this conjecture has been limited. In particular, Kolaitis and Pema [8] have proved a dichotomy into \( \text{PTIME} \) and \( \text{coNP-complete} \) 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 \( \text{PTIME} \) and \( \text{coNP-complete} \).

In this work, we significantly progress the status of the conjecture, by settling the dichotomy for a large class of
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 $R_1(x, y), R_2(x, y)$. If an instance contains the tuples $R_1(b_1, b_1), R_2(b_2, b_2)$ such that $b_1 \neq b_2$, we can remove the key-groups $R_1(\bot, -), R_2(\bot, -)$ without loss of generality in order to check for certainty. Thus, the conjunction of $R_1, R_2$ behaves as a single consistent relation $R(x, y)$. 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 $\text{Certainty}(Q)$ into $\text{PTIME}$ 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 $\text{Certainty}(Q)$ into $\text{PTIME}$ 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 $\text{PTIME}$ 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 $R(x, y)$.

Given the graph $G[Q]$, we give a necessary and sufficient condition for $\text{Certainty}(Q)$ 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 $u_R$ to $v_R$ 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 $u_S$ where no node in $P_S$ is reachable from $u_S$ through a directed path in $G - \{e_S\}$. Then:

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

The following example illustrates the main theorem.

---

1Indeed, 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:

\[ K_1 = R(x, y), S(z, w), T^z(y, w) \]
\[ K_2 = R(x, y), S(z, w), T^z(y, w), U^z(x, z) \]

Observe that the only difference between $K_1, K_2$ is the consistent relation $U$. 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 $e_R, e_S$ 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, $\text{Certainty}(K_1)$ is coNP-complete.

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

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 $\text{Certainty}(Q)$, 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 $\text{Certainty}(Q)$ is in $\text{PTIME}$. The algorithm for computing $\text{Certainty}(Q)$ in this case is based on a novel use of or-sets to represent efficiently answers to repairs. The polynomial time algorithm for $\text{Certainty}(Q)$ 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 $\text{attr}(R) = \{A_1, \ldots, A_k\}$, and a key, which is a subset of $\text{attr}(R)$. We typically write $R(x_1, \ldots, x_m, y_1, \ldots, y_l)$ 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 $R'$ for each relation name $R$, such that, if $R$ is of consistent-type, then $R'$ satisfies its key constraint. In other words, in an instance $I$ we allow relations $R'$ to violate the key constraints but always require the relations $R'$ to satisfy the key constraints. Notice that, if the key of $R$ consists of all attributes, then $R'$ always satisfies the key constraints, so we may assume w.l.o.g. that $R$ is of consistent-type.

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

166
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 $Q^f(r') \subseteq Q^f(r)$.

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

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

with the following repairs:

$r_1 = \{R(a_1, b_1), R(a_2, b_3), S(a_1, b_1), S(a_2, b_3), R(a_3, b_4), S(a_3, b_1), S(a_2, b_4), S(a_3, b_3)\}$

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

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

Then, the answer sets are $Q^f(r_1) = \{(a_1, b_1), (a_2, b_1), (a_3, b_3)\}$ and $Q^f(r_2) = \{(a_2, b_2), (a_3, b_3)\}$ respectively. Since $Q^f(r_3) \subseteq Q^f(r_1)$, the repair $r_3$ 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 $Q^f(r) \neq \emptyset$: therefore $r$ is a frugal repair, proving the claim.

The proposition also implies that we lose no generality if we study only frugal repairs in certain query answering. To check $I \models Q$ 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, $(a, 2, 3)$ denotes the or-set that is either 1 or 2 or 3: similarly $(\{1, \{1, 3\}\}$ means either the set $\{1\}$ or $\{1, 3\}$.

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

$$M_Q(I) = \langle Q^f(r) \mid r \in F_Q(I) \rangle$$

be the or-set of all answers of $Q^f$ on all frugal repairs. Notice that the type of $M_Q(I)$ is $\langle I^*_T \rangle$, where $T = \times_{i=1}^n T_i$ is a product of atomic types. For a simple illustration, in Example 2.4, $M_Q(I) = \langle\{(a_2, b_1), (a_3, b_1)\}, \{(a_2, b_2), (a_3, b_2)\}\rangle$, because $r_2, r_3$ are the only frugal repairs.

Give a type $T$, define the function $\alpha : \langle T \rangle \to \langle T \rangle [10]$:

$$\alpha([A_1, \ldots, A_m]) = \langle [x_1, \ldots, x_m] \mid [x_1 \in A_1, \ldots, x_m \in A_m] \rangle.$$ For example, $\alpha([1, 2], [3, 4]) = \langle [1, 3], [1, 4], [2, 3], [2, 4]\rangle$ and $\alpha(([1), [1, 2, 3])] = \langle [1], [1, 2], [1, 3]\rangle$.

Definition 2.6. Let $T = \times_{i=1}^n T_i$. 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 $A, B \subseteq S_0$, 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

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

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

$S$ is representable, since we can find a compression $S_0 = \{(a_1, b_1), (a_2, b_2), \langle(a_2, b_2)\rangle\}$. Notice that $a_1, b_2$ 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 $S$ be an or-set-of-sets of type $\langle \times_{i=1}^n T_i \rangle$, and suppose that its active domain has size $n$. If $S$ is representable, then its compression $S_0$ has size $O(n^n)$.

Proof. Let $S_0 = \{A_1, A_2, \ldots\}$, 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 $O(n^n)$.

If $M_Q(I)$ is representable, then we denote $A_Q(I)$ its compression; its size is at most polynomially large in $I$. In general, $A_Q(I)$ may not be representable.

By the definition of frugality, if $s_1, s_2 \in M_Q(I)$ then neither $s_1 \subseteq s_2$ nor $s_2 \subseteq s_1$ holds. This implies that, for any instance $I$, there are two cases. Either (1) $I \not\models Q$; in that case $M_Q(I) = \{\}$ is trivially representable as $A_Q(I) = \{\}$; or, (2) $I \models Q$, and in that case $M_Q(I) = \langle A_1, A_2, \ldots\rangle$, where $A_i \neq \{\}$ for all $i$, may be exponentially large and not necessarily representable. For a simple illustration, in Example 2.4, $M_Q(I)$ is representable, and its compression is $A_Q(I) = \langle\{(a_2, b_1), \langle(a_3, b_4), (a_1, b_1)\}\rangle$.

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 $\not\models \{\}$. The converse is not true, however: for example, consider the query $H = R(x, y), S(y, z)$, for which Certainty$(H)$ is in PTIME. However, for the instance $I' = \{R(a, b), S(b, c_1), S(b, c_2)\}$, $M_H(I') = \langle\{(a, b, c_1), (a, b, c_2)\}\rangle$ is not representable.
2.3 Purified Instances

Let \( Q \) be a any boolean conjunctive query. An instance \( I \) is called globally consistent [1, ... p Rq  r Rs Y t S P Ei | D P : vR Ø uS,P X u`, coupled p Rq  r Rs Y t S P Ei | D P : vR Ø uS,P X u ,Hu

LEMMA 2.8. Given a query \( Q \) and an instance \( I \), there exists a purified instance \( I^p \subseteq I \) such that \( M_Q(I) = M_Q(I^p) \), and thus \( I = Q \) if and only if \( I^p = Q \).

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, 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 \( R(u,v) \) 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 \( E^c(G) \) (\( E^i(G) \)) the set of all consistent (inconsistent) edges. Thus \( E(G) = E^c(G) \cup E^i(G) \).

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 \( \ell \geq 0 \). 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 \( P : x \sim y \) 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 \( \ell \geq 0 \); we write \( P : x \leftrightarrow y \) for an undirected path where \( v_0 = x \) and \( v_\ell = y \) (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 \( N \in V(G) \), then \( P \subseteq N \) denotes the set of vertices in \( P \) that occur in \( N \). The notation \( x \to y \) (or \( x \sim y \) or \( x \leftrightarrow y \)) means “there exists a path \( P : x \to y \)” (or \( P : x \sim y \), 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:

\[
H = R_1(x,y), R_2(y,z), R_3(z,x), V_1^1(u, y), V_1^2(\bar{u}, x), V_2^1(z, v), S(u, v), T(x, v), U^1(u, v, w)
\]

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

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 \( Q = R(u, v), S(y, z), T(z, x) \) then we will assume that \( \Pi_1(R^1) \cap \Pi_1(S^1) = \emptyset \), 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 \( R^1(a, b) \) 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,

\[
\begin{align*}
\bar{u} & = \{ v \in V(G) \mid u \not\to v \in G \} \\
u^+ & = \{ v \in V(G) \mid u \sim v \in G \} \\
u^{+R} & = \{ v \in V(G) \mid u \not\leftrightarrow v \in G \}
\end{align*}
\]

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

\[
\begin{align*}
x^{\bar{R}} &= \{x, v\} \\
x^{+R_1} &= \{x, v, w\} \\
x^{+} &= \{x, v, w, y, z\}
\end{align*}
\]

PROPOSITION 3.2. If \( R \in E^+, \bar{u}^{R} \cap u^{+R} \subseteq u^{+R} \).

Proof. Let \( v \in u^{\bar{R}} \). 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 \( G \) as well. Consequently, \( v \in u^{+R} \). The other inclusion is straightforward.

Define the binary relation \( R \leq S \) if \( u_S \subseteq u_R \). The relation \( \leq \) is a preorder on the sets of edges, since it is reflexive and transitive. If \( R \leq S \) and \( S \leq T \) then we say that \( R, S, T \) 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 \( R \in E^+ \), we define the following sets of coupled edges:

\[
\begin{align*}
\text{coupled}^\bar{R}(R) &= \{R^1 \cup \{S \in E^+ \mid 3P : u_R \leftrightarrow u_S, P \not\subseteq u_R^{\bar{}} = \emptyset\} \\
\text{coupled}^+(R) &= \{R^1 \cup \{S \in E^+ \mid 3P : u_R \leftrightarrow u_S, P \cap u_R^{+} = \emptyset\}
\end{align*}
\]
By definition, every edge $S$ that is source-equivalent to $R$ is coupled with $R$. In addition, coupled$^p(R)$ (coupled$^q(R)$) includes all inconsistent edges $S$ whose source node $u_S$ is in the same weakly connected component as $v_R$, in the graph $G - u_S^p$ ($G - u_R^q$ respectively). The notion of coupled$^p(R)$ 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:

\[ z^B = \{x, v\}, \quad x^+R_1 = \{x, v, w\}, \quad z^B = \{x, v\}, \quad z^+.R_3 = \{x, v, w\}, \quad u^B = \{u, y, w\}, \quad u^+.S = \{u, y, w, x, v, w\}, \quad v^B = \{v\}, \quad v^+.T = \{v\} \]

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:

- coupled$^B(R_1) = \{R_1, R_3, S\}$
- coupled$^B(R_3) = \{R_1, R_3, S\}$
- coupled$^B(S) = \{S\}$
- coupled$^B(T) = \{R_1, R_3, S, T\}$

Proposition 3.2 implies:

**Corollary 3.4.** If $R \in E^i$, coupled$^p(R) \equiv$ coupled$^q(R)$.

**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 $G[H]$ 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 $R_1 \not\in$ coupled$^*(S)$; $R_1, T$ are not coupled because $T \not\in$ 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 weakly connected components $Q_1, Q_2, \ldots$, in other words, $Q_t, Q_j$ 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 $Q_t$ that contains $R$. In this case, Theorem 3.6 implies that Certainty$(Q)$ is coNP-complete iff Certainty$(Q_t)$ is coNP-complete for some $i$. Second, if $Q$ is strongly connected, then it is, by definition, splittable: in case 3 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.

\[
\begin{array}{ccc}
R(\overline{z}, y) & S(y, z) & T(\overline{z}, x) \\
(a_1, b_1) & (b_1, c_1) & (c_1, a_1) \\
(a_2, b_2) & (b_2, c_2) & (c_2, a_2) \\
(a_3, b_3) & (b_3, c_3) & (c_3, a_3) \\
(a_4, b_4) & (b_4, c_3) & (c_3, a_4)
\end{array}
\]

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: $Q_1 = R(\overline{z}, y), Q_2 = R(\overline{z}, y), S(\overline{z}, y), Q_3 = R(\overline{z}, y), S(\overline{z}, y), T(\overline{z}, y)$, 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 $R \neq S$, thus, $Q_2$ is unsplittable. On the other hand, in $Q_1$ we have 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, $M_Q(I)$ 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 $A_Q(I) = \emptyset$; 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 $k \geq 2$, the cycle query $C_k$ is defined as:

$C_k = R_1(\overline{x}_1, x_2), R_2(\overline{x}_2, x_3), \ldots, R_k(\overline{x}_k, x_1)$

Wijten [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 $k \geq 2$ 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.

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

169
**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.

**Algorithm.** Fix $k \geq 2$. The algorithm FRUGALC takes as input a purified instance $I$ and returns the compression $A_{C_k}(I)$ of $MC_k(I)$, in four steps:

1. Compute the SCCs of $FC_3(I)$: $FC_3(I) = F_1 \cup \ldots \cup F_m$, 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: $A_i = \langle \langle a_1, \ldots, a_k \rangle \mid a_1, \ldots, a_k$ 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 $MC_k(I)$. Note that $I \in C_k$ iff $A_{C_k}(I) = \{\}$. If $I \in C_k$ iff $A_{C_k}(I) = \{\} \iff S = \emptyset$.

**Example 4.4.** We illustrate the algorithm on the query $C_3 = R(x, y), S(y, z), T(z, x)$. Consider the relations $R, S, T$ of the instance $I$ in Figure 2 and its graph $FC_3(I) = F_1 \cup F_2$ 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 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: $A_{C_3}(I) = \{\langle \langle a_1, b_1, c_1, a_2, b_2, c_2 \rangle \rangle\}$.

It remains to show that the algorithm is correct, and this follows from two lemmas. Recall from Subsection 2.2 that $FC_3(I)$ denotes the or-set of frugal repairs of $I$ for $C_3$. 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 $FC_3(I)$.

**Lemma 4.5.** For the frugal repairs of $I$, $FC_3(I) = \langle r_1 \cup \ldots \cup r_m \mid r_i \in FC_3(I_1), \ldots, r_m \in FC_3(I_m)\rangle$.

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 $FC_3(I)$ is strongly connected. Then:

\[
MC_k(I) = \begin{cases} \langle 1 \rangle & \text{if } I \text{ has a cycle of length } k, \\ \langle \{a_1, \ldots, a_k\} \rangle & \text{if } a_1, \ldots, a_k \text{ cycle in } FC_3(I) \\ \emptyset & \text{else} \end{cases}
\]

The lemma says two things. On one hand, if $I$ has a cycle of length $> k$, then $I \notin C_k$. Consider the case when all cycles in $I$ have length $k$. In general, if $r$ is a minimal repair, then the full query $C_k^*(r)$ may return any nonempty set of $k$-cycles. The lemma states that if $r$ is a frugal repair, then $C_k^*(r)$ 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 $> k$, $MC_k(I)$ is represented $A_{C_k}(I) = \{\langle a_1, \ldots, a_k \rangle \mid a_1, \ldots, a_k$ cycle in $FC_3(I)\}$; and if $I$ has a cycle of length $> k$ then $A_{C_k}(I) = \{\}$.

**4.2 A PTIME Algorithm for SCGs**

We now present the general algorithm that computes the compression $A_{C_k}(I)$ 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 path $P = u_0 \leadsto v$ 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 $Q_0 \cup P$, where $Q_0$ and $P$ are sets of atoms s.t. $P$ is a simple path 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 a strongly connected graph. Then there exists a sequence $G_0 \subseteq \ldots \subseteq G_m = G$ 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 $G_{i-1}$

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

\[5\text{Meaning that } P = R_1(x_1, x_2, x_3), R_2(x_1, x_2), \ldots, R_m(x_m, x_{m+1}), \text{ all variables } x_1, x_3, \ldots, x_{m} \text{ are distinct, and all variables } x_1, x_3, \ldots, x_{m} \text{ are distinct.}\]
Example 4.9. We will study the conjunctive query $H_2 = R(x, y), S(y, z), T(z, x), U(y, t), V(t, z)$. The query admits the following decomposition:

$$G_0 = G_0[q] \quad \text{where } q = R(x, y), S(y, z), T(z, x)$$

$$G_1 = G_0 \cup P \quad \text{where } P = U(y, t), V(t, z)$$

Our algorithm for computing $\text{CERTainty}(Q)$ 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 $Q_0, R^*(u, v)$, where $R^*(u, v)$ is a chord $Q$. The procedure FrugalChord takes an 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:

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

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 $Q \equiv Q_0, R^*(u, v)$ such that $R^*(u, v)$ is a chord of $Q_0$. If $\mathcal{M}_{Q_0}(I)$ is representable and its compression is $A_{Q_0}(I)$, then $\mathcal{M}_{Q}(I)$ 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:

$$A_{Q_0}(I) = \{ A_1, A_2, \ldots, A_m \} \quad (2)$$

Denote $n = \sum n_i$. Let $a_i$ for $i = 1, m$ be distinct constants, and let $b_{ij}$ for $i = 1, m, j = 1, n_i$ be distinct constants. Denote $\text{tup}(b_{ij}) = t_{ij}$ the tuple encoded by $b_{ij}$.

2. Create two new relations:

$$B^1_i = \{ (a_i, b_{ij}) \mid i = 1, m; j = 1, n_i \}$$

$$B^2_i = \{ (b_{ij}, \pi_0(t_{ij}) \mid i = 1, m; j = 1, n_i \}$$

$$B^3_i = \{ (\pi_1(t_{ij}), a_i) \mid i = 1, m; j = 1, n_i \}$$

$B^1_i$ is of inconsistent type (hence the superscript "in"), and $B^2_i, B^3_i$ 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 $Q^c$:

$$C_{k+3} = B^2_i \cup B^3_i \cup B^3_i$$

$$R_1(u, x_1), \ldots, R_k(x_{k-1}, v), B^3_i$$

$$Q^c = C_{k+3}(u, b, u, x_1, \ldots, x_{k-1}, v), B^3_i$$

where $R_1(u, x_1), \ldots, R_k(x_{k-1}, v)$ 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 $A_{C_{k+3}}(I)$ for $Q^c$.

6. Return the following set of or-sets:

$$A_Q(I) = \{ (\text{tup}(\pi_0(t_{ij})), \pi_0(\nu_1(I))) \mid t \in A \} \quad A \in A_Q(I) \quad (3)$$

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 $B'(a, b)$ 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 $B'$ has the form $\{(a_1, b_{ij_1}), \ldots, (a_m, b_{ij_m})\}$ for arbitrary choices $j_1 \in [n_1], \ldots, j_m \in [n_m]$. Therefore, the set of frugal repairs of $B'$ is $\alpha(S_{ij})$, where $S_{ij} = \{(a_i, b_{ij}) \mid j = 1, n_i \}$ $i = 1, m$, which is precisely Eq.(2) up to renaming of the tuples by constants. The relation $B^1_i$ decodes each constant $b_{ij}$ by mapping it to the $a$-projection of $t_{ij}$: similarly for $B^2_i$. Clearly, both $B^1_i, B^2_i$ are consistent, because every constant $b_{ij}$ needs to be stored only once. The relation $B^3_i$ 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 $A_1, A_2$ we have $\Pi_k(A_1) \cap \Pi_k(A_2) = \emptyset$.

Step 3 transforms $Q$ into a cycle $C_{k+3}$ plus a chord $B^2_i(u, v)$, by simply replacing the entire subquery $Q_0$ with the single relation $R^*(u, v)$ (which is correct, since $A_{Q_0}(I)$ is the same as the set of repairs of $B'$) plus the decodings $B^1_i(U(u, v))$, $B^2_i(u, v)$: note that we only needed $B^2_i(u, v)$ 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 FrugalChordPath respectively. Finally, the last step converts back $A_Q(I)$ into $A_{C_{k+3}}(I)$ by expanding the constants $b_{ij}$ into the tuples they encode, $t_{ij} = \text{tup}(b_{ij})$. The algorithm has assumed $u \neq v$. If $u = v$ are the same variable, the cycle $C_{k+3}$ is no longer a cycle: in that case, we split $u$ into two variables $u, v$ and add two consistent relations $R^*(u, v), S^*(u, u)$ to the query, and replace the last relation $R_k(x_{k-1}, u)$ of $P$ with $R_k(x_{k-1}, v)$. 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 $M_{Q_0}(I)$ is representable and $A_{Q_0}(I)$ is its compact representation, then $M_Q(I)$ 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 $Q_0, Q_1, \ldots, Q_m$ be 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 FrugalChordPath to compute $A_{Q_i}(I)$. Return $A_{Q_m}(I)$.

**Example 4.12.** Continuing Example 4.9, we will show how to compute $A_{C_2}(I_2)$ where $I_2$ is the instance shown in Figure 4. We write $H_2$ as $H_2 \equiv C_3, P$, where $C_3 = R(x, y), S(y, z), T(z, x)$ and $P = U(y, t), V(t, z)$. We start by computing $C_3$ on $I_2$: one can check that $A_{C_3}(I_2)$ contains exactly two cycles: $(a_1, b_1, c_1)$ and one of $(a_2, b_2, c_2)$ or $(a_2, b_3, c_2)$.
Figure 4: An inconsistent purified instance I₂ for H₂.

\[
\begin{align*}
R(\bar{x}, y) & \quad S(y, z) & \quad T(\bar{z}, x) & \quad U(y, t) & \quad V(\bar{t}, z) \\
(a₁, b₁) & \quad (b₁, c₁) & \quad (c₁, a₁) & \quad (d₁, c₁) \\
(a₂, b₂) & \quad (b₂, c₂) & \quad (c₁, a₂) & \quad (d₂, c₂) \\
(a₂, b₁) & \quad (b₂, c₁) & \quad (c₂, a₂) & \quad (d₂, c₂)
\end{align*}
\]

\[\text{Figure 5: The resulting instance } I' \text{ produced by the inductive step for } H₂.\]

\[
\begin{align*}
B(\bar{a}, b) & \quad B₁(\bar{b}, y) & \quad B₂(\bar{b}, z) & \quad B₃(\bar{z}, b) \\
(\bar{a₁}, b₁c₁) & \quad (b₁, c₁, b₁) & \quad (b₁, c₁, b₁) & \quad (c₁, a₁, b₁) \\
(\bar{a₂}, b₂c₂) & \quad (b₂, c₂, b₂) & \quad (b₂, c₂, b₂) & \quad (c₂, a₂, b₂)
\end{align*}
\]

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 \(R, S \in E'\): \(R \sim S\) if \(R\) and \(S\) are source-equivalent. Consider the equivalence relation defined by \(\sim\) on the set of inconsistent edges \(E'\), and denote \(E'/\sim\) the quotient set and \([R] \in E'/\sim\) the equivalence class for an edge \(R \in E'\). For our example graph \(G[H]\), we have \(R₁ \sim R₃\) (because \(R₁, R₂, R₃\) form a cycle), thus \([R₁] = [R₁, R₃]\). Also \(S_3 \sim T\), hence \(E'/\sim = \{[R₁], [S₃], [T]\}\).

For every \(C \in E'/\sim\), define

\[
C^+ = \bigcup_{R \in C} u_R^+ \quad \text{ and } \quad C^0 = \bigcup_{R \in C} u_R^0.
\]

Similar to how we have defined \(\text{coupled}^+(R), \text{coupled}^0(R)\) for any \(R \in E'\), we define \(\text{coupled}^+(C), \text{coupled}^0(C)\) for \(C \in E'/\sim\):

\[
\text{coupled}^+(C) = \{C' \in E'/\sim \mid \exists R \in C, S \in C': 3P : v_R \leftrightarrow u_S, P \cap C' = \emptyset\}
\]

\[
\text{coupled}^0(C) = \{C' \in E'/\sim \mid \exists R \in C, S \in C': 3P : v_R \leftrightarrow u_S, P \cap C^0 = \emptyset\}
\]

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:

\[
\text{coupled}^+(\{R₁, R₃\}) = \{\{R₁, R₃\}, [S₃]\}
\]

\[
\text{coupled}^0(\{S₃\}) = \{\{S₃\}\}
\]

Moreover, for \(G[H]\), the sets \(\text{coupled}^+, \text{coupled}^0\) coincide for every equivalence class. For \(C₁, C₂ \in E'/\sim\), we define a binary relation \(\ll\): we write \(C₁ \ll C₂\) if there exists \(S \in C₂\) such that \(u_S \in C₁^0\).

**Proposition 5.2.** \(\ll\) is antisymmetric and transitive.

We can now define \(C₁ \ll C₂\) to be such that \(C₁ \ll C₂\) and \(C₂ \ll C₁\). Then, following from Proposition 5.2, \(\ll\) 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) \(C \in E'/\sim\) is a sink if it is a maximal element of \(\ll\).

**Example 5.4.** Since \(u_{R₃} = z \in u_{R₃}^+(= u_{R₃}^0)\), we have \(\{S₃\} \ll \{R₁, R₃\}\). Also, since \(v \in u_{R₃}^0, v \notin u_{R₃}^+\), \{R₁, R₃\} \ll \{T\}.

By the transitivity of \(\ll\), we also obtain that \(\{S₃\} \ll \{T\}\). Hence, \(\{T\}\) is the only sink of the graph \(G[H]\).

**Definition 5.5.** (Separator). A sink \(C \in E'/\sim\) is a separator if for every \(C' \neq C\) such that \(C' \in \text{coupled}^0(C)\), we have that \(C' \ll C\).

In the specific case where \(E'/\sim\) contains a single sink \(C\), since \(\ll\) is a strict partial order, for any \(C' \in E'/\sim\), \(C' \neq C\), we have that \(C' \ll C\) 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 graph \(Q' = R(\bar{x}, y), S(\bar{x}, y), T'(\bar{x}, y)\), which contains two sinks, \{R, S\} and \{T\}. It is easy to see that \{T\} \notin \text{coupled}^0(\{R, S\}) and \{R, S\} \notin \text{coupled}^0(\{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 \(R \in E'(C)\), \(u^+_R \cap u^0_R \subseteq u^0_R\).
Indeed, \( G[q] \) is not f-closed, since \( v_R^0 = \{ y \} \) and \( V_R^0 = \{ z \} \). We will show in Subsection 5.3 that, given a splitable 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 \( I \equiv G \) if \( I \equiv G' \).

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 
\[ C^{\text{FC}} = \text{arg min}_{C \subseteq E} |C| \] is a separator.

In other words, the sink \( C \) with the smallest \( \text{coupled}(C) \) 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, \textsc{RecursiveSplit}, that takes as input an instance \( I \) and a splittable and f-closed graph \( G \) and returns \textsc{True} if \( I \equiv G \), \textsc{False} otherwise. The algorithm is recursive on the number of inconsistent relations, \( |E(G)| \) of \( G \). For the base case \( E(G) = \emptyset \) (all relations are consistent), it is straightforward that \textsc{RecursiveSplit}(\( I, G \)) = \textsc{True} if and only if \( G(I) \) is true.

We next show how to compute \textsc{RecursiveSplit}(\( I, G \)) when \( |E(G)| > 0 \). 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 \( E' \) into a left (\( L \)) and right (\( R \)) set as follows:

\[ L_C = \{ R \in E' : |R| \in \text{coupled}(C) \} \quad \text{and} \quad R_C = E' \setminus L_C \]

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 \textsc{FrugalSCC} to compute the compression \( A_{S_C}(I) \) of \( M_{S_C}(I) \) in polynomial time, since \( S_C \) is a strongly connected graph. Let \( A \) denote the set of all tuples that appear in some or-set of \( A_{S_C}(I) \), and \( B = \Pi_{C_R}(G'(I)) \).

For some \( a \in A \), we say that \( a \) is aligned with \( b \in B \), denoted \( [a, b] \), if there exists a tuple \( t \in G'(I) \) such that \( t[V(S_C)] = a \) and \( t[C_R] = b \). Also, define \( \text{aligned}(b) = \{ a \in A \mid [a, b] \} \). Observe that \( a \) can be aligned with at most one \( b \), since there exists a consistent directed path from every node of \( V(S_C) \) to every node of \( C_R \). Notice also that when \( C_R = \emptyset \) all the tuples in \( A \) are vacuously aligned with the empty tuple (\( \emptyset \)).

For every tuple \( t(b) \in G'(I) \) such that \( t(b)[C_R] = b \). For every tuple \( a \in A \), we now define a subinstance \( I[a] \subseteq I \) such that:

\[
R^{I[a]} = \begin{cases} \{ ([t(b)[u], t(b)[v]) | b : a, b \} & \text{if } R \in R_C, \\ \{ ([t(u), t(v)] | t \in G'(I), d[V(S_C)] = a \} & \text{if } R \in R_C, \\ R \} & \text{otherwise.} \end{cases}
\]

Notice that if some relation \( R \) belongs to \( S_C \), then it must contain exactly one tuple, while if \( uR \) belongs in \( V(S_C) \), then \( R^{I[a]} \) contains exactly one key-group. On the other hand, the relations that do not belong in \( L_C \) contain only one tuple that contributes to \( t(b) \). The first idea behind the construction of subinstances is captured by the following lemma, which shows that certain subinstances are independent in the relations of \( L_C \).

**Lemma 5.8.** Let \( a, a_2 \in A \). The instances \( I[a], I[a_2] \) share no key-groups in any relation \( R \in L_C \) if either of the following two conditions hold:

1. \( a, a_2 \) belong in different or-sets of \( A_{S_C}(I) \).
2. \( a_1 | b_1, a_2 | b_2 \), and \( b_1 \neq b_2 \).

The second key idea is that computing whether \( I[a] \equiv G \) can be reduced to a computation where \( G \) contains strictly less inconsistent relations. Indeed, recall that in \( I[a] \), every relation \( R_i \in C \), \( i = 1, \ldots, m \), contains exactly one key-group, \( R_i(a[u_R], -) \) (and if 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 \( I[a] \equiv G \) iff every new instance is certain for \( G \). The procedure \textsc{Simplify}(\( I[a], G \)) describes the algorithm we just sketched.

**Algorithm 1:** \textsc{Simplify}(\( I[a], G \))

\[ K = \{ (c_1, \ldots, c_m) | I[[a[u_R], c_i]] \} \quad G' \Leftarrow G \]

Let \( G \) where all edges of \( C \) are of consistent type \( G \in K \):

\[ I[a] \equiv (I[a] \setminus \bigcup R_i R_i(a[u_R], -)) \bigcup R_i(a[u_R], c_i) \]

\[ \text{return } (G \in K : \text{RecursiveSplit}(I[a], G')) \]

**Algorithm 2:** \textsc{RecursiveSplit}(\( I, G \))

if \( E(G) = \emptyset \) then return \( G(I) \)

Find a separator \( C \) of \( G \)

\[ B = \Pi_{C_R}(G'(I)) \]

\[ A_{S_C}(I) \Leftarrow \text{FrugalSCC}(I, S_C) \]

for \( b \in B \) do

\[ \forall a \in A \cap \text{aligned}(b) \Rightarrow \text{Simplify}(I[a], G) = \text{True} \]

if \( \forall b \in B \) then \text{Return} \( G \)

\[ G' \Leftarrow G \]

\[ \text{return } \text{RecursiveSplit}(I, G') \]

We can now analyze the algorithm \textsc{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 \( |E(G)| - |L_C| \) inconsistent edges, so by the induction argument it can be computed in polynomial time. Second, the algorithm calls \textsc{Simplify}(\( I[a], G \)) at most \( |A| \) times, and we have shown that each such call can be computed in polynomial time.

We next argue that \textsc{RecursiveSplit} correctly computes whether \( I \equiv G \) or not. We prove in [9]:

**Lemma 5.9.** \( \bigcup \text{aligned}(b) I[a] \equiv G \) if there exists an or-set \( A \in A_{S_C}(I) \) such that for every \( a \in A \cap \text{aligned}(b), I[a] \equiv G \).

Given a repair \( r \) of \( I \) and a repair \( r' \) of \( \bigcup \text{aligned}(b) I[a] \), we define \( \text{merge}(r, r') \) as a new repair \( r_m \) of \( I \) such that for
any key-group $R(q, -)$, if $R \notin L^C$ 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 $r'$ overtake $r$ only in the relations of $L^C$.

**Lemma 5.10.** For any frugal repair $r$ of $I$:

1. If $\bigcup_{a \in \text{alg}(b)} I[a] \neq G$ then $b \notin \bigcup_{a \in \text{alg}(b)} G^i(r)$.
2. If $\bigcup_{a \in \text{alg}(b)} I[a] = G$ then for any repair $r'$ of the instance $\bigcup_{a \in \text{alg}(b)} I[a] = G$, $G(r) = G(\text{merge}^C(r, r'))$.

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

### 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 $\mathcal{M}_C(I) = \mathcal{M}_C(I')$. For this, we exploit the following technical lemma.

**Lemma 5.11.** Let $R \in E^i$ and $v \in u_{R,E}^i \cap v_{R}^i$. Let $P : u_{R,E}^i \cap v_{R}^i$ be the directed path from $u_{R,E}^i$ to $v$ with $e_R$ as its first edge. If there exist $(a, b_1), (a, b_2) \in P(\overline{I})$ 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 $\mathcal{M}_C$ 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,E}^i \cap v_{R}^i) \setminus (u_{R,E}^i \cap v_{R}^i)$$

$P = u_{R,E}^i \cap v_{R}^i$ in $P(\overline{I})$

$T = \Pi_{u_{R,E}^i \cap v_{R}^i}(P(\overline{I}))$

$I_C \leftarrow I_C \cup \{(a, b) \in T \mid \Phi(a, b) \in T \text{ where } b' \neq b\}$

$G_C \leftarrow \{V(G_C), E(G_C) \cup \{(ur, v)\}\}$

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 $\mathcal{M}_C(I) = \mathcal{M}_C(I_C)$.

### 5.4 Proof Sketch of Separator Existence

We sketch here the proof for Theorem 5.7, which states that $C^{sep} = \arg \min_{\text{sep}} C(E^f, \min)$. Recall that we want to show that for any $C \in E^f$, either $C \notin C^{sep}$, or $C \notin C^{sep}$ and $C \notin C^{sep}$. We will show next that it suffices to consider only the sinks $C \in E^f$, and show that for any sink $C \notin C^{sep}$, $C \notin C^{sep}$. Indeed, we show in [9] that for a sink $C$, the set $\text{closed}^* C$ is upward closed: if $C < C^*$ then also $C < C^*$. Note that $\text{closed}^* C$ is not necessarily upward closed for any $C$ that is not a sink.

**Lemma 5.13.** If $C$ is a sink, $\text{closed}^* C$ is upward closed.

Now, suppose that we have shown that for any sink $C \notin C^{sep}$, $C \notin \text{closed}^* C$ and consider any $C' \in E^f$, $C' \notin C$. Then $C < C'$ for some $C' \in E^f$ that is a sink; hence, $C' \notin C^{sep}$. However, since $C^{sep}$ is a sink, we can apply Lemma 5.13 to conclude that $C' \notin C^{sep}$.

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 coinciding: $\text{closed}^* C = \text{closed}^* C$.

**Proposition 5.14.** Let $G$ be a splittable and f-closed graph. For any sink $C \in E^f$, $C^* = C^*$. The second result tells us that for two distinct equivalence classes $C_1, C_2$ where $C_1 \in \text{closed}^* C_2$, coupled$^* C_1 \in \text{closed}^* C_2$.

**Proposition 5.15.** Let $G$ be a splittable and $C_1, C_2 \in E^f$ such that $C_1 < C_2$. Then,

1. Either $C_1 \notin \text{closed}^* C_2$ or $C_2 \notin \text{closed}^* C_1$.
2. If $C_1 \in \text{closed}^* C_2$, coupled$^* C_1 \in \text{coupled}^* C_2$.

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

### 6. THE CONP-COMPLETE CASE

In this section, we prove part (2) of Theorem 3.6: if $G[Q]$ is unsplitable, then Certainty$(Q)$ is conNP-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 $\Phi$ the set of all clauses, $X$ the set of all variables, $X^*$ the set of all literals and $\mathcal{B} = \{T, F\}$ (true, false). Moreover, let us define $\mathcal{P} = \Phi \times X \times X^*$.

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

- $L$ is a lattice.
- $R \cup S \subset X^*$.

**Definition 6.1**
Figure 6: The lattice of the set of labels L.

1. \( L(u_R) = \Phi \) and \( L(v_R) \in \{ T, X, X^* \} \).
2. \( L(u_S) = X \) and \( L(v_S) \in \{ B, X^* \} \).
3. For every \( T \in E'(R, S) \), \( L(v_T) \geq L(v_R) \).
4. \( 3P_R : v_R \leftrightarrow u_S \) such that \( \forall v \in P_R, L(v) \geq X \).
5. \( 3P_S : v_S \leftrightarrow u_R \) such that \( \forall v \in P_S, L(v) \geq B \).

Proposition 6.2. If \( R, S \in E' \) are coupled and \( S \not\in R \), 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. \( R \not\in S \). This implies that we cannot have both \( R \leq S \) and \( S \leq R \), 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 \( \text{Certainty}(Q) \). First, we prove the proposition.

Proof. Since \( S \) is \textit{coupled} \( R \), there exists a path \( P_R : v_R \leftrightarrow u_S \) s.t. \( P_R \cap u_S \cap R = \emptyset \); similarly, there exists a path \( P_S : v_S \leftrightarrow u_R \) s.t. \( P_S \cap u_R \cap S = \emptyset \). 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:

\[ v_R \notin u_R^+ \quad u_S \notin v_S^{-R} \quad v_S \notin u_S^+ \quad u_R \notin v_R^+ \tag{4} \]

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 \( W \):

\[ L_0(v_R) = \Phi, \quad L_0(v_S) = T, \quad L_0(u_S) = X, \quad L_0(v_S) = X^* \]

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

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

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 \( W^{-1}(u_R) \) is either \( \{ u_R \} \) or \( \{ u_R, v_R \} \); indeed \( v_S \notin W^{-1}(u_R) \) because \( S \not\in R \) and \( u_S \notin W^{-1}(u_R) \) by Eq.(4). By definition, either \( L(u_R) = \Phi \) or \( L(u_R) = \Phi \land T = \Phi \); in both cases \( L(u_R) = \Phi \).

\( u_S \): We have \( \{ u_S \} \subseteq W^{-1}(u_S) \subseteq \{ u_S, v_R, u_S \} \), because Eq.(4) implies \( u_S \notin u_R^{-R} \). This implies \( X = L(u_S) \geq L(u_S) \land L(v_R) \land L(v_S) = X \land T \land X^* = X \), hence \( L(u_S) = X \).

\( v_R \): We have \( \{ v_R \} \subseteq W^{-1}(v_R) \subseteq \{ u_S, v_R, v_S \} \), because Eq.(4) implies \( v_R \notin u_R^{-R} \). Therefore, \( T \supseteq L(v_R) \geq X \land T \land X^* = X \), implying \( L(v_R) \in \{ X, X^* \} \).

\( v_S \): We have \( \{ v_S \} \subseteq W^{-1}(v_S) \subseteq \{ u_R, v_R, v_S \} \), because Eq.(4) implies \( v_S \notin u_S^{-S} \). Therefore, \( X^* \supseteq L(v_S) \geq \Phi \land T \land X^* = X \), implying \( L(v_S) \in \{ B, X^* \} \).

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

For (4), let \( P_R \) be the undirected path defined earlier s.t. \( P_R \cap u_S^{-R} = \emptyset \); we also have \( P_R \cap u_R^{-R} = \emptyset \). Let \( v \in P_R \) be any node on this path. Then \( u_R \notin W^{-1}(v) \), which implies that \( W^{-1}(v) \subseteq \{ u_R, u_S, v_S \} \), and therefore \( L(v) \supseteq T \land X \land X^* = X \).

Finally, for (4), let \( P_S \) be the undirected path defined earlier, s.t. \( P_S \cap u_R^{-S} = \emptyset \). For any node \( v \in P_S \) we have \( W^{-1}(v) \subseteq \{ u_R, v_R, v_S \} \), thus \( L(v) \supseteq \Phi \land T \land X^* = B \) \( \Box \).

Next, we show how to use a valid labeling to reduce the \textit{Monotone-3SAT} \( \Phi \) to \textit{Certainty}(Q).

The functions \( f_{L_1, L_2} \). For any pair of sets \( L_1, L_2 \in \mathcal{L} \) such that \( L_1 \supseteq L_2 \), we define a function \( f_{L_1, L_2} : L_1 \rightarrow L_2 \), 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:

\[ f_{\Phi, \emptyset} : \Phi \rightarrow T \text{ if } \Phi \text{ is a positive clause, else } F \]

\[ f_{X^*, X} : f_{X^*, X}(x^*) = x \]

\[ f_{X, B} : f_{X, B}(x^*) = F \]

\[ f_{T, \Phi} : f_{T, \Phi}(\phi(x^*)) = \phi \]

\[ f_{T, X^*} : f_{T, X^*}(\phi, x^*) = \phi(x^*) \]

\[ f_{B, X} : f_{B, X}(x) = x \]

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

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

If \( T \neq R \) or \( T \neq S \), then we know that \( L_1 \geq L_2 \). Define \( T' = \{(a, b) \mid a \in b \text{ or } f_{T_1}(a) \in L_1 \} \). Notice that the first attribute of \( T' \) is a key (because \( f_{T_1, L_1} \) is a function), and therefore \( T' \) always satisfies the key constraint.

If \( T = R \) or \( T = S \), then \( L_1 \nleq L_2 \). In this case we construct \( R' \) and \( S' \) 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:

\[ \Phi, \emptyset : \quad R' = \{(a, b) \mid b \in f_{T_1}(a) \} \quad (b) \]

\[ \Phi, X^* : \quad R' = \{(a, b) \mid a \in f_{T_1}(b) \} \quad (b-f) \]

\[ X, X^* : \quad S' = \{(a, b) \mid a \in f_{T_1}(b) \} \quad (b) \]

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

\(^7\)In a lattice, \( L_1 \) covers \( L_2 \) if \( L_1 \supseteq L_2 \) and there is no \( L_3 \) s.t. \( L_1 \supseteq L_3 \supseteq L_2 \).
Figure 7: A query graph with a $(R, S)$-valid labeling.

Example 6.3. Consider the formula $\phi = \phi_1 \land \phi_2$, where $\phi_1 = (z^\exists \land y^\exists \land z^\exists)$ and $\phi_2 = (z^\exists \land w^\exists \land t^\exists)$. If the inconsistent relation $R$ is labeled with $(\Phi, X)$, it will be populated by the tuples $(\phi_1, y), (\phi_1, z), (\phi_2, w), (\phi_2, t)$. On the other hand, a consistent relation $T \not\models R, S$ that is labeled with $(\Phi, B)$ 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 $R \not\models S$. Also, $u_R = x, v_R = u_S = y$ and $v_S = z$. Since $L^x(x) = \{L_0(x), L_5(x)\} = \{\Phi, L(x)\} = \Phi$. Also, $L^y(y) = \{L(v_S), L_0(u_S)\} = \{\Phi, X^*\}$. For variable $z$, $L^z(z) = \{L(v_S), L_0(u_S)\} = \{\Phi, X^*\}$ and $L(z) = \Phi \land X^* = B$. $L^t(t) = \{L_0(u_S), L_0(v_N), L_0(u_S)\} = \{\Phi, T, X\}$ and hence $L(t) = \Phi \land T \land X = \bot$.

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 $Q_2 = \mathcal{R}(x, y), \mathcal{S}(y, x)$, 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