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
May 22, 2014

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 \( \text{CERTAINTY}(Q) \) 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 \( \text{CERTAINTY}(Q) \) 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.

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 \cite{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 \( \text{CERTAINTY}(Q) \) 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 \textsc{Certainty}(Q) is always in coNP \cite{thesis}. 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 \( Q_1 = R(x,y), S(y,z) \), \textsc{Certainty}(\( Q_1 \)) is not only in P but, since one can show that \textsc{Certainty}(\( Q_1 \)) can be expressed as a first-order query over \( I \) \cite{wise}, it is in \( AC^0 \). On the other hand, for \( Q_2 = R(x,y), S(z,y) \), it has been proved in \cite{wise} that \textsc{Certainty}(\( Q_2 \)) is coNP-complete. Finally, for \( Q_3 = R(x,y), S(y,x) \), one can show \cite{wise} 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 \textsc{Certainty}(Q) remains open. In fact, a long-standing conjecture claims the following dichotomy.

\textbf{Conjecture 1.1.} Given a boolean conjunctive query \( Q \), \textsc{Certainty}(\( Q \)) is either in \( P \)TIME or is coNP-complete.

The progress that has been made towards proving this conjecture has been limited. In particular, Kolaitis and Pema \cite{thesis} have proved a dichotomy into \( P \)TIME and coNP-complete for the case where \( Q \) contains only two atoms and no self-joins (\( i.e. \) every relation name appears once). Wise \cite{wise} has given a necessary and sufficient condition for first-order rewriting for acyclic conjunctive queries without self-joins, and in a recent paper \cite{wise} further classifies several acyclic queries into \( P \)TIME and 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 \( R_1(\bar{a}, b_1), R_2(\bar{a}, b_2) \) such that \( b_1 \neq b_2 \), we can remove the key-groups \( R_1(\bar{a}, -), R_2(\bar{a}, -) \) without loss of generality in order to check for certainty\footnote{\( ^1 \)}. Thus, the conjunction of \( R_1, R_2 \) behaves as a single consistent relation \( R(x,y) \).

Our main result is

\textbf{Theorem 1.2.} For every boolean conjunctive query \( Q \) w/o self-joins consisting only of binary relations where exactly one attribute is the key, there exists a dichotomy of \textsc{Certainty}(\( Q \)) into \( P \)TIME and coNP-complete.

From here we derive:

\textbf{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 \textsc{Certainty}(\( Q \)) into \( P \)TIME and coNP-complete.

\footnote{\( ^1 \)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.}
Except for Appendix A, where we prove Corollary 1.3, the rest of the 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 $R(x, y)$.

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.

**Example 1.5.** Consider the following two queries:

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

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 attribute $\text{attr}(R) = \{A_1, \ldots, A_k\}$, and a key, which is a subset of $\text{attr}(R)$. We write $R(x_1, \ldots, x_m, y_1, \ldots, y_\ell)$ 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 $R(a_1, \ldots, a_m, b_1, \ldots, b_\ell)$. We define a key-group to be all the tuples of a relation with the same key, in notation $R(a_1, \ldots, a_m, -)$.

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 Answering). 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, $\text{CERTAINTY}(Q)$ 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

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

with the following repairs:

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

The repairs will produce the answer sets $Q^f(r_1) = \{(a_1, b_1), (a_2, b_3), (a_3, b_4)\}$, $Q^f(r_2) = \{(a_2, b_3), (a_3, b_4)\}$ and $Q^f(r_2) = \{(a_2, b_3), (a_3, b_5)\}$ respectively. Since $Q^f(r_2) \subseteq Q^f(r_1)$, the repair $r_1$ is not frugal. On the other hand, both $r_2$ and $r_3$ are frugal.
Proposition 2.5. \( I = Q \) 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.

The proposition also implies that we lose no generality if we study only frugal repairs in certain query answering. To check \( I = Q \) 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 [9]. An or-set is a set whose meaning is that one of its elements is selected nondeterministically. Following [9] 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 \( F_Q(I) = \langle r_1, r_2, \ldots \rangle \) 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 \( \langle \{T\} \rangle \), where \( T = \times_{i=1}^k T_i \) is a product of atomic types. For a simple illustration, in Example 2.4 we have \( M_Q(I) = \langle \{(a_2, b_3), (a_3, b_4)\}, \{(a_2, b_3), (a_3, b_5)\} \rangle \), because \( r_2, r_3 \) are the only frugal repairs.

Give a type \( T \), define the following function \( a : \langle \{T\} \rangle \rightarrow \langle \{T\} \rangle \) [9]: \( a(\{A_1, \ldots, A_m\}) = \langle \{x_1, \ldots, x_m\} \mid x_1 \in A_1, \ldots, x_m \in A_m \rangle \). For example, \( a(\{(1, 2), (3, 4)\}) = \langle \{1, 3\}, \{1, 4\}, \{2, 3\}, \{2, 4\} \rangle \) and \( a(\{(1), (1, 2, 3)\}) = \langle \{1\}, \{1, 2\}, \{1, 3\} \rangle \).

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) \( a(S_0) = S \) and (b) for any distinct or-sets \( A, B \in S_0 \), the tuples in \( A \) and \( B \) use distinct constants in all coordinates: \( A_i(A) \cap A_i(B) = \emptyset, \forall i = 1, k \).

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 \( S_0 = \langle \{(a_1, b_1), (a_1, b_2)\}, \{(a_2, b_3)\} \rangle \). 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 \( S \) be an 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 \( S = a(S_0) \), 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 \( O(n^k) \).
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 \( s_1 \subseteq s_2 \) nor \( s_2 \subseteq s_1 \) holds. This implies that, for any instance \( I \), there are two cases. Either (1) \( I \neq Q \); in that case \( \mathcal{M}_Q(I) = \{ \} \) is trivially representable as \( A_Q(I) = \{ \} \); or (2) \( I = Q \), 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 \( A_Q(I) = \{ \langle (a_2, b_3) \rangle, \langle (a_3, b_4) \rangle, \langle (a_3, b_5) \rangle \} \).

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 \( H = R(x, y), S(y, z) \), for which Certainty\((H)\) is in PTIME. However, for the instance \( I' = \{ R(b, b), S(b, c_1), S(b, c_2) \} \), \( \mathcal{M}_H(I') = \langle \{ (a, b, c_1) \}, \{ (a, b, c_2) \} \rangle \) 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 [14], if for every relation \( R \), \( \Pi_{\text{attr}(R)}(Q^f(I)) = R^f \), where \( \Pi_{\text{attr}(R)} \) 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 \( \mathcal{M}_Q(I) = \mathcal{M}_Q(I^p) \).

**Proof.** If \( I \) is not purified, there exists a tuple \( t \) in the key-group \( R(a_1, \ldots, a_m, \ldots) \) such that \( t \notin \Pi_{\text{attr}(R)}(Q^f(I)) \). Then, for any frugal repair \( r \) of \( Q \), no tuple from \( R(a_1, \ldots, a_m, \ldots) \) will contribute to some tuple in \( Q^f(r) \); otherwise, for the repair \( r' = r \setminus R(a_1, \ldots, a_m, \ldots) \cup \{ t \} \) we would have \( Q^f(r') \subseteq Q^f(r) \). Thus, for \( I' = I \setminus R(a_1, \ldots, a_m, \ldots) \), we have \( \mathcal{M}_Q(I) = \mathcal{M}_Q(I') \). We repeat this process until we get a purified instance \( I^p \). \( \square \)

### 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 Appendix A 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 \( R(u, v) \) 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 \( E^c(G) \) (\( E^c(G) \)) the set of all consistent (inconsistent) edges. Thus \( E(G) = E^c(G) \cup E^c(G) \).
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 $\ell \geq 0$. We write $P : x \rightarrow y$ 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 a directed path $P'$ with $e_i = (v_i, v_{i+1})$ or $e_i = (v_{i+1}, v_i)$ for $i = 1, \ldots, \ell$ and $\ell \geq 0$; we write $P : x \leftrightarrow y$ for an undirected path where $v_0 = x$ and $v_\ell = y$ (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 $P : x \rightarrow x$. If $\mathcal{N} \subseteq V(G)$, then $P \cap \mathcal{N}$ denotes the set of vertices in $P$ that occur in $\mathcal{N}$. The notation $x \rightarrow y$ (or $x \sim y$, or $x \leftrightarrow y$) means “there exists a path $P : x \rightarrow y$” (or $P : x \sim y$, or $P : x \leftrightarrow y$).

Finally, since $Q$ uniquely defines $G[H]$ and vice versa, we will often use $G$ to denote 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:

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

The graph $G[H]$ is depicted in [Figure 1](#fig001). 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 $u \rightarrow x$, since the only path from $u$ to $x$ contains inconsistent edges). Moreover, $y \rightarrow v$, 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 $Q = R(x, y), S(y, z), T(z, x)$ 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 consist of all the constants occurring in $I$, and there is an edge $(a, b)$ for every tuple $R^i(a, b)$ 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,

\[
\begin{align*}
\not{u} &= \{ v \in V(G) \mid u \rightarrow v \text{ in } G \} \\
\not{u}^+ &= \{ v \in V(G) \mid u \sim v \text{ in } G \} \\
\not{u}^{+,R} &= \{ v \in V(G) \mid u \sim v \text{ in } G - \{ e_R \} \}
\end{align*}
\]

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

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

Proposition 3.2. If \( R \in E^i \), \( u_R^{\oplus} \subseteq u^{+,R}_R \subseteq u^+ \).

Proof. Let \( v \in u_R^{\oplus} \). Then, there exists a path \( P : u_R \rightarrow v \) 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 \( R \preceq S \) if \( u_S \in u^+_R \). The relation \( \preceq \) is a preorder the set of edges, since it is reflexive and transitive. If \( R \preceq S \) and \( S \preceq R \) 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 an edge \( R \in E^i \), we define the following sets of coupled edges:

\[
\begin{align*}
\text{coupled}(R)^{\oplus} &= [R] \cup \{ S \in E^i \mid \exists P : v_R \leftrightarrow u_S, P \cap u^{\oplus}_R = \emptyset \} \\
\text{coupled}(R)^{+} &= [R] \cup \{ S \in E^i \mid \exists P : v_R \leftrightarrow u_S, P \cap u^{+,R}_R = \emptyset \}
\end{align*}
\]

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

\[
\begin{align*}
x^{\oplus} &= \{ x, v \} \\
x^{+,R_1} &= \{ x, v, w \} \\
x^{+,R_3} &= \{ z, v, w \} \\
x^{+,S} &= \{ u, y, w, x, v, w \} \\
x^{+,T} &= \{ v \}
\end{align*}
\]

Next, we compute \( \text{coupled}(e)^{+} \) for every inconsistent edge \( e \). For example, the set \( \text{coupled}(R_1)^{+} \) 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 \( \text{coupled}^+(R_1) \) contains \( S \); but \( y \) is no longer connected to the source node \( v \) of \( T \), therefore \( \text{coupled}^+(R_1) \) does not contain \( T \). By similar reasoning:

\[
\begin{align*}
\text{coupled}^+(R_1) &= \{R_1, R_3, S\} & \text{coupled}^+(R_1) &= \{R_1, R_3, S\} \\
\text{coupled}^+(R_3) &= \{R_1, R_3, S, T\} & \text{coupled}^+(R_3) &= \{R_3\} \\
\text{coupled}^+(S) &= \{S\} & \text{coupled}^+(S) &= \{S\} \\
\text{coupled}^+(T) &= \{R_1, R_3, S, T\} & \text{coupled}^+(T) &= \{R_1, R_3, S, T\}
\end{align*}
\]

**Proposition 3.2** implies:

**Corollary 3.4.** If \( R \in E_i \), \( \text{coupled}^+(R) \supseteq \text{coupled}^+(R) \).

**Definition 3.5.** Two edges \( R, S \in E_i \) are coupled if \( R \in \text{coupled}^+(S) \) and \( S \in \text{coupled}^+(R) \).

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 \( R_1 \sim R_3 \). Indeed, any other pair is not coupled: \( R_1, S \) are not coupled because \( R_1 \not\in \text{coupled}^+(S) \); \( R_1, T \) are not coupled because \( T \not\in \text{coupled}^+(R_1) \); 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 \( \text{Certainty}(Q) \) is in PTIME. (2) If \( G[Q] \) is unsplittable, then \( \text{Certainty}(Q) \) 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 \( \text{coupled}^+(R) \) is included in the weakly connected component \( Q_i \) that contains \( R \). In this case, **Theorem 3.6** implies that \( \text{Certainty}(Q) \) is coNP-complete iff \( \text{Certainty}(Q_i) \) 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 \( \text{Certainty}(Q) \) is in PTIME. In fact, the first step of our proof is to show that every strongly connected query is in PTIME.

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: \( Q_1 = R(\bar{x}, y), Q_2 = R(\bar{x}, y), S(\bar{z}, y), Q_3 = R(\bar{x}, y), S(\bar{z}, y), T(\bar{z}, y) \), 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 \( \text{coupled}^+(R) = \text{coupled}^+(S) = \{R, S\} \), 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 \text{coupled}^+(S) = \{S, T\}, \text{coupled}^+(T) = \{S, T\}, and therefore \( R, S \) are no longer coupled, nor are \( R, T \): \( Q_3 \) is splittable.

\footnote{The difference between \( Q_2 \) and \( Q_3 \) is that in \( Q_2 \) we have \( z^+, S = \{z\} \), while in \( Q_3 \) we have \( z^+, S = \{x, y, z\} \).}
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 \( \text{CERTAINTY}(Q) \) 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, \( \text{CERTAINTY}(Q) \) is false if and only if \( A_Q(I) = \{\} \); hence, as a corollary of the theorem we obtain:

**Corollary 4.2.** If \( G[Q] \) is strongly connected, \( \text{CERTAINTY}(Q) \) 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:

\[
C_k = R_1(x_1, x_2), R_2(x_2, x_3), \ldots, R_k(x_k, x_1)
\]

Wijsen [14] describes a PTIME algorithm for computing \( \text{CERTAINTY}(C_2) \). We describe here a PTIME algorithm for computing \( A_{C_k}(I) \) (and thus for computing \( \text{CERTAINTY}(C_k) \) 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.

**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 \( v \rightarrow u \), 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 \( M_{C_k}(I) \), 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 $S = \emptyset$.

Example 4.4. We illustrate the algorithm on $C_3 = R(x, y), S(y, z), T(z, x)$. Consider the relations $R, S, T$ of the instance $I$ in Figure 2 and its graph $F_{C_3}(I) = F_1 \cup F_2$ 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 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:

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

It remains to show that the algorithm is correct, and this follows from two lemmas. Recall from Subsection 2.2 that $F_{C_k}(I)$ 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 $F_{C_k}(I)$.

Lemma 4.5. $F_{C_k}(I) = \langle r_1 \cup \ldots \cup r_m \mid r_1 \in F_{C_k}(I_1), \ldots, r_m \in F_{C_k}(I_m)\rangle$

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:

$$M_{C_k}(I) = \begin{cases} \langle \{\} \rangle & \text{if } I \text{ has a cycle of length } > k, \\ \langle \{(a_1, \ldots, a_k)\} \mid a_1, \ldots, a_k \text{ cycle in } F_{C_k}(I)\rangle & \text{otherwise} \end{cases}$$

\footnote{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.}
The lemma says two things. On one hand, if $I$ has a cycle of length $> k$, then $I \not\equiv C_k$. 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 $r$.

**Proof.** To simplify the notation, we denote $F_{C_k}(I)$ by $F(I)$. The lemma follows from the following claim: for any cycle $C$ in $F(I)$, there exists a repair $r_C \subseteq I$ such that $F(r_C)$ contains only $C$ as a cycle. Indeed, if $I$ has some cycle $C$ of length $> k$, then the query $C_k$ is false on $r_C$, proving that $I \not\equiv C_k$; otherwise, for every cycle $C$ of length $k$, $C_k^f(r_C)$ returns only that cycle, and therefore $M_Q(I)$ is an or-set of singleton sets of the form $\{C\}$, for every $k$-cycle $C$. Thus, it remains to prove the claim.

Any subset $r \subseteq I$ represents a subset of edges of $F(I)$. Denote $V(r)$ the set of constants in $r$, and denote $K(r)$ the set of constants that occur in key positions in $r$, i.e. $K(r) = \{a \mid \exists R(a, b) \in r\}$; obviously, $K(r) \subseteq V(r)$. Note that $K(I) = V(I)$ because $I$ is purified.

To prove the claim, fix a cycle $C$, and define a strictly increasing sequence of instances $r_0 \subset r_1 \subset \ldots \subset r_t \subseteq I$ such that for every $i$: (1) $r_i$ is consistent (i.e. it satisfies all key constraints), (2) $V(r_i) = K(r_i)$, (3) $r_i$ contains only $C$ as a cycle. In addition, $r_t$ is a repair [Definition 2.1]. Then, the claim follows by setting $r_C = r_t$.

We start the sequence by setting $r_0 = C$. Clearly $r_0$ satisfies all key constraints and $K(r_0) = V(r_0)$. Now, consider some $r_i$ for $i \geq 0$. If $K(r_i) = V(I)$ then $r_i$ is a repair [Definition 2.1] and we stop, setting $\ell = i$. Otherwise, let $V' = V(I) - K(r_i)$. Since $F(I)$ is strongly connected, there exists an edge from $V'$ to $K(r_i)$, in other words, there exists a tuple $R(a, b)$ such that $a \in V'$ and $b \in K(r_i)$. Define $r_{i+1} = r_i \cup \{R(a, b)\}$. We check the three properties. (1) $r_{i+1}$ is consistent, because $a$ did not occur as a key in $r_i$. (2) $V(r_{i+1}) = V(r_i) \cup \{a\}$ and $K(r_{i+1}) = K(r_i) \cup \{a\}$; by induction we conclude $V(r_{i+1}) = K(r_{i+1})$. (3) Let $C' \neq C$ be a cycle in $r_{i+1}$, then $C'$ must include the new edge $(a, b)$ (since $r_i$ has only $C$ as cycle). Then the preceding edge $(c, a)$ must be in $r_i$, which is a contradiction because $a \not\in K(r_i) = V(r_i)$.

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 $> k$, then $M_{C_k}(I)$ is represented by $A_{C_k}(I) = \{(a_1, \ldots, a_k) | a_1, \ldots, a_k \text{ a cycle in } F_{C_k}(I)\}$; 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 $A_{C_k}(I) = A_{C_k}(I_1) \cup \ldots \cup A_{C_k}(I_m)$. This completes the correctness proof of the algorithm.

We conclude this section with an observation on FO-expressibility. Recall that [13] proves that the $\text{CERTAINTY}(C_2)$ is not first-order (FO)-expressible. The following proposition completes the complexity landscape for cycle queries.

**Proposition 4.7.** For a cycle query $C_k (k > 1)$, $\text{CERTAINTY}(C_k)$ 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 path $P : u \sim v$ 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 $G = G[Q]$ 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 $G_{i-1}$.

**Proof.** We construct $G_i$ inductively. Let $G_0$ be any simple cycle in $G$ (there exists one, since $G$ is strongly connected). For $i \geq 1$, suppose $G_{i-1} \neq G$. Since $G$ is strongly connected, there exists some edge $e_{G_0} = (u, v) \in E(G) \setminus E(G_{i-1})$ such that $u \in V(G_{i-1})$, and there exists a simple path $P'_i$ from $v$ to some node in $G_{i-1}$, $P'_i : v \sim w, w \in V(G_{i-1})$ (if $v \in V(G_{i-1})$, then $P'_i$ is empty and $w = v$). Define $P_i = e_{G_0}, P'_i$ and $G_i = G_{i-1} \cup P_i$. □

**Example 4.9.** Consider the query $H_2 = R(x, y), S(y, z), T(z, x), U(y, t), V(t, z)$. The query admits the following decomposition:

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

Our algorithm for computing $	ext{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 $Q_0, R^c(u, v)$, where $R^c(u, v)$ 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:

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

(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:

**Lemma 4.10.** Let $Q = Q_0, R^c(u, v)$ such that $R^c(u, v)$ is a chord of $Q_0$. If $M_{Q_0}(I)$ is representable and its compression is $A_{Q_0}(I)$, then $\overline{M_{Q_1}} = A_{Q_1}(I)$ is also representable and its compression is given by Eq. (1).

**Proof.** For the one direction, consider a frugal repair $r$ with answer set $Q^f(r)$. We need to show that for any tuple $t \in Q^f(r)$, $t \in O_A$ for some or-set $O_A \in A_{Q_0}(I)$ such that for all tuples $t' \in O_A$, $(t'[u], t'[v]) \in R^c$. Indeed, let $t' \in O_A$ be a tuple for which $(t'[u], t'[v]) \notin R^c$. Then, we can create a repair $r'$ that returns a strictly smaller answer set than $r$ (does not include $t$). For the other direction, let $\{t_1, \ldots, t_m\} \in \alpha(B)$, where $B$ is the R.H.S. of Eq. (1), and $r$ repair such that $Q^f(r) = \{t_1, \ldots, t_m\}$. Then, $r$ must be frugal, otherwise we would have a contradiction on the fact that $A_{Q_0}(I)$ is correctly structured. □

---

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

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

\[
A_{Q_0}(I) = \{ A_1, \ldots, A_m \} \text{ where: } A_i = \langle t_{i1}, t_{i2}, \ldots, t_{in_i} \rangle
\]

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 \( t \cup \pi(b_{ij}) = t_{ij} \) the tuple encoded by \( b_{ij} \).

2. Create four new relations:

\[
B' = \{ (a_i, b_{ij}) | i = 1, m; j = 1, n_i \}
\]

\[
B'_i = \{ (b_{ij}, \pi_u(t_{ij})) | i = 1, m; j = 1, n_i \}
\]

\[
B'_2 = \{ (b_{ij}, \pi_v(t_{ij})) | i = 1, m; j = 1, n_i \}
\]

\[
B'_0 = \{ (\pi_v(t_{ij}), a_i) | i = 1, m; j = 1, n_i \}
\]

\( B' \) is of inconsistent type (hence the superscript “\( \pi \)”), and \( B'_1, B'_2, B'_0 \) 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 \( Q' \) the following queries:

\[
C_{k+3} = B'(a, b), B'_1(b, u), R_1(u, x_1), \ldots, R_k(x_{k-1}, v), B'_0(v, a)
\]

\[
Q' = C_{k+3}'(a, b, u, x_1, \ldots, x_{k-1}, v), B'_2(b, v)
\]

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 \( A_{Q'}(I) \) for \( Q' \).

6. Return the following set of or-sets:

\[
A_Q(I) = \{ \langle (t \cup \pi_v(\pi_b(t)), \pi_{\text{Vars}(p)}(t)) | t \in A \rangle | A \in A_{Q'}(I) \}
\]

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 \( a(A_{Q_0}(I)) \), up to renaming of constants. To see this notice that each repair of \( B' \) 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 \( B' \) is \( \alpha(S_0) \), where \( S_0 = \{ (a_i, b_{ij}) | j = 1, n_i \} | i = 1, m \}, \) which is precisely Eq. \( (2) \) up to renaming of the tuples by constants. The relation \( B'_1 \) decodes each constant \( b_{ij} \) by mapping it to the \( u \)-projection of \( t_{ij} \); similarly for \( B'_2 \). Clearly, both \( B'_1, B'_2 \) are consistent, because every constant \( b_{ij} \) needs to be stored only once. The relation \( B'_0 \) 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 \( \Pi_v(A_{i_1}) \cap \Pi_v(A_{i_2}) = \emptyset \).
Step 3 transforms \( Q \) into a cycle \( C_{k+3} \) plus a chord \( B^c_{k}(b, v) \), by simply replacing the entire subquery \( Q_0 \) with the single relation \( B^i(a, b) \) (which is correct, since \( A_{Q_0}(I) \) is the same as the set of repairs of \( B^i \)) plus the decodings \( B^i_0(b, u), B^i_2(b, v) \): note that we only needed \( B^i_0(u, a) \) 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, \( t_{ij} = t_{up}(b_{ij}) \). 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 \( R^c(y, v), S^c(y, u) \) 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 \( Q_0, Q_1, \ldots, Q_m \) 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. Write \( H_2 \) as \( H_2 = C_3, P \), 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 \( A_{C_3}(I_2) = \{A_1, A_2\} \) where \( A_1 = \langle \langle a_1, b_1, c_1 \rangle \rangle \) and \( A_2 = \langle \langle a_2, b_2, c_2 \rangle \rangle \). Encode the two sets with the new constants \( A_1, A_2 \), and encode the three tuples with three new constants \( \{a_1 b_1 c_1\}, \{a_2 b_2 c_2\}, \{a_2 b_3 c_2\} \). The new relations \( B^i(a, b), B^i_1(b, y), B^i_2(b, z), B^i_3(z, b) \) are shown in Figure 5. Thus, we have to compute the following queries:

\[
C_3 = B^i(a, b), B^i_1(b, y), U_1(y, t), V(t, z), B^i_3(z, a) \\
Q' = C_3^i(a, b, y, t, z), B^i_2(b, z)
\]

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

\[
A_{C_3}(I') = \{\langle (A_1, [a_1 b_1 c_1]), b_1, d, c_1 \rangle, (A_2, [a_2 b_2 c_2]), b_2, d, c_2 \rangle, (A_2, [a_2 b_3 c_2]), b_3, d, c_2 \rangle \}
\]

\[
A_{Q'}(I') = A_{C_3}(I')
\]

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

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

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

\[\text{Every repair of } I_2 \text{ contains exactly two cycles: } (a_1, b_1, c_1) \text{ and one of } (a_2, b_2, c_2) \text{ or } (a_2, b_3, c_2).\]
Also to set up some notation. polynomial time algorithm for splittable graphs. Before we present the formal definition, we need the graph \( G \) self-reducible 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 Separators

In this section, we define the notion of a separator, which is central to the construction of the polynomial time algorithm for splittable graphs. Before we present the formal definition, we 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 \( S \leq [R_1], S \leq T \), hence \( E^i/\sim = \{[R_1], [S], [T]\} \).

For some \( C \in E^i/\sim \), let us define

\[
C^+ \overset{\text{def}}{=} \bigcap_{R \in C} u^{+,R}_R \quad \text{and} \quad C^\oplus \overset{\text{def}}{=} \bigcap_{R \in C} u^{\oplus,R}_R.
\]

Similarly to how we have defined \( \text{coupled}^+(R), \text{coupled}^\oplus(R) \) for edges \( R \in E^i \), we can define \( \text{coupled}^+(C), \text{coupled}^\oplus(C) \) for \( C \in E^i/\sim \):

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

\[
\text{coupled}^\oplus(C) \overset{\text{def}}{=} \{C\} \cup \{C' \in E^i/\sim \mid \exists R \in C, S \in C' : \exists P : v_R \leftrightarrow u_S, P \cap C^\oplus = \emptyset\}
\]
The definitions essentially "lift" the notion of coupling from a single inconsistent edge to an equivalence class. To illustrate with an example, in $G[H]$ we have the following:

$$\text{coupled}^+([R_1, R_3]) = \{[R_1, R_3], \{S\}\} \quad \text{coupled}^+([S]) = \{[S]\} \quad \text{coupled}^+([T]) = \{[R_1, R_3], [T], \{S\}\}$$

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

For $C_1, C_2 \in E^i/\sim$, define the binary relation $\leq^\oplus$: we say that $C_1 \leq^\oplus C_2$ if there exists some $S \in C_2$ such that $u_S \in C_1^\oplus$.

**Proposition 5.2.** The relation $\leq^\oplus$ is antisymmetric and transitive.

*Proof.* To show that $\leq^\oplus$ is antisymmetric, notice that if $C_1 \leq^\oplus C_2$ and $C_2 \leq^\oplus C_1$, $C_1$ and $C_2$ would describe the same equivalence class, and thus $C_1 = C_2$. To show transitivity, assume that $C_1 \leq^\oplus C_2$ and $C_2 \leq^\oplus C_3$. Then, there exists $S \in C_2$ such that $u_S \in C_1^\oplus$ and also $T \in C_3$ such that $u_T \in C_2^\oplus$, and in particular $u_T \in C_1^\oplus$. Thus, $u_T \in C_1^\oplus$ and $C_1 \leq^\oplus C_3$.

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

**Definition 5.3 (Sink).** $C \in E^i/\sim$ is a sink if it is a maximal element of $\leq^\oplus$.

**Example 5.4.** Since $(u_{R_3} = )z \in u^\oplus(= u_S^\oplus)$, we have $\{S\} \leq^\oplus \{R_1, R_3\}$. Also, since $v \in u_{R_1}^\oplus \cap u_{R_3}^\oplus$, $\{R_1, R_3\} \leq^\oplus \{T\}$. By the transitivity of $\leq^\oplus$, we also obtain that $\{S\} \leq^\oplus \{T\}$. 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 $C' \in \text{coupled}^\oplus(C)$, we have that $C' \leq^\oplus C$.

In the specific case where $E^i/\sim$ contains a single sink $C$, since $\leq^\oplus$ is a strict partial order, for any $C' \in E^i/\sim$, $C' \neq C$, we have that $C' \leq^\oplus C$ and thus the single sink $C$ is trivially a separator.

All the equivalence classes of $G[H]$ are separators. Indeed, since $\{S\} \leq^\oplus \{R_1, R_3\} \leq^\oplus \{T\}$, $\{T\}$ is a separator. Also, $\{S\}$ is a separator, since $\{R_1, R_3\}, \{T\} \notin \text{coupled}^\oplus(\{S\})$.

In order to prove the existence of a separator in a graph, it is not a sufficient condition that the graph is splittable. For example, consider the splittable query $Q = R^i(x, y), S^i(x, y), T^i(z, y)$, which contains two sinks, $\{R, S\}$ and $\{T\}$. It is easy to see that $\{T\} \notin \text{coupled}^\oplus(\{R, S\})$, and $\{R, S\} \notin \text{coupled}^\oplus(\{T\})$; 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).** A graph $G$ is f-closed if for every $R \in E^i(G)$, $u_R^\oplus \cap u_R^+ \subseteq u_R^\oplus$.

Indeed, $G[Q]$ is not f-closed, since $u_R^\oplus = \{y\}$, $u_R^+ = \{y\}$ 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 $C^{sep} = \arg\min_{C \in \mathcal{E}} |\text{coupled}^{\oplus}(C)|$ is a separator.

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

We next show how to recursively compute \textsc{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:

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

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 \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 $\mathcal{A}$ denote the set of all tuples that appear in some or-set of $A_{S_C}(I)$, and $B = \Pi_{C^{\oplus}}(G^f(I))$. For some $a \in \mathcal{A}$, we say that $a$ is \textit{aligned} with $b \in B$, denoted $a \parallel b$, if there exists a tuple $t \in G^f(I)$ such that $t[V(S_C)] = a$ and $t[C^{\oplus}] = b$. Also, define $\text{align}(b) = \{a \in \mathcal{A} \mid a \parallel b\}$. 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 $C^{\oplus}$. Notice also that when $C^{\oplus} = \emptyset$, all the tuples in $\mathcal{A}$ are vacuously aligned with the empty tuple $()$.

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

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

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 $R^{I[a]}$ contains exactly one key-group. On the other hand, the relations that do not belong in $LC$ contain only one tuple that contributes to $t^b$.

The first key idea behind the above 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 $a_1, a_2 \in \mathcal{A}$. The instances $I[a_1], I[a_2]$ share no key-groups in any relation $R \in \mathcal{L}^C$ if either of the following two conditions hold:

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

18
Lemma 5.9. \( \text{We next argue that } R \mid \text{at most } \big| \{1, \ldots, p\} \text{ presents the algorithm we sketched.} \)

To show (1), assume for some \( R \in \mathcal{L}^C \) that the key-group \( R(\bar{G}, \bar{\cdot}) \) appears both in \( I[\bar{a}_1], I[\bar{a}_2] \). Since \([R] \in^\circ C\), there exists a path \( P_S : u_R \to u_S \) where \( u_S \in V(S_C) \). It follows from our construction that both \((c, a_1[u_S]), (c, a_2[u_S]) \in \Pi_{R,u_S}(P_S(I))\). But since \( P_S \) contains only consistent relations, it must be that \( a_1[u_S] = a_2[u_S] \), a contradiction to the fact that \( a_1, a_2 \) are value-disjoint (since they belong in different or-sets).

To show (2), let \( R \in \mathcal{L}^C \) and assume that a key-group \( R(\bar{G}, \bar{\cdot}) \) appears both in \( I[\bar{a}_1], I[\bar{a}_2] \). As the argument for (1), there will be some \( u_S \in V(S_C) \) such that \( a_1[u_S] = a_2[u_S] \). Since \( a_1 \parallel b_1 \), there exists a tuple \( t_1 \in G'(I) \) such that \( t_1[u_S] = a_1[u_S] \) and \( t_1[C^\circ] = b_1 \). Similarly, since \( a_2 \parallel b_2 \), there exists a tuple \( t_2 \in G'(I) \) such that \( t_2[u_S] = a_2[u_S] \) and \( t_2[C^\circ] = b_2 \). But now, \( t_1[u_S] = t_2[u_S] \) and \( t_1[C^\circ] \neq t_2[C^\circ] \), which is a contradiction, since each node \( u_S \) for \( S \in C \) has a consistent path \( P : u_S \to v \) for every \( v \in C^\circ \).

The second key idea is that computing whether \( I[\bar{a}] \models G \) can be reduced to a computation where \( G \) contains strictly less inconsistent relations. Indeed, recall that in \( I[\bar{a}] \), every relation \( R_i \in C, i = 1, \ldots, m \), contains exactly one key-group, \( R_i(\bar{a}[u_{R_i}], \bar{\cdot}) \) (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 can be computed in polynomial time by induction. The procedure \( \text{SIMPLIFY}(I[\bar{a}], G) \) formally presents the algorithm we sketched.

Algorithm 1: \( \text{SIMPLIFY}(I[\bar{a}], G) \)

\[
K = \{(c_1, \ldots, c_m) \mid \forall i : R_i(\bar{a}[u_{R_i}], c_i)\} \\
G' \leftarrow G \text{ where all edges of } C \text{ are of consistent type} \\
\forall c \in K: I[\bar{a}]^{(c)} \leftarrow (I[\bar{a}] \setminus \bigcup_{i=1}^m R_i(\bar{a}[u_{R_i}], \bar{\cdot})) \bigcup_{i=1}^m R_i(\bar{a}[u_{R_i}], c_i) \\
\text{return } (\forall c \in K: \text{RECURRENCSPLIT}(I[\bar{a}]^{(c)}, G') = \text{True})
\]

We first argue that the algorithm \( \text{RECURRENCSPLIT} \) runs in polynomial time. First, the final recursive call on \( I, G' \), the graph \( G' \) has \( |E'(G)| - |L|^C < |E'(G)| \) inconsistent edges, so by the induction argument can be computed in polynomial time. Second, the algorithm calls \( \text{SIMPLIFY}(I[\bar{a}], G) \) at most \( |A| \) times, and we have shown that each such call can be computed in polynomial time. We next argue that \( \text{RECURRENCSPLIT} \) correctly computes whether \( I \models G \) or not. We prove first:

**Lemma 5.9.** \( \bigcup_{\text{algn}(b)} I[\bar{a}] \models G \) if and only if there exists an or-set \( A \in As_C(I) \) such that for every \( a \in A \cap \text{algn}(b), I[\bar{a}] \models G \).

**Proof.** For the one direction, assume for the sake of contradiction that for every or-set \( A_i \) (let \( i = 1, \ldots, M \)), there exists a tuple \( a_i \in A_i \) such that \( a_i[\bar{b}] \) and \( I[\bar{a}_i] \not\models G \). Then, there exists a repair \( r^{(a_i)} \subseteq I[\bar{a}_i] \) such that \( G(r^{(a_i)}) \) is false. By [Lemma 5.8(1)], the repairs \( r^{(a_i)} \) will never conflict on their choices for the relations in \( L^C \), and by the construction of \( I[\bar{a}_i] \), all the other relations are consistent and contain the same tuples. Hence, \( r = \bigcup_{\text{algn}(b)} r^{(a_i)} \) is a repair for \( \bigcup_{\text{algn}(b)} I[\bar{a}] \) that cannot satisfy \( G \), a contradiction.

For the inverse direction, assume that there exists an or-set \( A \) such that for every \( a \in A \cap \text{algn}(b), I[\bar{a}] \models G \). If \( r \) is a frugal repair for \( \bigcup_{\text{algn}(b)} I[\bar{a}] \), it must be that \( a \in \prod_{V(S_C)} G^f(r) \) for
For any relation $T$

Lemma 5.11.

includes the choice of $r$

some $a$

We first show that if $r$

Proof.

that $r$

$S$

Lemma 5.10.

For any frugal repair $r$ of $I$:

$G'$

$G$ where all edges in $L^C$ are of consistent type

return $\text{RECURSIVE}\text{SPLIT}(I', G')$

some $a \in A$. But since $I[a] \models G$, any choice that $r$ has made on the key-groups of $L^C$ that appear in $I[a]$ will create a tuple in $G^f(r)$.

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

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

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

Proof. To show (2), we will show that if $G(r)$ is true, then for $r'' = \text{merge}^C(r, r')$, $G(r'')$ is true as well (this suffices to prove (2), since for each repair $r$, there exists a repair $r''$ of $\bigcup_{a \in \text{align}(b)} I[a]$ such that $r = \text{merge}^C(r, r'')$). Let $t \in G^f(r)$. If $t[C^\oplus] \neq b$, then $t \in G^f(r'')$ as well, since the merging of $r, r'$ influences only tuples where $t[C^\oplus] = b$. So now assume that $t[C^\oplus] = b$.

Define the set of vertices $V^C$ to contain all the nodes $v \in V(G)$ for which there exists a path $P : v \leftrightarrow u_S$ for some $S \in R^C$ such that $P \cap C^\oplus = \emptyset$. We show first:

Lemma 5.11. For any relation $T \in E^f(G)$, $T \in L^C$ iff $u_T \in V^C$.

Proof. We first show that if $u_T \in V^C$, $T \notin L^C$. Indeed, if $T \in L^C$, we would have a path $P' : u_T \leftrightarrow v_R$, for some $R \in C$ such that $P' \cap C^\oplus = \emptyset$ and, since $u_T \in V^C$, another path $P : u_T \leftrightarrow u_S$ for some $S \in R^C$ where $P \cap C^\oplus = \emptyset$. But then, the path $P'' = P', P$ connects $v_R$ with $u_S$ and is not intersected by $C^\oplus$, which contradicts the fact that $[S] \notin \text{coupled}^\oplus(C)$.

For the other direction, assume that $u_T \notin V^C$. If $T \notin L^C$, then we would have $u_T \in C^\oplus$, which would imply that $C <^\oplus [T]$. However, this is a contradiction to the fact that $C$ is a separator.

Algorithm 2: $\text{RECURSIVE}\text{SPLIT}(I, G)$

```plaintext
if $E^f(G) = \emptyset$ then return $G(I)$
Find a separator $C$ of $G$
$B \leftarrow \Pi_C G^f(I)$
$A_{SC}(I) \leftarrow \text{FRUGALSCC}(I, S_C)$
for $b \in B$ do
    if $\exists$ or-set $A \in A_{SC}(I)$ s.t. $\forall a \in A \cap \text{align}(b) \Rightarrow \text{SIMPLIFY}(I[a], G) = \text{TRUE}$ then
        $r[b] \leftarrow \text{any repair of } \left( \bigcup_{a \in \text{align}(b)} I[a] \right)$
    else
        $r[b] \leftarrow \emptyset$
end
$orall R \in E(G), R' = \begin{cases} R^I \cap (\bigcup_{b \in B} r[b]) & \text{if } R \in L^C, \\ R^I & \text{otherwise.} \end{cases}$
$G' \leftarrow G$ where all edges in $L^C$ are of consistent type
return $\text{RECURSIVE}\text{SPLIT}(I', G')$
```
For \( r' \), there must exist a tuple \( t \in G^f(r') \) such that \( t[C^\ominus] = b \). Now, define a tuple \( t'' \) as follows: if \( v \in V^{(C)} \), \( t''[v] = t[v] \), otherwise \( t''[v] = t'[v] \). We will show that \( t'' \in G^f(r'') \), which proves that \( G(r'') \) is true. In particular, we will show that for every relation \( T \in E(G) \), the tuple \( s = (t''[u_T], t''[v_T]) \) belongs in \( r'' \). We distinguish four cases:

- \( u_T, v_T \in V^{(C)} \): Then, \( s = (t[u_T], t[v_T]) \). Clearly, \( s \) belongs in \( r \), and since \( T \notin L_C \), \( s \) belongs in \( r'' \) as well.

- \( u_T, v_T \notin V^{(C)} \): Then, \( s = (t'[u_T], t'[v_T]) \). Clearly, \( s \) belongs in \( r' \). If \( T \) is consistent, then it will belong in \( r'' \) as well. If not, then by the above lemma \( T \in L_C \), which implies that the merging will add \( s \) in \( r'' \).

- \( u_T \in V^{(C)}, v_T \notin V^{(C)} \): Since there exists a path from \( u_T \) to some node \( u_S \), where \( S \in R_C \), not intersected by \( C^\ominus \), and no such path from \( v_T \), it must be that \( v_T \notin C^\ominus \). But then, \( t''[v_T] = t'[v_T] = b[v_T] = t[v_T] \). Thus, \( s = (t[u_T], t[v_T]) \), and then the argument goes as in the first item.

- \( u_T \notin V^{(C)}, v_T \in V^{(C)} \): this scenario is not possible. Indeed, similar to the above case, it must be that \( u_T \in C^\ominus \). Now, if \( T \) is consistent, we would have \( v_T \in V^{(C)} \) as well, a contradiction. If \( T \) is inconsistent, then it must be that \( T \in C \) (since \( C \) is a sink); but then, the fact that \( v_T \in V^{(C)} \) implies that \( C \in coupled^\ominus ([R]) \), where \( R \in R_C \), a contradiction.

To show (1), assume that there exists a tuple \( t \in G^f(r) \) such that \( t[C^\ominus] = b \); we will show that this is a contradiction. Since \( \bigcup_{a \in \text{alg}(b)} I[a] \neq G \), there exists a repair \( r'' \) of \( \bigcup_{a \in \text{alg}(b)} I[a] \) such that \( G(r'') \) is false. Let \( r'' = \text{merge}^C(r, r') \); we will show that \( G^f(r'') \subset G^f(r) \), which contradicts the fact that \( r \) is frugal. Notice first that if \( t'' \in G^f(r'') \) and \( t''[C^\ominus] \neq b \), then \( t'' \in G^f(r) \) as well. So now, let \( t'' \in G^f(r'') \) such that \( t''[C^\ominus] = b \). As in the proof for item (1), we construct a tuple \( t' \) such that if \( v \in V^{(C)}, t'[v] = t(b)[v] \), otherwise \( t'[v] = t''[v] \), and using a similar argument one can show that \( t' \in G^f(r') \); however, this is a contradiction, since \( G(r') \) is false.

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 \( A \in A_{Sc}(I) \), 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, \( b \notin \Pi_{C^\ominus} G^f(r) \). 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 is exactly what setting \( r[b] = \emptyset \) achieves. On the other hand, assume that for some \( b \in B \), there exists an or-set \( A \in A_{Sc}(I) \), 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 \( \mathcal{M}_G(I) = \mathcal{M}_{G'}(I') \). For this, we need the following technical lemma.

**Lemma 5.12.** Let \( R \in E^i \) and \( v \in u_R^+ \cap v_R^\ominus \). 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 \( (a, b_1), (a, b_2) \in \Pi_{a \in R}(P^f(I)) \) such that \( b_1 \neq b_2 \), then no frugal repair of \( G \) contains a.
Proof. Suppose for the sake of contradiction that there exists a frugal repair $r$ such that for some tuple $t \in G^f(r)$, $t[u_R] = a$ and let $t[v] = b$. Assume w.l.o.g. that $b \neq b_1$. Let us focus on the key-group $R(a, -)$ and assume that $R(a, c) \in r$. For the tuple $t_p \in P^f(I)$ where $t_p[u_R] = a, t_p[v] = b_1$, it must be that $t_p[v] = c'$ (since $t_p[v_R] = c$, then it would have been that $t_p[v] = b \neq b_1$). Now, construct the new repair $r' = (r \setminus \{R(a, c)\}) \cup \{R(a, c')\}$. We will show that $G^f(r') \subseteq G^f(r)$, which contradicts the frugality of $r$.

First, consider any tuple $t \in G^f(r')$ such that $t[u_R] \neq a$. Then, $t \in G^f(r)$ as well, since $r, r'$ differ only on the choice for the key-group $R(a, -)$. Next, we will show that no tuple $t$ with $t[u_R] = a$ can belong in $G^f(r')$; this completes the proof, since $G^f(r)$ contains such a tuple. Indeed, in this case we would have $t[v] = b$ (since there exists a directed path from $u_R$ to $v$ that does not go through $e_R$, which is the only relation where $r, r'$ differ) and also $t[v] = b_1$ (since now $R(a, c') \in r'$), which is a contradiction.

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

**Algorithm 3: F-CLOSE($I, G$)**

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

end

return $I_C, G_C$

**Proposition 5.13.** Let $I$ be an instance of graph $G$. F-CLOSE returns an instance $I_C$ of an f-closed graph $G_C$ in polynomial time such that $M_C(I) = M_{G_C}(I_C)$.

**Proof.** Note that at an iteration where $v \in (u_R^+ \cap u_R^\oplus) \setminus u_R^\oplus$, we add a consistent edge $(u_R, v)$ (such that $v \in u_R^\oplus$ in the new graph). Since there are at most $|E(G) \cdot |V(G)|$ pairs of inconsistent edges and nodes, the algorithm will terminate after that many steps and return an f-closed graph. It remains to show that if we have the instance $I$ of $G$ at the beginning of the iteration and $I_C, G_C$ at the end, then $M_C(I) = M_{G_C}(I_C)$.

Notice that there exists a 1-1 correspondence between the repairs of $I, I_C$, since the added relation $R^v$ is consistent. Let $r_C$ be a repair of $I_C$ and $r$ the corresponding repair of $I$; we will first show that, if $r$ is frugal, $G^f(r) = G^f_C(r_C)$. Indeed, $G^f(r) \supseteq G^f_C(r_C)$, since $G_C$ contains additional constraints ($R^v$). To show that $G^f(r) \subseteq G^f_C(r_C)$, let $t \in G^f(r)$, where $t[u_R] = a$. Since $r$ is frugal, by **Lemma 5.12** there exists a tuple $R^v(a, b)$, where $t[v] = b$. Hence, $t \in G^f_C(r_C)$. Finally, notice
Hence, \( P \) exists some \( T \) such that \( P \) connects \( H \) and \( S \). Thus, \( u_T \in u_S^\oplus \) and there exists a directed consistent path \( P' : u_S \rightarrow u_T \). Now, the path \( P'' = P, P' \) connects \( v_R \) with \( u_T \). Notice that it is not possible that \( P' \cap C^\oplus \neq \emptyset \), otherwise we would have that \( u_T \in C^\oplus \), which contradicts the fact that \( C \) is a sink. Hence, \( P'' \cap C^\oplus = \emptyset \) and \( C_1 \in coupled^\oplus(C) \).

Now, suppose that we have shown that for any sink \( C \neq C^\text{sep} \), \( C \neq coupled^\oplus(C^\text{sep}) \), and consider any \( C' \in E^\text{i}/\sim \), \( C' \neq C \) that is not a sink. Then \( C' \prec C'' \) for some \( C'' \in E^\text{i}/\sim \) that is a sink; hence, \( C'' \neq coupled^\oplus(C^\text{sep}) \). However, since \( C^\text{sep} \) is a sink, we can apply Lemma 5.14 to conclude that \( C' \neq coupled^\oplus(C^\text{sep}) \).

The bulk of the proof consists of two technical results. The first result tells us that for a sink \( C \), the two types of coupling coincide: \( coupled^+(C) = coupled^\oplus(C) \).

**Proposition 5.15.** Let \( G \) be a splittable and \( f \)-closed graph. For any sink \( C \in E^\text{i}/\sim \), \( C^+ = C^\oplus \).

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 5.16.** Let \( G \) be a splittable graph and \( C_1, C_2 \in E^\text{i}/\sim \) such that \( C_1 \neq C_2 \). Then,

1. Either \( C_1 \neq coupled^+(C_2) \) or \( C_2 \neq coupled^+(C_1) \).
2. If \( C_1 \in coupled^+(C_2) \), then \( coupled^+(C_1) \subset coupled^+(C_2) \).

Now, consider a sink \( C \neq C^\text{sep} \). If \( C \in coupled^+(C^\text{sep}) \), then by Proposition 5.16(2) and Proposition 5.15 it must be that \( coupled^\oplus(C^\text{sep}) = coupled^+(C^\text{sep}) \supset coupled^+(C) = coupled^\oplus(C) \), which contradicts the minimality of \( coupled^\oplus(C^\text{sep}) \), and this proves our main theorem. In the rest of this section, we will present the proofs of Proposition 5.15 and Proposition 5.16.

We start with a proposition that will be used later.

---
Proposition 5.17. If \( C' \in \text{coupled}^+ (C) \) then there exists \( R \in C \) such that for all \( S \in C' \), \( S \in \text{coupled}^+(R) \).

Proof. For any node \( v \in V(G) \), define

\[
\mathcal{L}_C(v) = \{ T \in C \mid v \in u^+_T \}
\]  

(4)

It is easy to see that \( \emptyset \subseteq \mathcal{L}_C(v) \subseteq C \). Moreover, \( v \in C^+ \) if and only if \( \mathcal{L}_C(v) = C \).

Since \( C' \in \text{coupled}^+(C) \), by definition there exists \( R^0 \in C, S \in C' \) and a path \( P^0 : v_{R^0} \leftrightarrow u_S \) such that \( P^0 \cap C^+ = \emptyset \), or equivalently for every \( v \in P^0 \), \( \mathcal{L}_C(v) \subseteq C \). We will show that there exists \( R \in C \) and a path \( P : v_R \leftrightarrow u_S \) such that \( P \cap u^+_R \neq \emptyset \); this proves the proposition, since for any \( S' \in C' \), there exists a directed path \( P' : u_{S'} \rightarrow u_S \) that cannot be cut by \( u^+_R \) (otherwise it would be that \( u_S \in u^+_R \), a contradiction to the fact that \( P \) is not cut by \( u^+_R \)).

If for every \( v \in P^0 \) we have that \( v \notin u^+_R \), then our claim holds trivially for \( R = R^0 \) and \( P = P^0 \). Otherwise, there exists a node \( v \in P^0 \) such that \( \mathcal{L}_C(v) \supseteq \{ R^0 \} \neq \emptyset \). If \( P^0 \) visits in order the nodes \( v_{R^0} = v_1, v_2, \ldots, v_m = u_S \), let \( j \) be the largest index with the property that \( \mathcal{L}_C(v_j) \supseteq \emptyset \). We thus have established that \( \emptyset \subseteq \mathcal{L}_C(v_j) \subseteq C \).

Since there exists an edge \( T \in \mathcal{L}_C(v_j) \), \( v_j \in u^+_T \). Moreover, for any \( U \in C, u_T \in u^+_U \). Consequently, for any \( U \in C, v_j \in u^+_U \). But now, consider an edge \( R \in C \setminus \mathcal{L}_C(v_j) \) (such an edge always exists): since \( v_j \in u^+_R \), \( u_R \) reaches \( v_j \) only by going through the edge \( e_R \) first. Hence, there exists a simple path \( P_j : v_R \rightsquigarrow v_j \) such that \( P_j \cap u^+_R = \emptyset \). Finally, let us construct the path \( P : P_j, v_{j+1}, (v_{j+1}, v_{j+2}), \ldots, (v_m-1, v_m), v_m \) from \( v_R \) to \( u_S \). By our construction, for all \( i = j + 1, \ldots, m \) we have \( \mathcal{L}_C(v_i) = \emptyset \), and thus \( v_i \notin u^+_R \). Hence \( P \cap u^+_R = \emptyset \). \( \square \)

Proof. (of Proposition 5.16)

(1) Assume for the sake of contradiction that \( C_1 \in \text{coupled}^+ (C_2) \) and \( C_2 \in \text{coupled}^+ (C_1) \). Then, from Proposition 5.17, there exists some \( S_0 \in C_2 \) such that for all \( R \in C_1, R \in \text{coupled}^+(S_0) \) and some \( R_0 \in C_1 \) such that for all \( S \in C_2 \) we have \( S \in \text{coupled}^+(R_0) \). In particular, \( R_0 \in \text{coupled}^+(S_0) \) and \( R_0 \in \text{coupled}^+(S) \). But then, \( R_0, R_0, S_0 \) would be an unsplittable pair that are not source-equivalent, a contradiction.

(2) From property (1), we obtain \( C_2 \notin \text{coupled}^+ (C_1) \). Since \( C_2 \in \text{coupled}^+ (C_2) \), it suffices to show that \( \text{coupled}^+(C_1) \subseteq \text{coupled}^+ (C_2) \). Indeed, let \( C \in \text{coupled}^+ (C_1) \), where \( C \neq C_1, C_2 \) (otherwise the claim is trivial).

Since \( C \in \text{coupled}^+ (C_1) \), by applying Proposition 5.17, there exists edges \( T \in C, R \in C_1 \) and a path \( P_{TR} : u_T \leftrightarrow v_R \) such that \( P_{TR} \cap u^+_R = \emptyset \). Additionally, since \( C_1 \in \text{coupled}^+ (C_2) \), by applying Proposition 5.17 we obtain that there exists \( S \in C_2 \) such that for every \( R' \in C_1, R' \in \text{coupled}^+(S) : \) in particular, \( R \in \text{coupled}^+(R) \). Thus, there exists a path \( P_{RS} : u_R \leftrightarrow v_S \) such that \( P_{RS} \neq u^+_S \) (see Figure 6).

Construct now the path \( P^+ = P_{TR}, e_R, P_{RS} \), which is of the form \( P^+ : v_S \leftrightarrow u_T \). We will show that \( P^+ \cap C_2^+ = \emptyset \), which proves that \( C \in \text{coupled}^+ (C_2) \).

Suppose not; then, \( P^+ \cap u^+_S \neq \emptyset \). Since the nodes of \( P_{RS} \) do not intersect with \( u^+_S \), there must exist a node \( v \in P_{TR} \cap u^+_S \), which in turn implies the existence of a directed path \( P_S : u_S \rightsquigarrow v \) that does not contain the edge \( e_S \). If \( P_{RD} \) denotes the fragment of the path \( P_{RT} \) from node \( v_R \) to node \( v \), construct the path \( P^0 = P_{RD}, P_S \) from \( v_R \) to \( u_S \). However, the fact that \( R \in C_1, S \in C_2 \) and \( C_2 \notin \text{coupled}^+ (C_1) \) implies that \( P^0 \cap C_1^+ \neq \emptyset \), and consequently \( P^0 \cap u^+_R \neq \emptyset \). But then, since
Let $v \in u^+_R$, a contradiction to the fact that the path $P_{TR}$ does not intersect with $u^+_R$.

Proof. (of Proposition 5.15) Assume that $v \in C^+$; we will then show that $v \in C^\oplus$ as well. Let $A$ be the unique SCC that contains the edges in $C$. We will first need the following lemma.

**Lemma 5.18.** Let $v \in C^+$ and $v \in w^\oplus$ for some $w \in V(A)$. Then, $v \in C^\oplus$.

**Proof.** We will show that for any $u \in V(A)$, $v \in u^\oplus$ using induction on the distance between $u, w$, denoted $d(u, w)$. For the basis of the induction, where distance is zero, we have that $v \in w^\oplus$ and $d(w, w) = 0$.

For the induction step, consider some node $u$ with $d(u, w) = d + 1$. Then, there exists an edge $e_T$ such that $u = u_T$ and $d(v_T, w) = d$. By the induction hypothesis, $v \in v^T_T$. If $T$ is consistent relation, then trivially $v \in u^\oplus_T$. Otherwise, $T \in C$, and since $v \in C^+$, $v \in u^+_T$. Since $G$ is f-closed, this implies that $v \in u^T_T$.

We now distinguish two cases for some $v \in C^+$. If $v \in V(A)$, then $v \in v^\oplus$ and thus by Lemma 5.18, $v \in C^\oplus$. Otherwise, $v \in V(G) \setminus V(A)$. Since $v \in C^+$, there exists a directed path $P : w \leadsto v$ such that $P \cap V(A) = \{w\}$. Let $P$ visit in sequence the nodes $w = v_0, v_1, \ldots, v_m = v$ and notice that if $i < j$, $L_C(v_i) \subseteq L_C(v_j)$ ($L_C$ as defined in [4]). Since $L_C(v_m) = C$, let $k$ be the first index such that $L_C(v_k) = C$. We will next show that $v_k \in w^\oplus$, which implies that $v_k \in C^\oplus$. Since $C$ is a sink all the edges $(v_i, v_{i+1})$ for $i = k, \ldots, m - 1$ must be consistent and thus $v = v_m \in C^\oplus$ as well.

By the choice of $v_k$, there exists some $S \in L_C(v_k)$ such that for any $i < k$, $S \notin L_C(v_i)$. Additionally, since $S \in L_C(v_k)$, there exists a path $P' : u_S \leadsto v_k$. Let $w'$ be the last node of the path $P'$ inside $A$; we know that $w \neq w'$, since $S \notin L_C(w)$. Finally, let $P''_{w'}$ be the part of the path $P'$ from $w'$ to the first node $v_k$. The important observation is no node of $P''_{w'}$ will be in the same SCC as nodes $v_1, \ldots, v_{k-1}$, since otherwise $S \in L_C(v_i)$ for some $i < k$. So, now we can create the following 2 paths from $w$ to $v_k$: the first path $P_1$ follows $P$ from $w$ to $v_k$, while the second path $P_2$ follows the simple path inside $A$ from $w$ to $w'$ and then $P''_{w'}$. By our previous argument, for any $w \in P_2 \setminus \{w, v_k\}$, $w$ does not belong in the same SCC with any of the nodes in $P_1 \setminus \{w, v_k\}$. We can now apply the following lemma to conclude that $v_k \in w^\oplus$.  

---

**Figure 6:** The setting for the proof of Proposition 5.16 part (2).
Lemma 5.19. Let G be a f-closed and splittable graph. If for u, v ∈ V(G) there exist two directed simple paths P_A, P_B : u ⇝ v such that no w ∈ A \ {u, v}, w ∈ B \ {u, v} are in the same SCC, then v ∈ u⁺.

Proof. Let P_A visit in order the nodes u = w₁^A, ..., w_m^A = v and similarly P_B the nodes u = w₁^B, ..., w_m^B = v. We will show that for any pair w_i^A, w_j^B, v ∈ w_i^A ∪ w_j^B. This proves the lemma, because we can choose i = j = 1. Suppose the claim does not hold, and consider the pair w_i^A, w_j^B such that v ≠ w_i^A ∪ w_j^B and i + j is maximum. First, note none of these nodes is v, otherwise v ∈ w_i^A ∪ w_j^B trivially. Next, consider any node w_k^A, for k > i. Then, it must be that v = w_k^A, since the pair w_k^A, w_j^B has k + j > i + j and v ≠ w_j^B. Similarly for any node w_k^B with k > j, v ∈ w_k^B.

Hence, both edges e_R = (w_j^A, w_j^B+1) and e_S = (w_j^B, w_j^B+1) must be inconsistent. So, R, S ∈ E^i and v ∈ V^i, v ∈ V^j. Moreover, if u_R = u then v ∈ u_j^B as well, which implies that v ∈ u_j^B, a contradiction (similarly, u_S ≠ u). By our assumption of the path structure, u_R, u_S do not belong in the same SCC and thus R ∪ S.

Finally, let P_RS be the path that visits in order the nodes v_R = w_i+1^A, ..., w_m^A = v = w_m^B, ..., w_l^B = u_S and symmetrically P_SR the path that visits v_S, ..., u_R. Since G is splittable and R ∪ S, either P_RS ∩ u_j^B ∩ S or P_SR ∩ u_j^B ∩ S = ∅. Assume that w ∈ P_RS and P_RS ∩ u_j^B ∩ S. Since every node in P_RS has directed path to v that does not go through e_R, this further implies that v ∈ u_j^B. Additionally, we have already shown that v ∈ V^j. Since G is f-closed, this immediately implies that v ∈ u_j^B, which is a contradiction to the existence of the pair.

This concludes the proof.

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 a NP-complete problem [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 B = {true, false}. Moreover, let us define T = Φ × B = {(φ, x) | x ∈ φ, φ ∈ Φ} and ⊥ = {()}. We order the set L = {⊥, B, X, Φ, X*, T} as shown in Figure 1. and T are the minimal and maximal elements, and B ⊆ X, X* and B ⊆ X* are minimal and maximal elements, respectively. The reader may check that L is a lattice. For example, Φ ∧ X* = B and B ∨ X = X*.

Definition 6.1 (Valid Labeling). Let R, S ∈ E^i. A labeling L : V(G) → L is (R, S)-valid if the following conditions hold:

1. L(u_R) = Φ and L(v_R) ∈ {⊤, X, X*}.
2. L(u_S) = X and L(v_S) ∈ {B, X*}.
3. For every T ∈ E^i \ {R, S}, L(u_T) ⊲ L(v_T).
4. ∃R : v_R ⇝ u_S such that ∀v ∈ P_R, L(v) ⊳ X.
5. ∃S : v_S ⇝ u_R such that ∀v ∈ P_S, L(v) ⊳ B.
Proposition 6.2. If $R, S \in E$ are coupled and $S \nleq R$, 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. $R \nsubseteq S$. This implies that we cannot have both $R \subseteq S$ and $S \subseteq R$, 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 $S \in \text{coupled}^+(R)$, there exists a path $P_R : v_R \leftrightarrow u_S$ s.t. $P_R \cap u_R^R = \emptyset$; similarly, there exists a path $P_S : v_S \leftrightarrow u_R$ s.t. $P_S \cap u_S^+ = \emptyset$. 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:

\begin{equation}
 u_R \notin u_R^R \\
 v_R \notin u_S^+ \\
 v_S \notin u_S^+ \\
 u_S \notin u_R^R
\end{equation}

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 $W$:

\begin{align*}
 L_0(u_R) &= \Phi, \\
 L_0(v_R) &= \top, \\
 L_0(u_S) &= X, \\
 L_0(v_S) &= X^*
\end{align*}

For every node $v \in V(G)$, denote $W^{-1}(v) = \{w \mid w \in W, v \in w^{+,R,S}\}$, 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, $W^{-1}(v)$ 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 $S \nsubseteq R$, and $u_S \notin W^{-1}(u_R)$ by Eq.(5). By definition, either $L(u_R) = \Phi$ or $L(u_R) = \Phi \land \top = \Phi$; in both cases $L(u_R) = \Phi$.

$u_S$: We have $\{u_S\} \subseteq W^{-1}(u_S) \subseteq \{u_S, v_R, v_S\}$, because Eq.(5) implies $u_S \notin u_R^{+,R,S}$. This implies $X = L_0(u_S) \geq L(u_S) \geq L_0(v_R) \land L_0(v_S) = X \land X^* = X$, 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.(5) implies $v_R \notin u_R^{+,R,S}$. Therefore, $\top \geq L(v_R) \geq X \land X^* = X$, implying $L(v_R) \in \{X, X^*, \top\}$.

Figure 7: The lattice of the set of labels $L$. 

We prove:

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 $W$:

\begin{align*}
 L_0(u_R) &= \Phi, \\
 L_0(v_R) &= \top, \\
 L_0(u_S) &= X, \\
 L_0(v_S) &= X^*
\end{align*}

For every node $v \in V(G)$, denote $W^{-1}(v) = \{w \mid w \in W, v \in w^{+,R,S}\}$, 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, $W^{-1}(v)$ 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 $S \nsubseteq R$, and $u_S \notin W^{-1}(u_R)$ by Eq.(5). By definition, either $L(u_R) = \Phi$ or $L(u_R) = \Phi \land \top = \Phi$; in both cases $L(u_R) = \Phi$.

$u_S$: We have $\{u_S\} \subseteq W^{-1}(u_S) \subseteq \{u_S, v_R, v_S\}$, because Eq.(5) implies $u_S \notin u_R^{+,R,S}$. This implies $X = L_0(u_S) \geq L(u_S) \geq L_0(v_R) \land L_0(v_S) = X \land X^* = X$, 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.(5) implies $v_R \notin u_R^{+,R,S}$. Therefore, $\top \geq L(v_R) \geq X \land X^* = X$, 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.(5) implies \(v_S \notin \mathcal{u}_S^{+R,S}\). Therefore, \(X^* \supseteq L(v_S) \supseteq \Phi \land \top \land X^* = \mathbb{B}\), implying \(L(u_S) \in \{\mathbb{B}, X^*\}\).

To show property (5), 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 \(L(u_T) \supseteq L(v_T)\).

For (4), let \(P_R\) be the undirected path defined earlier s.t. \(P_R \cap u_R^{+R} = \emptyset\); we also have \(P_R \cap \mathcal{u}_R^{+R,S} = \emptyset\). 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 \(L(v) \supseteq \top \land X \land X^* = X\).

Finally, for (4), let \(P_S\) be the undirected path defined earlier, s.t. \(P_S \cap \mathcal{u}_S^{+S} = \emptyset\). As before, for any node \(v \in P_S\) we have \(\mathcal{W}^{-1}(v) \subseteq \{u_R, v_R, v_S\}\), and therefore \(L(v) \supseteq \Phi \land \top \land X^* = \mathbb{B}\). □

Next, we show how to use a valid labeling to reduce the \textsc{Monotone-3Sat} \(\Phi\) to \textsc{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 \(f_{L_1,L_2} : L_1 \to L_2\), 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:

\[
\begin{align*}
(\Phi, \mathbb{B}) : f_{\Phi, \mathbb{B}}(\phi) &= T \text{ if } \phi \text{ is a positive clause, else } F \\
(X^*, X) : f_{X^*, X}(x^+) &= f_{X^*, X}(x^-) = x \\
(X^*, \mathbb{B}) : f_{X^*, \mathbb{B}}(x^+) &= T \text{ and } f_{X^*, \mathbb{B}}(x^-) = F \\
(\top, \Phi) : f_{\top, \Phi}(x, \phi) &= \phi \\
(\top, X^*) : f_{\top, X^*}(x, \phi) &= x^* \\
(\mathbb{B}, \bot), (X, \bot) : f_{\mathbb{B}, \bot}(b) &= f_{\mathbb{B}, \bot}(x) = ()
\end{align*}
\]

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

**Instance Construction.** Now we define the database instance \(I\), by defining a binary relation \(T^I\) for every relation name \(T\). Let \(L_1 = L(u_T), L_2 = L(v_T)\) be the labels of the source and target node of \(e_T\). We distinguish two cases, depending on whether \(T\) is \(R\) or not.

If \(T \neq R\), \(T \neq S\), then we know that \(L_1 \supseteq L_2\). 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 we ensure that \(T^I\) always satisfies the key constraint, regardless of whether the type of \(T\) was consistent or inconsistent.

If \(T = R\) or \(T = S\), then \(L_1 \not\supseteq L_2\). 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” in the lattice, or going “back and forth”, depending on the particular combination of \(L_1, L_2\) given by \textbf{Definition 6.1}:

\[
\begin{align*}
(\Phi, \top) : R^I &= \{(a, b) \mid b \in f_{\top, \Phi}^{-1}(a)\} \text{ (back)} \\
(\Phi, X^*) : R^I &= \{(a, b) \mid \exists c \in f_{\top, \Phi}^{-1}(a) : f_{\top, X^*}(c) = b\} \text{ (back-and-forth)} \\
(\Phi, X) : R^I &= \{(a, b) \mid \exists c \in f_{\top, \Phi}^{-1}(a) : f_{\top, X}(c) = b\} \text{ (back)} \\
(X, X^*) : S^I &= \{(a, b) \mid b \in f_{X^*, X}^{-1}(a)\} \text{ (back)} \\
(X, \mathbb{B}) : S^I &= \{(a, b) \mid \exists c \in f_{X^*, X}^{-1}(a) : f_{X^*, \mathbb{B}}(c) = b\} \text{ (back-and-forth)}
\end{align*}
\]

\(^7\)In a lattice, \(L_1\) covers \(L_2\) if \(L_1 > L_2\) and there is no \(L_3\) s.t. \(L_1 > L_3 > L_2\).
Notice that in all cases $R^I$ and $S^I$ are inconsistent. In the first case, a repair of $R^I$ chooses for each clause $\phi \in \Phi$ a value $(\phi, b)$ with $b \in B$; in the second case, a repair of $R^I$ chooses for each clause $\phi$, a literal $x^* \in \phi$, while in the third case a repair chooses for each clause $\phi$ a variable $x$ in that clause.

**Example 6.3.** Consider the formula $Y = \phi_1 \land \phi_2$, where $\phi_1 = (x^+ \lor y^+ \lor z^+)$ and $\phi_2 = (z^- \lor w^- \lor t^-)$. If the inconsistent relation $R$ is labeled with $(\Phi, X)$, 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 $(\Phi, B)$ will contain the tuples $(\phi_1, T), (\phi_2, F)$.

Thus, given a valid labeling we can create a database instance using the construction we just presented. We prove:

**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, $I \not= Q$ if and only if $M$ has a satisfying assignment.

**Proof.** First, note that the valid labeling guarantees that, if $T \neq R, S$, then $T$ will be a consistent relation in the instance $I$. On other hand, the relations $R$ and $S$ will be inconsistent.

Consider a satisfying assignment for $M$, where $v(x)$ denotes the assigned value (true or false) for variable $x$. We will construct a repair $r$ that does not satisfy $Q$. Since the assignment satisfies the formula, for every clause $\phi$ there exists a literal $x^*$ that evaluates to true. Then, for the relation $R$, $r$ includes the tuple $(\phi, x^*)$ (if $e_R$ has labels $(\Phi, X^*)$) or $(\phi, x)$ (if $(\Phi, X)$) or $(\phi, (\phi, x^*))$ (if $(\Phi, T)$). As for the relation $S$, we have two cases. If the labels are $(X, A)$, $r$ includes $(x, F)$ when $v(x) = T$, and $(x, T)$ when $v(x) = F$. Similarly, for $(X, X^*)$, if $v(x) = T$, $r$ includes the tuple $(x, x^-)$, otherwise if $v(x) = F$, $r$ includes $(x, x^+)$. It remains to show that $Q(r)$ evaluates to false. For the sake of contradiction, assume that $Q(r)$ is true and consider a tuple $t \in Q^f(r)$. Let $t[u_R] = \phi$ and assume w.l.o.g. that it is a positive clause. Then, $t[v_R] \in \{x, x^+, (\phi, x^+)\}$, for a variable $x$ with assignment $v(x) = T$. Note that there must be a path from $v_R$ to $u_S$ such that every label in the path has a consistent mapping to $X$. Hence, $t[u_S] = x$, which implies that $t[v_S] \in \{F, x^-\}$ by our construction of $I$. But this is a contradiction, since there exists a path from $v_S$ to $u_R$ $(t[v_S] \in \{F, x^-\})$ to $u_R$ $(t[u_R] = \phi$ is a positive clause), where each label has a consistent mapping to $A = \{T, F\}$.

For the inverse direction, assume that $I$ has a repair such that $Q(r)$ is false. We construct an assignment for the variables in $M$ as follows: if the repair $r$ contains a tuple $(x, T)$ (or $(x, x^+)$) in relation $S$, we let $v(x) = F$; otherwise, $v(x) = T$. Now, consider a positive (w.l.o.g.) clause $\phi$ of the instance $M$. Assume that $r$ contains in $R$ a tuple $(\phi, x)$ (or $(\phi, x^+)$ or $(\phi, (\phi, x^+))$). Using similar arguments as before, one can see that $r$ cannot include $(x, T)$ (or $(x, x^+)$); otherwise, $Q(r)$ will evaluate to true. Hence, $v(x) = T$ and clause $\phi$ will be satisfied. \[\square\]

**Example 6.5.** As an example of the labeling construction, consider the query of Figure 8. Notice that $R \subseteq S$. 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 $L(y) = \top \land X = X$. For variable $z$, $L^+(z) = \{L_0(v_S), L_0(u_R)\} = \{\Phi, X^*\}$ and $L(z) = \Phi \land X^* = B$. Finally, $L^+(t) = \{L_0(u_R), L_0(v_R), L_0(u_S)\} = \{\Phi, T, X\}$ and hence $L(t) = \Phi \land T \land X = \bot$. 29
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 \textsc{certainty}(Q) is first-order expressible, \textit{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 \textsc{cf}orest, 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 [11, 13], 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 [12] gave a polynomial time algorithm for the query \( Q_2 = R(x,y), S(y,x) \), 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 \textsc{certainty}(Q) when the query has only two atoms and no self-joins into polynomial time and coNP-complete. Finally, Wijsen [14] recently classified several acyclic queries into \textsc{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 [10] showed that this problem admits a dichotomy in \textsc{ptime} and \#\textsc{ptime}-complete for conjunctive queries without self-joins.

Finally, we should mention that the problem of consistent query answering is closely related to probabilistic databases, and in particular \textit{disjoint-independent} probabilistic databases [4]. Wijsen in [14] discusses the precise connection between the complexity of evaluating a query \( Q \) on a probabilistic database and \textsc{certainty}(Q).

8 Conclusion

In this paper, we make significant progress towards proving a dichotomy on the complexity of \textsc{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 such a dichotomy exists for general conjunctive queries, even in the simpler case where there are no self-joins.
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 V. Vianu and C. H. Papadimitriou, editors, 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 L. Libkin, editor, 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 K. Böhm, C. S. Jensen, L. M. Haas, M. L. Kersten, P.-Å. Larson, and B. C. Ooi, editors, 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] L. Libkin and L. Wong. Semantic representations and query languages for or-sets. In C. Beeri, editor, PODS, pages 37–48. ACM Press, 1993.

[10] D. Maslowski and J. Wijsen. On counting database repairs. In G. H. L. Fletcher and S. Staworko, editors, LID, pages 15–22. ACM, 2011.

[11] J. Wijsen. Consistent query answering under primary keys: a characterization of tractable queries. In R. Fagin, editor, ICDT, volume 361 of ACM International Conference Proceeding Series, pages 42–52. ACM, 2009.

[12] J. Wijsen. On the first-order expressibility of computing certain answers to conjunctive queries over uncertain databases. In J. Paredaens and D. V. Gucht, editors, PODS, pages 179–190. ACM, 2010.

[13] J. Wijsen. A remark on the complexity of consistent conjunctive query answering under primary key violations. Inf. Process. Lett., 110(21):950–955, 2010.

[14] J. Wijsen. Charting the tractability frontier of certain conjunctive query answering. In R. Hull and W. Fan, editors, PODS, pages 189–200. ACM, 2013.
A Simplifying the Structure

In this section, we show how to transform any query that consists of atoms where the key is either a single attribute or all attributes to a query which we call graph-representable.

**Definition A.1.** A boolean connected CQ $Q$ is graph-representable if it is w/o self-joins, w/o constants, w/o duplicate variables in a single atom, and further contains only binary atoms where each atom has exactly one attribute as key.

First, note that we can assume w.l.o.g. that the hypergraph for $Q$ is connected; otherwise, we can solve $\text{CERTAINTY}(Q)$ for each of the connected components and decide that $Q$ is certain if and only if every component is certain.

We write that $\text{CERTAINTY}(Q) \equiv \text{CERTAINTY}(Q')$ if there exists an FO-expressible reduction from $\text{CERTAINTY}(Q)$ to $\text{CERTAINTY}(Q')$ and vice versa.

**Theorem A.2.** Let $Q$ be a connected boolean CQ without self-joins, where the key for each atom is either a single attribute or all attributes. Then, there exists a graph-representable query $Q'$ such that $\text{CERTAINTY}(Q) \equiv \text{CERTAINTY}(Q')$.

The FO-expressible reduction described in the above theorem can be decomposed in a sequence of simpler steps, which we describe next, thus proving Theorem A.2.

In the case where a query $Q$ contains an atom $R$ with constants and/or variables that appear twice, we can reduce the query $Q$ to a query $Q'$ where $R$ is replaced by an atom $R'$ that contains only variables that appear exactly once.

**Proposition A.3.** Let $Q$ be a CQ that contains an atom $R$. Let $Q'$ be the query where we have replaced $R$ with an atom $R'$ without constants, and where every variable appears exactly once. Then, $\text{CERTAINTY}(Q) \equiv \text{CERTAINTY}(Q')$.

We can further simplify the query structure by removing unary relations.

**Proposition A.4.** Let $Q$ be a connected CQ and $Q'$ be the query derived from $Q$ by removing all occurrences of unary atoms. Then, $\text{CERTAINTY}(Q) \equiv \text{CERTAINTY}(Q')$.

**Proof.** Notice that every unary relation is consistent by definition, since the only attribute is the primary key. Let $U(x)$ be such a unary relation in $Q$ and consider another appearances of variable $x$ in the query. Consider any atom that contains $x$ as a variable. Then, by Lemma A.7, we can remove from this atom any key-group such that $x$ assumes a value $a$ and $a \notin U^P$, since no frugal repair will contain $a$ in the answer set. After this processing of $I$, $U$ plays no role in whether a repair satisfies the query and hence can be removed to obtain a query $Q_{-U}$ without the atom $U(x)$. Notice also that the processing is FO-expressible. For the inverse reduction, we can add a unary relation $U(x)$ to $Q_{-U}$ such that $U = \Pi_x(Q_{-U}(I))$ (since $Q$ is connected, $Q_{-U}$ always contains an appearance of variable $x$).

Next, we show how to handle the atoms where the primary key consists of all the attributes: such an example could be $R(x, y)$ or $S(x, y, z)$. In the general setting, we are given an atom of the form $R(x_1, \ldots, x_k)$. Observe that the relation $R$ will be always consistent, since it is not possible to have any key violations.


Proposition A.5. Let \( Q \) be a connected CQ containing \( R(x_1, \ldots, x_k) \), and let \( Q' \) be the query obtained by replacing \( R \) in \( Q \) with \( k \) new consistent relations \( R_1(\overline{x}, x_1), \ldots, R_k(\overline{x}, x_k) \) (\( x \) is a new variable that does not appear in \( Q \)). Then, \( \text{CERTAINTY}(Q) \overset{\text{FO}}{\sim} \text{CERTAINTY}(Q') \).

Proof. To reduce \( \text{CERTAINTY}(Q') \) to \( \text{CERTAINTY}(Q) \), we simply compute \( R(x_1, \ldots, x_k) \) as the natural join of the relations \( R_1, \ldots, R_k \) on the common variable \( x \), where we have projected out the joining variable \( x \). For the inverse reduction, we populate \( R_1, \ldots, R_k \) by introducing, for every tuple \( R(a_1, \ldots, a_k) \), \( k \) new tuples \( R_1((a_1, \ldots, a_k), a_1), \ldots, R_k((a_1, \ldots, a_k), a_k) \). It is easy to see that every \( R_i \) is a consistent relation where the variable \( x \) is the primary key. Additionally, the two instances are equivalent w.r.t. the repairs they admit. \( \square \)

It now remains to deal with the case of relations that have arity \( \geq 3 \) and additionally have a single variable as primary key. For this, we need the following lemma.

Proposition A.6. Let \( Q \) be a connected CQ containing \( R(x, y_1, \ldots, y_k) \). Denote by \( Q^s \) the query obtained by replacing \( R \) with \( R'(\overline{x}, y), S'_1(y, y_1), \ldots, S'_k(y, y_k) \), where \( y \) is a new variable. Then, \( \text{CERTAINTY}(Q) \overset{\text{FO}}{\sim} \text{CERTAINTY}(Q') \).

Proof. For the one direction, assume we have query \( Q \), along with a database instance \( I \). We transform \( I \) to an instance \( I^s \) for query \( Q^s \) as follows. For a tuple \( R(a, b_1, \ldots, b_k) \), we introduce in \( I^s \) the tuple \( R'(a, (b_1, \ldots, b_k)) \) and also, for \( i = 1, \ldots, k \) the tuples \( S_i((b_1, \ldots, b_k), b_i) \). Observe that our construction guarantees that \( S_i \) are consistent relations. It suffices to show that \( I \models Q \) iff \( I^s \models Q^s \). Notice that there is a one-to-one correspondence between repairs of \( I, I^s \). Indeed, if some repair \( r \) of \( I \) chooses \( R(a, b_1, \ldots, b_k) \), the corresponding repair \( r^s \) of \( I^s \) will choose \( R(a, (b_1, \ldots, b_k)) \) and vice versa. Now, observe that if \( Q(r) \) evaluates to true, so will \( Q(r^s) \) and vice versa.

For the inverse direction, assume \( Q^s \) and an instance \( I^s \). We transform \( I^s \) to an instance \( I \) of \( Q \) by constructing \( R(x, y_1, \ldots, y_k) = R'(\overline{x}, y), S_1(y, y_1), \ldots, S_k(y, y_k) \), i.e. in order to construct \( R \), we join all relations on \( y \) and then project out \( y \). We will show that \( I \models Q \) iff \( I^s \models Q^s \). First, assume that \( I^s \models Q^s \); we will show that \( I \models Q \). Indeed, consider a repair \( r \) of \( I \) and construct a repair \( r^s \) that makes the same choices as \( r \) for all common relations between \( Q, Q^s \) and, if \( R(a, b_1, \ldots, b_k) \in r \), then \( R'(a, (b_1, \ldots, b_k)) \in r^s \) for some \( b \) such that \( S_i((b_1, \ldots, b_k), b_i) \in I^s \) for every \( i = 1, \ldots, k \) (note that by our construction such a \( b \) always exists). Since \( Q(r^s) \) is true, \( Q(r) \) will be true as well.

For the inverse, assume \( I \models Q \) and consider a repair \( r^s \) of \( I^s \). Notice first that, for a key group \( R'(a, -) \), if \( R'(a, b) \) and \( \exists i : S_i((b, -)) \not\in I^s \), \( a \) will never contribute towards an answer for \( Q^s \), hence we can throw away w.l.o.g. such a key-group from consideration. Let \( R'(a, -) \) be any key group in \( I^s \); equivalently, \( R(a, -) \) is a key group in \( I \). Now, let \( R'(a, b) \) be the unique tuple in \( r^s \) from this key-group. As we have argued, there exist tuples \( S_1((b_1, b_1), \ldots, S_k((b_k, b_k)) \in I^s \) (and \( r^s \)) and these tuples are unique. By our construction, the instance \( I \) contains the tuple \( R(a, b_1, \ldots, b_k) \): this is the tuple that we include in \( r \). Since \( Q(r) \) evaluates to true, so must \( Q^s(r^s) \). \( \square \)

The combination of the above propositions proves Theorem A.2.

Lemma A.7. Let \( a \) be a value that does not appear in \( \mathcal{M}_Q(I) \), and let \( I^{-a} \subset I \) s.t. every key-group that contains \( a \) has been removed. Then, \( I \models Q \) iff \( I^{-a} \models Q \).