UNIVERSITY OF CALIFORNIA
SANTA CRUZ

CONSISTENT QUERY ANSWERING OF CONJUNCTIVE QUERIES UNDER PRIMARY KEY CONSTRAINTS

A dissertation submitted in partial satisfaction of the
requirements for the degree of

DOCTOR OF PHILOSOPHY

in

COMPUTER SCIENCE

by

Enela Pema

March 2014

The Dissertation of Enela Pema
is approved:

Table of Contents

List of Figures

List of Tables

Abstract

Consistent Query Answering of Conjunctive Queries Under Primary Key Constraints

by

Enela Pema

Integrity constraints are rules that express semantic conditions that a database should satisfy in order to be an accurate representation of the real world. Relational Database Management Systems are typically equipped with built-in mechanisms for enforcing integrity constraints on the data. Thus, ideally, every database should satisfy its integrity constraints. Yet, in reality violations of integrity constraints arise frequently under several different circumstances. For example, when integrating data from multiple heterogeneous sources into one common target schema, it is often the case that the data will violate the constraints of the target schema due to the mutually incompatible constraints of the sources. In other settings, integrity checking may be too expensive, and updates that might be causing violations to occur are nevertheless allowed to go through. A database that violates one or more of its integrity constraints is referred to as an *inconsistent database.* Inconsistent databases have long posed the challenge to develop suitable tools for meaningful query answering. The traditional approach for answering queries over inconsistent databases is data cleaning. This approach resolves violations of the integrity constraints by changing the database (e.g., by adding tuples, removing tuples, etc.) until it is transformed to a consistent one, which is then used to answer queries. Data cleaning algorithms typically rely on statistical and clustering techniques, which often entail making arbitrary decisions on what information to omit, add, or change. Such ad hoc nature of data cleaning can result in information loss.

An alternative, less intrusive, and more principled approach is the *consistent query answering* framework. In contrast to the data cleaning approach, the inconsistent database is left *as-is*. Instead, inconsistencies are handled at query time by considering all possible *repairs* of the inconsistent database, where a repair of a database I is a database r that is consistent with respect to the integrity constraints, and differs from I in a "minimal" way. Then, the consistent answers of q on I are the tuples that lie in the intersection of the results of q applied on each repair of I.

In this dissertation, we study the problem of computing the consistent answers to conjunctive queries over databases that may violate primary key constraints. For this class, the problem can be coNP-complete in data complexity. We study heuristics for efficiently computing the consistent answers in practice, regardless of the theoretical computational complexity. We develop EQUIP, a system that represents a fundamental departure from existing approaches. At the heart of EQUIP is a technique based on Binary Integer Programming (BIP). We use BIP to model the problem of computing the consistent answers, and rely on existing fast BIP solvers to efficiently compute the consistent answers. We carry out an extensive experimental investigation that validates the effectiveness of our approach, and shows that EQUIP scales well to large databases.

In addition, we study the data complexity of consistent query answering, aiming to delineate the boundary between tractability and intractability of the problem. We establish a dichotomy on the data complexity of consistent query answering for queries with two atoms, by giving a syntactic condition based on which, one can precisely determine the complexity as being either in PTIME, or coNP-complete. We provide sufficient conditions for tractability and intractability of consistent query answering for the class of acyclic and self-join free conjunctive queries. For this class of problems, we conjecture that there exists a dichotomy, and give a criterion for determining the complexity of each instance of the class.

Finally, we study the combined complexity of consistent query answering, where both the database instance and the query are part of the input. We show that, while consistent query answering can be Π_2^P-complete in combined complexity for conjunctive queries, it is coNP-complete for acyclic conjunctive queries. We leverage this advantage of acyclic conjunctive queries to explore alternative heuristics for consistent answers using Binary Integer Programming, which scale well on complex queries with many atoms. We implement EQUIP-AC, a module of EQUIP specialized to handle acyclic conjunctive queries, and show experimentally that EQUIP-AC is significantly more efficient than EQUIP in evaluating queries with a large number of atoms.

Acknowledgments

I would like to express my deepest gratitude to my advisers, Wang-Chiew Tan and Phokion Kolaitis, for their devoted guidance and encouragement throughout my PhD studies. I feel truly blessed to have been mentored by two amazing researchers and professors. I am thankful for every effort they have made to teach me how to become a good researcher. Without their intellectual and moral support, this work would not have been possible.

I am thankful to Jef Wijsen for serving as the external member of my dissertation committee. I appreciate the time and effort he put into thoroughly reading my dissertation and providing detailed feed-back on my work. I particularly enjoyed my visit at the University of Mons, as well as our collaboration during and after my visit.

I would like to also thank Anastasios Kementsietsidis for his mentorship during my internship at IBM Research. Our continuous collaboration has been a valuable and pleasant learning experience for me.

A special appreciation goes to all of the members of the Database Lab, who have made my life as a PhD student so enjoyable.

Last but not least, I am grateful to my family, for believing in me and for continuously supporting me in all my endeavors.

Chapter 1

Introduction

1.1 Motivation and Related Work

A fundamental issue in data management is assessing data quality. The immense importance of data quality is well understood, as it has a direct impact on the reliability of data analysis, and consequently, on any critical business decision making process. A key aspect of data quality is *data consistency*, which refers to the data being free of any discrepancies and in alignment with certain domain-specific constraints that capture semantic properties of the data. Typically, data consistency is dictated by a set of rules, commonly referred to as *data dependencies* or *integrity constraints*. Any traditional Database Management System is equipped with built-in mechanisms that impose integrity constraints on the data. Unfortunately, under several different circumstances, real-life data may become *dirty* or, *inconsistent*. To mention one typical scenario where data may frequently become inconsistent, consider a *data integration* setting. Data from different autonomous sources is combined into one global target schema. Even though the independent sources may be consistent, inconsistencies may arise during the data integration process due to potential disagreement between the integrity constraints of the source schemas and the target schema. In other settings, integrity checking may be so expensive that resolving violations in real-time would introduce intolerable latencies; hence, the application is forced to allow violations to arise. These motivational examples point to the need for inconsistency management systems that provide the necessary tools to interpret inconsistent information and answer queries in a meaningful way. While inconsistency management is a problem relevant to all data models (see [23] for a survey on general data quality issues), this dissertation focuses on the relational data model and on managing databases that may violate one or more integrity constraints, commonly referred to as *inconsistent databases.*

A popular approach for managing inconsistent databases is *data cleaning.* There is a large body of work on *data cleaning*, aiming to make meaningful sense of an inconsistent database (see [18, 22, 33] for a survey). In data cleaning, the approach taken is to first bring the database to a consistent state by resolving all conflicts that exist in the database, then use the *cleansed* database to answer queries. While data cleaning makes it possible to derive *one* consistent state of the database, this process usually relies on statistical and clustering techniques, which entail making decisions on what information to omit, add, or change; quite often, these decisions are of an ad hoc nature. Numerous cleaning strategies have been devised seeking to identify the "best" cleansed version of an inconsistent database. However, their effectiveness may vary from one application domain to another. Finally, data cleaning is a complex, time-consuming process that needs to be carried out on a regular basis as the data changes.

1.1.1 Consistent Query Answering Framework

An alternative, less intrusive, and more principled approach to handling inconsistent databases is the framework of *consistent query answering*, introduced by Arenas, Bertossi, and Chomicki in 1999 [4]. In contrast to the data cleaning approach, which modifies the database, the proposed approach suggests that the inconsistent database is left *as-is*. Instead, inconsistencies are handled at query time by examining all possible *repairs* of the inconsistent database, where a *repair* of an inconsistent database I is a database r that is consistent with respect to the integrity constraints, and differs from I in a "minimal" way. The minimality criterion captures the idea that a repair should be "as close as possible" to the original inconsistent database. Then, a *consistent answer* of a query q on a database I is a tuple that belongs to the set $\bigcap\{q(r) : r \text{ is a repair of } I\}$. In other words, the consistent answers of q on I are the tuples that lie in the intersection of the results of q applied on each repair of I. The intuition behind the proposed framework is that in reality, there are many reasonable ways of repairing an inconsistent database; thus, more meaningful query answers can be derived by taking into account all of these possible repairs. We illustrate the notions of repairs and consistent query answers with Example 1.

Example 1 Consider the instance of the relation $Employee(SSN, name, salary)$ shown in Figure 1.1. The attribute $Employee.SSN$ is a primary key, meaning that no two distinct employees can have the same SSN. The tuples (112, John, 20K) and (112, John, 30K) together violate the primary key because they agree on the SSN value. Similarly, the tuples (412, Anna, 34K) and (412, Ann, 34K) together violate the primary key constraint. One natural way to repair this database is to delete tuples until there are no more violations of the primary key. There are four possible repairs that can be constructed this way, as shown in Figure 1.1.

Employee

SSN	**name**	**salary**
112	John	20K
112	John	30K
412	Anna	34K
412	Ann	34K

repair r_1

SSN	**name**	**salary**
112	John	20K
412	Anna	34K

repair r_2

SSN	**name**	**salary**
112	John	20K
412	Ann	34K

repair r_3

SSN	**name**	**salary**
112	John	30K
412	Anna	34K

repair r_4

SSN	**name**	**salary**
112	John	30K
412	Ann	34K

Figure 1.1: Repairs of *Employee* w.r.t. the primary key $SSN \rightarrow \{name, salary\}$

Let q be the query $q(y) : -Employee(x, y, z)$. On the given database, "Anna" is not a consistent answer because it is not an answer on the repairs r_2 and r_4; whereas, "John" is a consistent answer because it appears in the answers of q on every repair.

In Example 1, the repairs are subsets of the inconsistent database since we derived them via tuple deletions only. Also, they are maximal with respect to set inclusion. In Example 1, repairing the inconsistent database by removing tuples seems natural since a violation of the primary key is witnessed by the presence of two or more tuples with the same key. In a variety of different settings, alternative repair semantics may be more suitable. For example, depending on the integrity constraints, a database could be repaired not only by deleting tuples, but also by introducing new tuples or modifying attribute values. Moreover, one may adopt a different notion of maximality of a repair. For instance, if a repair is obtained by removing or adding tuples, the "distance" between the repair and the inconsistent database can be better captured by the *symmetric difference* between the two databases. What makes a repair semantics more desirable than others, is typically an issue closely related to the class of integrity constraints at hand. Initially, the repair semantics proposed by Arenas et al. was symmetric-difference repairs [4]. We refer the reader to [2, 11] for an overview of different repair semantics that have been studied in the literature.

The consistent query answering framework has been well-received from the database research community as a principled approach to model and interpret inconsistent databases. Unfortunately, it was early observed that the number of possible repairs associated with a given inconsistent database can be exponential in the size of the database [4]. In Example 1, we only needed to examine a small number of repairs. However, even in this simple example with one relation and a primary key, it is easy to see that the number of repairs can be exponential. Clearly, a naive approach to compute the consistent answers by materializing all possible repairs and evaluating the query in every repair can be tremendously expensive and impractical. Therefore, research in consistent query answering has focused on discovering, when possible, polynomial-time algorithms for restricted classes of queries and constraints, and developing heuristics for efficient computation of consistent query answers.

Computational complexity of the framework A substantial amount of work has focused on analyzing the complexity of the framework of consistent query answering. Two main computational problems have been studied in the literature:

- *Repair checking*: For a fixed set of integrity constraints Σ, given instances I and r over the same database schema, is r a repair of I with respect to Σ?
- *Consistent query answering:* For a fixed query q and integrity constraints Σ over the same database schema, given a database I and a tuple t, is t a certain answer of q with respect to Σ?

The two decision problems formulated above capture data complexity: the set of integrity constraints and the query are assumed to be fixed, and complexity is expressed in terms of the database size only. The complexity of repair checking and consistent query answering can vary widely depending on the class of queries and constraints in consideration, as well as on the repair semantics. For subset-based repairing with respect to the very general class of *universal constraints*, the complexity of the repair checking problem can be as high as coNP-complete, and the complexity of consistent query answering can be as high as Π_2^P-complete for *first-order* queries [15]. These early results gave rise to an extensive, on-going study of the computational complexity under restricted classes of queries and constraints. Unfortunately, even when the much simpler classes of conjunctive queries and primary key constraints are considered, while the subset repair checking problem becomes tractable, computing the consistent query answers remains intractable; more precisely, it can be coNP-complete [15].

It is not our goal to give a detailed account of all literature in the consistent query answering area. In the following section, we will summarize some of the most influential work in consistent query answering that is related, and serves as a motivation for the work that will be presented in this dissertation. For a systematic and thorough survey of the problem of consistent query answering, we refer the reader to [11].

1.1.2 Problem Statement and Motivation

In this dissertation, we study the problem of consistent query answering for the class of conjunctive queries and primary key constraints. Conjunctive queries form a broad class of queries, widely used in practice. Primary key constraints are one of the most basic classes of integrity constraints, currently supported by any RDBMS. The subset repair semantics has been proposed as a suitable repair semantics in the presence of primary key constraints [15], as it is natural to resolve violations of primary key constraints by removing tuples from the database. In this dissertation, we are concerned with subset repairs only. As previously pointed out, the problem of computing the consistent query answers to conjunctive queries with respect to a set of primary key constraints can be coNP-complete in data complexity. The study of this class of queries and constraints has been a main theme in consistent query answering research over the past decade (see [63] for a survey). This work has focused on the computational complexity analysis of consistent query answering, and on developing approaches for computing the consistent answers in practice. In this dissertation, our main contributions are in both directions: (i) we study the computational complexity of consistent query answering, aiming to advance the knowledge on the boundary between tractability and intractability; and (ii) we study heuristics for efficiently evaluating the consistent answers. In the remainder of this section we elaborate into more detail on the two problems stated above.

(i) Data complexity of consistent query answering It was early established that in the presence of primary key constraints, the repair checking problem can be solved in polynomial time, and computing the consistent query answers to conjunctive queries can be coNP-complete. Observe that, under primary key constraints, to check if r is a subset repair of I, it suffices to check that r satisfies the primary key constraints, and that for every tuple t in $I - r$, the instance $r \cup \{t\}$ violates the constraints. Thus, repair checking can be done in polynomial time. Then, the complement of consistent query answering is in NP because one can guess a repair of the database and check if the query is false on that repair. Thus, consistent query answering is in coNP. In [15], coNP-hardness was established for a conjunctive query with repeated relation names and primary key constraints. Later on, more coNP-hard examples of conjunctive queries with primary keys were provided, even without repeated relation names in [29]. Much of the pursuit of tractable cases of consistent query answering for conjunctive queries under primary key constraints has focused on the *first-order expressibility* technique, initially proposed in [4], and further studied by Fuxman et al. [29, 30, 32], and Wijsen [57, 58, 61]. The technique amounts to taking the original query q, together with the constraints Σ, and constructing a first-order query q' such that the usual evaluation of q' on the inconsistent database I returns exactly the consistent answers to q on I with respect to Σ. If such a query q' exists, we say that the consistent answers of q are *first-order expressible*, or, that q is *first-order rewritable.* Fuxman et al. investigated the first-order expressibility approach and identified a subclass, called C_{forest}, of self-join free conjunctive queries that are first-order rewritable under primary key constraints. While C_{forest} is a broad class of conjunctive queries, many first-order rewritable conjunctive queries outside C_{forest} are known to exist. Major progress in this direction was made by Wijsen [61], who gave a necessary and sufficient condition for first-order expressibility of the consistent answers to a query q, provided q is a boolean acyclic conjunctive query without self-joins. The class C_{forest} and Wijsen's class of first-order expressible queries are incomparable, in the sense that, there exist queries that belong to one class but not the other. Unfortunately, the approach has its limitations as it may be the case that the consistent query answers of some query are not first-order expressible; however, they can be computed in polynomial time using some other algorithm. Such examples have been identified that are as simple as involving only two different binary relations. Concretely, Wijsen [60] showed that the consistent answers of the conjunctive query $q() : -R_1(x, y), R_2(y, x)$, where the first attribute of R_1 and R_2 is a key, are polynomial-time computable, but the query is not first-order expressible. Therefore, most recent study of the consistent query answering problem for conjunctive queries and primary key constraints has focused on determining the computational complexity when first-order expressibility is not an option [46, 60, 62]. Consistent query answering under primary keys raises a challenging complexity classification problem. Depending on the primary keys and the query, the actual complexity of consistent query answering may vary widely, as illustrated by the following three examples in which the underlined variables indicate that the corresponding attribute is the primary key constraint:

- If q_1 is the query $q_1() : -R_1(\underline{x}, y), R_2(\underline{y}, z)$, then the consistent query answers are first-order expressible [32, 61]. Hence, computing the consistent answers to q_1 is in P; actually, it is in the much lower class AC^0.
- If q_2 is the query $q_2() : -R_1(\underline{x}, y), R_2(\underline{y}, x)$, then the consistent query answers can be computed in polynomial time, but are not first-order expressible [60].
- If q_3 is the query $q_3() : -R_1(\underline{x}, y), R_2(\underline{x'}, y)$, then computing the consistent answers is coNP-complete [32].

How can these differences in complexity be explained? More precisely, are there efficiently checkable criteria that can be used to pinpoint the exact complexity of consistent query answering? It has been conjectured (e.g., in [2, 46, 61]) that a *dichotomy theorem* holds for the complexity of consistent query answering, namely, either the problem is in P or it is coNP-complete. The existence of a dichotomy in this class has been recognized as an intriguing question [2, 11, 45, 46, 51]. A dichotomy is an important theoretical result that, for a class of NP problems, it precisely determines the complexity of each instance of the problem as being either NP-complete or in P. To appreciate the point of this conjecture and the significance of a dichotomy theorem, recall that Ladner [47] has shown that if P $\neq$ NP, then there are decision problems that are in NP, but are neither in P nor are NP-complete; thus, the existence of a dichotomy theorem for a class of decision problems cannot be taken for granted a priori. Furthermore, a dichotomy theorem would have important implications in practice. Assuming that the dichotomy criterion is efficiently checkable, the dichotomy theorem could be used to determine, for any given instance of the problem, if a system should use some polynomial algorithm to compute the consistent answers, or heuristics when the problem is intractable. This dissertation makes a contribution in this direction by investigating the existence of a dichotomy for acyclic conjunctive queries without repeated relation names, in the presence of primary key constraints. The class of acyclic conjunctive queries is a broad class of queries, well-known for their good properties. Many hard problems in databases can be solved in polynomial time for acyclic conjunctive queries. For conjunctive queries with self-joins, other problems, very similar in spirit to the problem of consistent query answering, are difficult, e.g., getting a dichotomy for query evaluation on probabilistic databases [56]. Furthermore, obtaining a dichotomy for richer classes of constraints and queries is outside the reach at this point.

(ii) Heuristics for consistent query answering Another important direction of work in consistent query answering, which has been developed in parallel with the computational complexity study of the problem, has focused on building systems for solving the problem in practice. A few prototypes have implemented polynomial algorithms for restricted classes of queries and constraints; others have engineered heuristic-based approaches to efficiently handle more general classes of queries and constraints, regardless of their theoretical complexity.

In the direction of developing polynomial algorithms for consistent query answering, the most thoroughly studied technique is first-order rewriting. Fuxman et al. developed a system, ConQuer [29, 30, 31, 32], that generates first-order rewritings for the class C_{forest} and evaluates them over an RDBMS to compute the consistent answers. Through an extensive set of experiments, the authors present empirical evidence for the efficiency of the first-order expressibility technique, even on large databases with millions of tuples. However, ConQuer will reject any query not pertaining to the class C_{forest}, including several simple conjunctive queries with only two atoms. Such behavior may be undesirable in practical situations. More recently, Decan et al. [19] have implemented the first-order rewriting technique for acyclic and self-join free conjunctive queries, based on the algorithm proposed by Wijsen [61]. The main shortcoming of the first-order rewriting technique in general, is that it is limited to a sub-class of conjunctive queries and primary key constraints. To broaden the application of the first-order rewriting technique, recent work by Greco et al. [38] takes advantage of primary key constraints that might happen to be satisfied by the database at hand, to determine if consistent query answers can be obtained via a first-order rewriting. However, very simple queries are known to exist for which first-order rewriting cannot be used to compute the consistent answers. Two such examples are queries q_2 and q_3 mentioned earlier, which are boolean and involve only two atoms.

A different approach to tractable consistent query answering is the *conflict-hypergraph* technique, introduced by Arenas et al. [6] and further studied by Chomicki et al. [16, 17]. The conflict hypergraph is a graphical representation of the inconsistent database in which nodes represent database facts, and hyperedges represent minimal sets of facts that together give rise to a violation of the integrity constraints, where the class of integrity constraints can be as broad as *denial constraints*. A nice property of the conflict hypergraph is that its maximal independent sets are exactly the database repairs. Chomicki et al. designed a polynomial-time algorithm that processes the conflict hypergraph to compute the consistent answers to projection-free queries that may contain union and difference operators. This algorithm was implemented in a system called Hippo [17]. While the class of constraints supported by Hippo goes well beyond primary key constraints, the restriction to queries without projection limits its applicability. In particular, for the class of conjunctive queries and primary key constraints, this technique does not bring much to the table, as conjunctive queries that are self-join free and projection-free belong to C_{forest}.

In the direction of designing heuristics for consistent query answering, *disjunctive logic programming* and *stable model semantics* have been applied to arbitrary first-order queries under broad classes of constraints such as universal constraints, which include primary key constraints as a special case [5, 7, 21, 36, 37, 53]. For this, disjunctive rules are used to model the process of repairing violations of constraints. These rules form a disjunctive logic program, called the *repair program*, whose stable models are tightly connected with the repairs of the inconsistent database (and in some cases are in one-to-one correspondence with the repairs). For every fixed query, the *query program* is formed by adding a rule on top of the repair program. Query programs can be evaluated using engines, such as DLV [20], for computing the stable models of disjunctive logic programs, a problem known to be Π_2^P-complete. Two systems that have implemented this approach are Infomix [48] and ConsEx [13, 14]. The latter uses the *magic sets* method [9] to eliminate unnecessary rules and generate more compact query programs. Clearly, these systems can be used to compute the consistent answers of conjunctive queries under primary key constraints, which are the object of our study in this dissertation. The experimental evaluations with ConsEx show significant improvement over previous *DLV*-based systems. However, their experiments are run over small databases (about 10,000 tuples in total) and give little insight on what the overhead for computing the consistent answers is, and how the system scales over larger datasets. As we shall demonstrate in Section 3.5.1, ConsEx reveals a poor performance even on databases with a few thousands of tuples.

In a different direction, Flesca at al. [24, 25, 26, 27] studied the problem of repairing and querying databases with numerical attributes. To this effect, they used *Mixed Integer Linear Programming* to model repairs based on a minimal number of updates at the attribute level, and to extract consistent answers from inconsistent numerical databases. Their approach is focused on numerical databases and on aggregate constraints. In subsequent investigations [25, 27], they used *Integer Linear Programming* to compute the consistent answers of Boolean aggregate queries, as well as the range-consistent answers of `SUM`, `MIN`, and `MAX` queries, in this framework.

The limitations imposed by existing polynomial techniques and the poor performance exhibited by approaches based on logic programming have served as motivation for us to investigate efficient and scalable heuristics for consistent query answering.

In this dissertation, we propose a new heuristic for computing consistent query answers to all conjunctive queries under primary key constraints. Our heuristic builds on top of Binary Integer Programming (BIP), and makes use of existing powerful BIP solvers to support fast computation of consistent answers over large databases.

On top of the two main problems stated above, we also study combined complexity of consistent query answer, where both the query and the database are part of the input to the decision problem. Apart from some previous study of the combined complexity concerning much broader classes than primary key constraints [12, 55], this problem has received little attention. In our study, we leverage the good properties of acyclic conjunctive queries in relation to combined complexity, to devise better heuristics for computing the consistent answers with Binary Integer Programming. We implement EQUIP-AC, a module of EQUIP specialized to handle acyclic conjunctive queries, and show experimentally that EQUIP-AC is significantly more efficient than EQUIP in evaluating queries with a large number of atoms.

1.2 Contributions

In this section, we summarize the main contributions of this dissertation.

Heuristics for consistent query answering We develop heuristics for efficiently computing the consistent answers of conjunctive queries under primary key constraints. We propose a new approach that leverages Binary Integer Programming to compute the consistent answers to all conjunctive queries under primary key constraints. Our main contributions in this part of the dissertation are:

- We give an explicit polynomial-time reduction from the complement of the consistent answers of a conjunctive query under primary key constraints to the solvability of a binary integer program. The binary integer program is of size polynomial in the size of the database instance.
- We present an algorithm for computing the consistent answers of an arbitrary conjunctive query by evaluating binary integer programs using any off-the-shelf BIP solver. Our algorithm relies on the solutions of the binary integer program to determine the query answers that are not consistent.
- We build a system, called EQUIP, that implements our BIP-based approach over a relational database management system and a BIP solver.
- We conduct an extensive suite of experiments over both synthetic data and data derived from TPC-H [10] to determine the feasibility and effectiveness of EQUIP. Our experimental results show that EQUIP exhibits good performance with reasonable overheads on a variety of queries. Furthermore, EQUIP performs significantly better than existing systems, and also scales well.

Computational complexity analysis We investigate data complexity of computing the consistent answers to acyclic and self-join free conjunctive queries under primary key constraints. In this direction, we bring the following contributions:

- We establish a dichotomy on the data complexity of consistent query answering for self-join free conjunctive queries with two atoms, under primary key constraints. More specifically, we prove that for a given query and constraints in this class, computing the consistent query answers is either coNP-complete, or in P.

- We give a sufficient condition for intractability of consistent query answering of self-join free and acyclic conjunctive queries under primary key constraints.

- We give a sufficient condition for tractability of consistent query answering of self-join free and acyclic conjunctive queries under primary key constraints. In proving this result, we use a novel polynomial algorithm that translates the problem of determining the consistent answer to a boolean query, into the problem of finding a maximum-size independent set in a simple graph.

- We conjecture that the class of acyclic and self-join free conjunctive queries under primary key constraints exhibits a dichotomy in the complexity of consistent query answering, i.e., each instance is either solvable in PTIME or is coNP-complete. We give a syntactic criterion, which we believe determines the boundary between tractability and intractability, and show evidence that supports our conjecture.

Combined Complexity Our study in this part of the dissertation focuses on combined complexity of consistent query answering. We obtain the following results:

- We determine that consistent query answering is Π_2^P-complete in combined complexity, for conjunctive queries and primary key constraints.

- We observe that for acyclic conjunctive queries, the problem has lower complexity; it is coNP-complete.
- We explore a better reduction from the consistent answers of acyclic conjunctive queries to BIP. This reduction consists in generating programs whose size is polynomial in the size of the database and the query.
- We implement the new reduction for acyclic conjunctive queries in EQUIP-AC, a specialized module built on top of EQUIP.
- We run experiments with a variety of queries containing up to 7 atoms, to show that EQUIP-AC scales significantly better than EQUIP in evaluating queries with larger number of atoms.

Finally, we establish a new result concerning tractability of consistent query answering for a sub-class of acyclic and self-join free conjunctive queries via a reduction to Linear Programming, a problem well known to be solvable in polynomial time. This result gives rise to a new polynomial algorithm for consistent query answering based on Linear Programming, which can be further explored in the future and compared against existing polynomial approaches.

1.3 Organization of the Dissertation

In Chapter 2 we present background notions relevant to this dissertation. In Chapter 3 we focus on conjunctive queries and primary key constraints, and present a heuristic for computing the consistent query answers to this class using Binary Integer Programming. We present EQUIP, a system that implements our BIP-based approach, and describe the experiments we have conducted to validate its performance. In Chapter 4 we present our analysis on the data complexity of consistent query answering for acyclic and self-join free conjunctive queries under primary key constraints, and discuss the progress towards a dichotomy for consistent query answering. In Chapter 5 we discuss combined complexity of consistent query answering for conjunctive queries. We describe the implementation of EQUIP-AC, as a specialized module of EQUIP for evaluating acyclic conjunctive queries, and present experimental evaluations comparing EQUIP-AC against EQUIP.

Chapter 2

Basic Notions

Here we formalize basic notions and introduce preliminary notation that will be used throughout this dissertation.

Relational model A *relational database schema* is a finite collection $\mathbf{R}$ of relation symbols R, each with an associated set of attributes, denoted $\mathrm{Attr}(R)$. The attributes of a relation symbol R need not have names. Thus, if R is an n-ary relation symbol, then its attributes can be identified with the positions $1, \cdots, n$, which means that the set $\mathrm{Attr}(R)$ of the attributes of R coincides with the set $\{1, \cdots, n\}$. The *arity* of a relation symbol R is the number $|\mathrm{Attr}(R)|$. An *instance* of a relation symbol R is a set of tuples that have the same arity as R. An instance over the schema $\mathbf{R}$ is a set of instances of the relational symbols of $\mathbf{R}$. If R is a relation symbol in $\mathbf{R}$ and I is an instance over $\mathbf{R}$, then R^I denotes the interpretation of R on I. In what follows, we will often refer to the interpretation of R on I, as simply *the relation R in I*. A *fact* of an instance I is an expression of the form $R^I(a_1, \ldots, a_n)$ such that the tuple $(a_1, \ldots, a_n)$ is in R^I. If f is a fact of the form $R^I(a_1, \ldots, a_n)$, we also say that f is an R-fact.

Primary key constraints A *key constraint* of R is a subset X of $Attr(R)$. A key is said to be satisfied by an instance I if R^I does not contain two distinct facts that agree on all attributes of X. For example, if $R(A, B, C)$ is a relation schema and $X = \{A, B\}$ is the key of R, then in any database instance over the schema of R, it cannot happen that there are two facts $R(a, b, c)$ and $R(a, b', c')$, where $b \neq b'$ or $c \neq c'$. When there is a unique key constraint defined over a relation schema, this key is called a *primary key*. To express the fact that X is a key of R, we also use the notation $X \rightarrow \mathrm{Attr}(R)$. In what follows, we will assume that a relational schema $\mathbf{R}$ comes with a set Σ of primary key constraints over the relation symbols of $\mathbf{R}$. When a database instance satisfies a set of constraints Σ, we write $I \models \Sigma$. Otherwise, we write $I \not\models \Sigma$.

Conjunctive queries A *conjunctive query* is a first-order formula built from atomic formulas, conjunctions, and existential quantification. Thus, every conjunctive query is logically equivalent to an expression of the form $q(\mathbf{z}) = \exists \mathbf{w}. R_1(\mathbf{x_1}) \wedge \ldots \wedge R_m(\mathbf{x_m})$, where each $\mathbf{x}_i$ is a tuple of variables and constants, $\mathbf{z}$ and $\mathbf{w}$ are disjoint tuples of variables and each variable in $\mathbf{x_1}, \ldots, \mathbf{x_m}$ appears in exactly one of $\mathbf{z}$ and $\mathbf{w}$. We will often write conjunctive queries as *rules*; specifically, the rule expressing the preceding conjunctive query is $q(\mathbf{z}) : -R_1(\mathbf{x_1}), \ldots, R_m(\mathbf{x_m})$. We refer to the conjuncts $R_1(\mathbf{x_1}), \ldots, R_m(\mathbf{x_m})$ as the *atoms* of q. The *arity* of q is the number $|\mathbf{z}|$. A *boolean* conjunctive query is a conjunctive query in which all variables are existentially quantified; thus, when a boolean conjunctive query q is written as a rule, the left-hand side of the rule is $q()$. A conjunctive query that is not boolean is called *non-boolean.* If a conjunctive query has repeated relation names, we say that it contains a *self-join.* We refer to queries that do not contain self-joins as *self-join free* queries. In what follows, whenever we write a conjunctive query, we underline in each atom the positions of attributes that belong to the primary key of the relation symbol; such variables are called *key variables*, whereas, variables that appear in positions of attributes that are not part of the key are called *non-key variables.* For example, by writing $q() : -R_1(\underline{x}, y), R_2(\underline{y}, x)$, we indicate that the first attributes of R_1 and R_2 are, respectively, the keys of R_1 and R_2; furthermore, x is a key variable of the atom $R_1(\underline{x}, y)$ and y is a non-key variable of $R_1(\underline{x}, y)$, while y and x are, respectively, a key variable and a non-key variable of the atom $R_2(\underline{y}, x)$. The same variable can be at the same time a key variable and a non-key variable. For instance, in the atom $R(\underline{x}, x)$, variable x is a key variable and a non-key variable. In general, when a conjunctive query is presented in this form, we omit explicitly specifying the schema and the primary key constraints, since they can be derived from the formulation of the query itself.

Let q be a conjunctive query and let $R(\mathbf{x}, \mathbf{y})$ one of its atoms. We define $vars(R(\underline{\mathbf{x}}, \mathbf{y}))$ to be the set of variables appearing in the atom $R(\underline{\mathbf{x}}, \mathbf{y})$. We define $key(R(\underline{\mathbf{x}}, \mathbf{y}))$ to be the set of key-variables in the atom $R(\underline{\mathbf{x}}, \mathbf{y})$. Note that constants may occur in $\mathbf{x}$, but are not members of $key(R(\underline{\mathbf{x}}, \mathbf{y}))$; in particular, $key(R(\underline{\mathbf{x}}, \mathbf{y}))$ may be the empty set. We define $nkey(R(\underline{\mathbf{x}}, \mathbf{y}))$ to be the set of the non-key variables in the atom $R(\underline{\mathbf{x}}, \mathbf{y})$. Note that it is possible to have that $key(R(\underline{\mathbf{x}}, \mathbf{y})) \cap nkey(R(\underline{\mathbf{x}}, \mathbf{y})) \neq \emptyset$.

For simplicity, given a self-join free conjunctive query without self-joins, an atom $R(\mathbf{x}, \mathbf{y})$ will be denoted by its relation name R. Thus, we write $vars(R)$ instead of $vars(R(\underline{\mathbf{x}}, \mathbf{y}))$, and $key(R)$ instead of $key(R(\underline{\mathbf{x}}, \mathbf{y}))$.

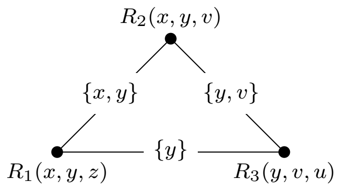
Acyclic conjunctive queries are a well-known class of conjunctive queries that were introduced in [8]. Several equivalent syntactic characterizations of acyclicity have appeared in the literature [8, 35, 50]. One such characterization is given next.

Definition 1 Let q be a conjunctive query with atoms $F_1, \cdots, F_i, \cdots, F_n$.

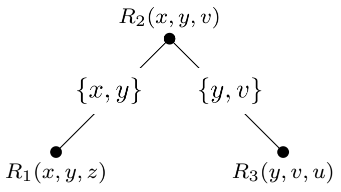
- The *complete intersection graph* of q is a labeled graph that has the atoms of q as vertices, and an edge between every two distinct atoms F_i and F_j, labeled by $L_{i,j}$, where $L_{i,j}$ is the set of variables that F_i and F_j share.
- An *intersection tree* of q is a spanning tree of the complete intersection graph of q.
- A *join tree* for q is an intersection tree that satisfies the following *connectedness condition*: whenever the same variable x occurs in two atoms F_i and F_j, then x occurs in every atom on the unique path linking F_i and F_j.
- We say that q is *acyclic* if it has a join tree.

We illustrate the notion of acyclicity of conjunctive queries in Example 2.

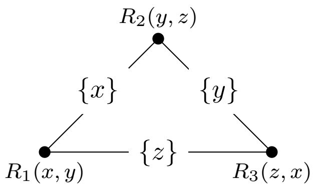
Example 2 Let q_1 be the query $q_1() : -R_1(x, y, z), R_2(x, y, v), R_3(y, v, u)$. Since this query is self-join free, we use the relation names to refer to the atoms. Figure 2.1a shows the complete intersection graph of q_1. Figure 2.1b is a join tree for q_1 because the only variable that atoms R_1 and R_3 share, which is y, appears also in the atom R_2. Let q_2



(a) Intersection graph of q_1



(b) Join tree of q_1



(c) Intersection graph of q_2

Figure 2.1: Intersection graphs for queries q_1, q_2

be the query $q_2() : -R_1(x,y), R_2(y,z), R_3(z,x)$. The complete intersection tree of q_2 is shown in Figure 2.1c. This query has no join tree; hence, it is cyclic.

Consistent Query Answering framework In this dissertation we focus on *subset repairs*, which are repairs that are obtained via deletions of entire tuples from the database. Next, we give precise definitions of the subset repairs, consistent query answers, and the decision problem of consistent query answering, CERTAINTY(q):

Definition 2 Let **R** be a relational database schema and Σ a set of integrity constraints over **R**.

- Let I be an instance of **R**. An instance r of **R** is a *subset repair* or, simply, a *repair* of I w.r.t. Σ if r is a maximal sub-instance of I that satisfies Σ, i.e., $r \models \Sigma$ and there is no instance r' such that $r' \models \Sigma$ and $r \subset r' \subseteq I$.

- Let q be a query and I an instance. We say that a tuple t is a *consistent answer of* q if for every repair r of I w.r.t Σ, we have $t \in q(r)$.

- Let q be a boolean query.

– If I is an instance, then the notation $I \models_{\Sigma} q$ denotes that q is true in every repair of I w.r.t. Σ, whereas the notation $I \not\models_{\Sigma} q$ denotes that q is false in at least one repair of I w.r.t. Σ.

– CERTAINTY(q) is the following decision problem: given an instance I, does it hold that $I \models_{\Sigma} q$?

In the notation CERTAINTY(q), we omit Σ (i.e., we write CERTAINTY(q) instead of CERTAINTY(q, Σ)) under the assumption that the primary key constraints can be inferred from the underlined positions in the atoms of q.

We say that two facts $R^I(a_1, \ldots, a_n)$ and $R^I(b_1, \ldots, b_n)$ are *key-equal* if they have the same relation name and agree on all attributes of the primary key. Two distinct key-equal facts $R^I(a_1, \ldots, a_n)$ and $R^I(b_1, \ldots, b_n)$ are said to form a conflict. A *key-equal group* of facts in a database I is a maximal set of key-equal facts, i.e., it is a set K of facts from I such that every two facts from K are key-equal, and no fact from K is key-equal to some fact from $I \setminus K$.

Complexity classes In this dissertation, we consider the following complexity classes:

- PTIME: decision problems solvable in polynomial time by deterministic Turing machines;

- NP: decision problems solvable in polynomial time by nondeterministic Turing machines;

- coNP: decision problems whose complements are in NP;

- Σ_2^P : decision problems solvable in polynomial time by nondeterministic Turing machines with an NP oracle;
- Π_2^P : decision problems whose complements are in Σ_2^p;

Chapter 3

EQUIP: A System for Consistent Query Answering

In this chapter, we describe our proposed approach for using Binary Integer Programming to compute the consistent answers to conjunctive queries under primary key constraints [44]. First, we will introduce some preliminary notions. Next, we will provide an explicit reduction from CERTAINTY(q) to Binary Integer Programming, based on which we devise an algorithm for computing the consistent answers. We will discuss the architecture of EQUIP the prototype system that implements our BIP-based algorithm, and we will present extensive experimental evaluations with synthetic and TPC-H data. All of the results presented in this chapter have been published in [44].

3.1 Preliminaries

Integer Linear Programming Integer Linear Programs (ILP) are optimization problems of the form $max\ \{\mathbf{c}^T\mathbf{x} \mid A\mathbf{x} \leq \mathbf{b}; x \in \mathbb{Z}^*\}$ (or, in the dual form $min\ \{\mathbf{b}^T\mathbf{y} \mid A^T\mathbf{y} \geq \mathbf{c}; y \in \mathbb{Z}^*\}$), where $\mathbf{b}$ and $\mathbf{c}$ are vectors of integer coefficients, $\mathbf{b}^T$ and $\mathbf{c}^T$ are the transpose of, respectively, $\mathbf{b}$ and $\mathbf{c}$, A is a matrix of integer coefficients and $\mathbf{x}$ (or, $\mathbf{y}$) is a vector of variables, ranging over the set $\mathbb{Z}^*$ of the non-negative integers. The function $\mathbf{c}^T\mathbf{x}$ (or, $\mathbf{b}^T\mathbf{y}$) is called the *objective function.* The system of inequalities $A\mathbf{x} \leq \mathbf{b}$ (or, $A^T\mathbf{y} \geq \mathbf{c}$) are the constraints to the program. Binary Integer Programming is the special case of Integer Linear Programming in which the variables must take values from the set $\{0, 1\}$. A *solution* to an integer program is an assignment of non-negative integers to the variables of the program that satisfies the constraints. An *optimal solution* to the integer program is a solution that yields the optimal value of the objective function. If $\mathbf{x}$ is the vector of variables in the program, we will use the notation $\hat{x}$ to denote a solution of the integer program, and the notation x^* to denote an optimal solution.

The decision problem underlying ILP is: Given a system $A\mathbf{x} \leq \mathbf{b}$ of linear inequalities with integer coefficients, does it have a solution? Similarly, the decision problem underlying Binary Integer Programming is: Given a system $A\mathbf{x} \leq \mathbf{b}$ of linear inequalities with integer coefficients, does it have a solution consisting of 0's and 1's? It is well known that both these decision problems are NP-complete (see [34]). In what follows, we will use the term ILP to refer to both the optimization problem and the underlying decision problem; similarly for BIP.

Let q be a non-boolean conjunctive query and let Σ be the set of primary key constraints in q. Since q is a monotone query and since the repairs of a database instance I are sub-instances of I, we have that the consistent answers of q on I form a subset of $q(I)$. Thus, every tuple in $q(I)$ is a candidate for being a consistent answer to q on I w.r.t. Σ. We refer to the tuples in $q(I)$ as the *potential consistent answers* to q on I w.r.t. Σ, or, simply, the *potential answers* to q on I w.r.t. Σ. In the case in which q is boolean, if $q(I)$ is *false*, then the consistent answer to q is also *false.* If $q(I)$ is *true*, then *true* is the potential answer to q on I. The *consistent part* of I is the the sub-instance of I consisting of all facts of I that are not involved in any conflict; more precisely, facts that are not key-equal with any other fact in the database. Clearly, the consistent part of a database I is the intersection of all repairs of I. It is also the union of all singleton key-equal groups. Given a database I, we will use the notation I_C to refer to the consistent part of I.

3.2 Modeling Consistent Query Answering with Binary Integer Programming

Let q be a conjunctive query. Since CERTAINTY(q) is in coNP and since Binary Integer Programming is NP-complete, there is a polynomial-time reduction from the complement of CERTAINTY(q) to Binary Integer Programming. In this section, we will give explicit natural reductions of CERTAINTY(q) to Binary Integer Programming.

3.2.1 Consistent Answers of Boolean Conjunctive Queries

We focus first on boolean conjunctive queries and show how one can model CERTAINTY(q), where q is a boolean conjunctive query, as the problem of checking if a set of linear equalities and inequalities with integer coefficients is satisfiable. In what follows, we will make use of the notion of a *minimal witness* to a conjunctive query on a database instance. Let I be a database instance and S a sub-instance of I. We say that S is a *minimal witness* to a boolean conjunctive query q if q is *true* on S and for every proper subset S' of S, we have that q is false on S'.

Theorem 1 *Let q be a boolean conjunctive query. Given a database instance I over the same schema as q, we construct in polynomial time the following system of linear equalities and inequalities with binary variables $x_{f_1}, \cdots, x_{f_i}, \cdots, x_{f_n}$, where each variable x_{f_i} is associated with a fact f_i of I.*

System (1):

$$\begin{array}{lll} (a) & \sum_{f_i \in K} x_{f_i} = 1, & \textit{for every key-equal group } K \textit{ of } I. \\ (b) & \sum_{f_i \in S} x_{f_i} \leq |S| - 1, & \textit{for every minimal witness } S \textit{ to } q \textit{ on } I. \end{array}$$

Then the following statements are equivalent:

- *There is a repair r of I in which q is false.*
- *System* (1) *has a solution.*

PROOF. Intuitively, the constraints (a) express the fact that from every key-equal group of facts, exactly one fact must appear in a repair. Therefore, every solution is associated with a repair. The constraints (b) express the fact that for every minimal witness S to q, not every fact of S should appear in a repair. There is a one-to-one mapping between the repairs in which q is false and the solutions to the set of constraints (a) and (b). Given a repair r of I on which q is false, one can construct a solution $\hat{x}$ by assigning $\hat{x}_{f_i} = 1$ if and only if f_i is a fact in r. In the other direction, given a solution $\hat{x}$ to the constraints (a) and (b), one can construct a repair r of I in which q is false by adding to r precisely the facts f_i such that $\hat{x}_{f_i} = 1$. Let $\hat{x}$ be a solution to System (1). The instance $r = \{f_i \in I : \hat{x}_{f_i} = 1\}$ is a repair because the constraints (a) are satisfied (constraints (a) guarantee that from every key-equal group, exactly one fact is chosen). Moreover, r does not satisfy q because the constraints (b) are satisfied. In the opposite direction let r be a repair that does not satisfy q. Then, we claim that the vector $\hat{x}$ such that $\hat{x_i} = 1$ if $f_i \in r$ and $\hat{x_i} = 0$ otherwise, is in fact a solution. Because r is a repair, the constraints (a) are satisfied. In addition, because $r \not\models q$, from every minimal witness S, at least one fact from S is not in r; Hence, the constraints (b) are also satisfied.

It remains to show that System (1) can be constructed in polynomial time. The size of System (1) is polynomial in the size of the database I (however, the degree of the polynomial depends on the fixed query q). Indeed, there are $|I|$ different variables, that is, as many variables as facts in I. One equality constraint is introduced for every key-equal group. Since every database fact belongs to exactly one key-equal group, the number of constraints in (a) is at most $|I|$. One inequality constraint is introduced for every minimal witness. If q has k atoms, then there are at most $|I|^k$ different minimal witnesses. Thus, there are polynomially many constraints of type (b). □

We illustrate Theorem 1 with Example 3 next.

Example 3 Let q be the query $q() : -R_1(\underline{x}, y, z), R_2(\underline{x'}, y)$. Let I be the displayed

R_1

	A	**B**	**C**
f_1	a	b$_1$	c$_1$
f_2	a	b$_2$	c$_1$
f_3	e	b$_1$	c$_2$

R_2

	D	**E**
f_4	d	b$_1$
f_5	d	b$_2$

database, where $f_1, \cdots, f_5$ are names used to identify database facts. For every fact f_i, we introduce a boolean variable x_{f_i}. Since $\{f_1, f_2\}$ forms a key-equal group of facts, we create the constraint $x_{f_1} + x_{f_2} = 1$. Doing the same for all the key-equal groups, we obtain the equalities (a). The sets of facts that are minimal witnesses to q on I are $\{f_1, f_4\}$, $\{f_2, f_5\}$, and $\{f_3, f_4\}$. From these minimal witnesses, we obtain inequalities (b).

$$(a) \quad \begin{aligned} x_{f_1} + x_{f_2} &= 1 \\ x_{f_3} &= 1 \\ x_{f_4} + x_{f_5} &= 1 \end{aligned} \qquad\qquad (b) \quad \begin{aligned} x_{f_1} + x_{f_4} &\leq 1 \\ x_{f_2} + x_{f_5} &\leq 1 \\ x_{f_3} + x_{f_4} &\leq 1 \end{aligned}$$

It is easy to check that the consistent answer to q is *false*, because q is false on the repair $r = \{f_1, f_3, f_5\}$. This gives rise to a solution to the system of the constraints (a) and (b) by assigning value 1 to a variable x_{f_i} if and only if $f_i \in r$. Thus, we obtain the solution $(x_{f_1}, x_{f_2}, x_{f_3}, x_{f_4}, x_{f_5}) = (1, 0, 1, 0, 1)$. $\square$

3.2.2 Consistent Answers of non-Boolean Conjunctive Queries

Theorem 1 gives rise to a technique for computing the consistent answers of a boolean conjunctive query under primary key constraints using a BIP solver. This technique can be extended to non-boolean queries as follows. Let q be a conjunctive query of arity k, for some $k \geq 1$. If I is a database instance and $\mathbf{t}$ is a k-tuple with values from the active domain of I, then $\mathbf{t}$ is a consistent answer of q on I if and only if for every repair r of I, we have that $\mathbf{t}$ is in $q(r)$. This is the same as the boolean query $q[\mathbf{t}]$ being true in every repair r of I, where $q[\mathbf{t}]$ is the query obtained from q by substituting variables from the head of q (i.e., variables that are not existentially quantified) with corresponding constants from $\mathbf{t}$. Thus, for every potential answer $\mathbf{a}$ to q, we can use Theorem 1 to check if $q[\mathbf{t}]$ is true in every repair.

The preceding approach for computing the consistent answers of a non-boolean query q on some database instance I requires that we solve as many instances of BIP as the number of potential answers of q on I. Even though the number of potential answers is a polynomial in the size of the database, if the database itself is large, it is conceivable that evaluating even hundreds of such programs may be expensive in practice. Furthermore, evaluating a BIP instance for every potential answer could be an overkill, as the different BIP instances share many constraints in common. For this reason, we explore a different technique for handling non-boolean queries that avoids constructing and evaluating a binary integer program for every potential answer. We will present this technique in two steps. First, we give a reduction that, given a database instance I, constructs a system of linear equalities and inequalities such that one can reason about all the potential answers to q by exploring the set of solutions to the system. This result, which is stated in Theorem 2, serves as a building block for Algorithm EliminatePotentialAnswers, which is presented later in Section 3.3.

Theorem 2 *Let q be a conjunctive query. Given a database instance I over the same schema as q, we construct in polynomial time the following system of linear equalities and inequalities:*

System (2):

Variables:
$x_{f_i} \in \{0,1\}$ *for every database fact f*
$u_{\mathbf{a}} \in \{0,1\}$ *for every $\mathbf{a} \in q(I)$*

Constraints:

$$\begin{array}{lll} (a) & \displaystyle\sum_{f_i \in K} x_{f_i} = 1, & \text{for every key-equal group } K \text{ of } I. \\ (b) & \displaystyle(\sum_{f_i \in S} x_{f_i}) - u_{\mathbf{a}} \leq |S| - 1, & \text{for every } \mathbf{a} \in q(I) \text{ and minimal witness } S \text{ of} \\ & & q[\mathbf{a}] \text{ on } I. \end{array}$$

Then, for every potential answer $\mathbf{a}$ to q on I, the following statements are equivalent:

- *There is a repair r of I in which $q[\mathbf{a}]$ is false.*
- *System (2) has a solution $(\hat{x}, \hat{u})$ such that $\hat{u}_{\mathbf{a}} = 0$.*

Proof. If r is a repair of I such that $\mathbf{a}$ is not an answer to q on r, we can construct a solution $(\hat{x}, \hat{u})$ by assigning $\hat{x}_{f_i} = 1$ if and only if f_i is a fact in r, and by assigning $\hat{u}_{\mathbf{a}} = 0$, and $\hat{u}_{\mathbf{b}} = 1$ for every other variable $u_{\mathbf{b}}$, where $\mathbf{b} \neq \mathbf{a}$. Because r is a repair, it is easy to see that $(\hat{x}, \hat{u})$ is a solution to the constraints (a). Every constraint of type (b) that does not involve $u_{\mathbf{a}}$ is trivially satisfied as a result of $\hat{u}_{\mathbf{b}}$ being assigned 1 for every other $u_{\mathbf{b}}$ such that $\mathbf{b} \neq \mathbf{a}$. Moreover, from $r \not\models q[\mathbf{a}]$ it follows that constraints of type (b) that does involve $u_{\mathbf{a}}$ are satisfied as well. In the other direction, given a solution $(\hat{x}, \hat{u})$ such that $\hat{u}_{\mathbf{a}} = 0$, we can construct a repair r to I that does not satisfy $q[\mathbf{a}]$ by adding to r precisely the facts f_i such that $\hat{x}_{f_i} = 1$. The fact that $(\hat{x}, \hat{u})$ satisfies the constraints (a) directly implies that r is a repair. Assume towards a contradiction that there exists a minimal witness S of $q[\mathbf{a}]$ in r. Then, $\sum_{f_i \in S} x_{f_i} = |S|$. Because $\hat{u}_{\mathbf{a}} = 0$, the constraint $\sum_{f_i \in S} x_{f_i} - u_{\mathbf{a}} \leq |S| - 1$ cannot be satisfied.

The size of the system of constraints is polynomial in the size of the database instance $|I|$. Indeed, the number of variables is equal to the number $|I|$ of facts in I plus the number $|q(I)|$ of potential answers. If the query q has k atoms, then $|q(I)|$ is at most $|I|^k$. The number of equality constraints is at most $|I|$. Finally, for every potential answer there are at most $|I|^k$ inequality constraints. $\square$

Observe that for every solution to System (2), there is a database repair that corresponds to the solution, and vice-versa. Furthermore, it is straightforward to verify that if only the constraints (a) are considered, then the solutions (to the constraints (a)) are in one-to-one correspondence with the repairs of I.

We illustrate Theorem 2 with Example 4.

Example 4 Consider the query $q(z) : -R_1(\underline{x}, y, z), R_2(\underline{x'}, y)$. Let I be the following database:

R_1

	A	**B**	**C**
f_1	a	b_1	c_1
f_2	a	b_2	c_1
f_3	e	b_1	c_2
f_4	g	b_1	c_3
f_5	h	b_2	c_3

R_2

	D	**E**
f_6	d	b_1
f_7	d	b_2

The key-equal groups give rise to equations in (a). The potential answers to q are c_1, c_2 and c_3. To represent each of these potential answers, we use variables u_{c_1}, u_{c_2} and u_{c_3}, respectively. Since $\{f_1, f_6\}$ is a minimal witness to potential answer c_1, we generate the inequality $x_{f_1} + x_{f_6} - u_{c_1} \leq 1$. The set $\{f_2, f_7\}$ is also a minimal witness to c_1. Hence, we generate the inequality $x_{f_2} + x_{f_7} - u_{c_1} \leq 1$. Doing the same for every minimal witness to every potential answer, we create the set of inequalities in (b):

$$(a) \quad \begin{array}{l} x_{f_1} + x_{f_2} = 1 \\ x_{f_3} = 1 \\ x_{f_4} = 1 \\ x_{f_5} = 1 \\ x_{f_6} + x_{f_7} = 1 \end{array} \qquad (b) \quad \begin{array}{l} x_{f_1} + x_{f_6} - u_{c_1} \leq 1 \\ x_{f_2} + x_{f_7} - u_{c_1} \leq 1 \\ x_{f_3} + x_{f_6} - u_{c_2} \leq 1 \\ x_{f_4} + x_{f_6} - u_{c_3} \leq 1 \\ x_{f_5} + x_{f_7} - u_{c_3} \leq 1 \end{array}$$

The vector $(\hat{x}, \hat{u})$ with $\hat{x} = (0, 1, 1, 1, 1, 1, 0)$ and $\hat{u} = (0, 1, 1)$ is a solution of the system (a) and (b). It is easy to check that the database instance $r = \{f_i \in I \mid \hat{x}_{f_i} = 1\}$ is a repair of I in which c_1 is not an answer to q. Hence, c_1 is not a consistent answer. Similarly, $\hat{x} = (0, 1, 1, 1, 1, 0, 1)$ and $\hat{u} = (1, 0, 1)$ form a solution of the system. The instance $r = \{f_i \in I \mid \hat{x}_{f_i} = 1\}$ is a repair of I in which c_2 is not an the answer to q. As Theorem 2 states, it is not a coincidence that $\hat{u}_{c_1} = 0$. On the other hand, c_3 is a consistent answer of q. Indeed, since in every solution of the above system it must hold that $\hat{x}_{f_4} = 1$ and $\hat{x}_{f_5} = 1$, and one of $\hat{x}_{f_6}$ or $\hat{x}_{f_7}$ must be assigned value 1, it follows that in order to satisfy both inequalities $x_{f_4} + x_{f_6} - u_{c_3} \leq 1$ and $x_{f_5} + x_{f_7} - u_{c_3} \leq 1$ the variable u_{c_3} must always take value 1. □

The system of constraints in Theorem 2 can be viewed as a *compact representation* of the repairs, since, as mentioned earlier, the solutions to the program encapsulate all repairs of the database. The conflict hypergraph and logic programming techniques, which were discussed in Section 1.1.2, can also be viewed as compact representations of all repairs. Specifically, in the case of the conflict hypergraph, the repairs are represented by the maximal independent sets, while in the case of logic programming, the repairs are represented by the stable models of the repair program.

3.3 An Algorithm for Consistent Query Answers using Binary Integer Programming

Theorem 2 gives rise to the following technique for computing the consistent query answers to a conjunctive query q: Given a database instance I, construct System (2), find all its solutions, and examine each of these solutions to determine which potential answers are not consistent. This technique avoids building a separate system of constraints for every potential answer, unlike the earlier technique based on Theorem 1. However, it still has to examine all possible solutions of System (2). Obviously, the number of solutions can be as large as the number of repairs of the database. Thus, it is not a priori obvious how these two techniques would compare in practice. In what follows in this chapter, we will demonstrate the advantage that Theorem 2 brings.

Algorithm EliminatePotentialAnswers In a sense, the solutions of System (2) form a compact representation of all repairs together with information about which potential answers are not found as an answer in each repair. For a given solution of System (2), let $\mathbf{a_1}, \cdots, \mathbf{a_n}$ be all potential answers such that $u_{\mathbf{a_1}} = u_{\mathbf{a_2}} = \cdots = u_{\mathbf{a_n}} = 0$. The repair r encoded by this solution (because of equations of type (a)) shows that none of the $\mathbf{a_i}$'s is a consistent answer. We will say that such a solution *eliminates*, or *filters out*, the potential answers $\mathbf{a_i}$ for $(1 \leq i \leq n)$. The encoding of repairs as solutions allows us to explore the space of solutions in an efficient manner, so that we can quickly filter out potential answers that are not consistent answers. It also provides the intuition behind our algorithm EliminatePotentialAnswers that we will describe next. EliminatePotentialAnswers is based on two main observations. First, one can differentiate among the solutions to System (2) according to the number of potential answers that they provide evidence for filtering out. Intuitively, a solution can be thought of as being better than another if it makes it possible to filter out a larger number of potential answers than the second one does. Thus, it is reasonable to examine first the solution that allows to filter out the most potential answers. We can model this idea by building a binary integer program that minimizes the sum of all variables $u_{\mathbf{a_j}}$. Second, some solutions may provide redundant information about potential answers that are not consistent answers as, in practice it often happens that the same potential answer is not found as a query answer in more than one repair. The algorithm EliminatePotentialAnswers, presented in Figure 3.1, is an iterative process that in each iteration evaluates a binary integer program, uses the optimal solution to eliminate potential answers that are not consistent answers, incorporates this knowledge into the binary integer program by adjusting the constraints, and re-evaluates the tweaked binary integer program to filter out more potential answers. This process continues until no more potential answers can be filtered out, and have remained unfiltered are automatically returned as consistent answers.

```
ALGORITHM  EliminatePotentialAnswers

1. Input
       $q$ : conjunctive query
       $I$ : database over the same schema as $q$
       $\mathcal{C}$: the set of constraints constructed from $q, I$ as described in Theorem 2.
       $\{\mathbf{a_1}, \cdots, \mathbf{a_p}\}$: the set of potential answers to $q$ on $I$

2. let CONSISTENT be a boolean array with subscripts $1, \ldots, p$ CONSISTENT$[j]$
   represents the element $\mathbf{a_j}$ and every entry is initialized to true.
3. let $i := 1$
4. let filter:=true
5. let $\mathcal{C}_1 := \mathcal{C}$
6. while (filter=true)
7.     let $P_i = \min\{\sum_{j \in [1..p]} u_{\mathbf{a_j}} | \text{subject to } \mathcal{C}_i\}$
8.     Evaluate $P_i$ using BIP engine
9.     let $(x^*, u^*)$ be an optimal solution for $P_i$
10.    let $\mathcal{C}_{i+1} := \mathcal{C}_i$
11.    let filter:=false
12.    for $j := 1$ to $p$
13.        if $u^*_{\mathbf{a_j}} = 0$ then
14.            let CONSISTENT[j]:=$false$
15.            Add to system $\mathcal{C}_{i+1}$ the equality $(u_{\mathbf{a_j}} = 1)$
16.            let filter:=true
17.    let $i := i + 1$
18. for $j := 1$ to $p$
19.    if CONSISTENT$[j]$ = true
20.        return $\mathbf{a_j}$
```

Figure 3.1: Algorithm for computing the consistent query answers using BIP.

In Algorithm EliminatePotentialAnswers, the optimal solution (x^*, u^*) returned is the one that contains the largest number of 0s in u^*, because the binary integer program minimizes the sum $\sum_{j\in[1..p]} u_{\mathbf{a_j}}$. This allows us to filter out right away many potential answers that are not consistent answers. The reason for adding the new constraints at the end of each iteration (line 15) is to trivially satisfy the constraints of type (b) related to potential answers that are already determined not to be consistent answers. Each time that we modify the program and re-evaluate it, we filter out more and more potential answers. In theory, the worst case scenario is encountered when we have to modify and re-evaluate the program once for every potential answer. This scenario is very unlikely to encounter in practice, especially when the number of potential answers is large. In Section 3.5, we show experimentally that even over databases with hundreds of thousands of tuples, two to four iterations will suffice to filter out all the potential answers that are not consistent answers. Note that if q is a boolean conjunctive query, then the only potential answer is *true*.

Theorem 3 *Let q be a conjunctive query and I a database instance. Then, Algorithm* EliminatePotentialAnswers *computes exactly the consistent answers to q on I.*

Proof. The proof uses the following two loop invariants:

1. At the i-th iteration, every optimal solution of P_i is a solution to constraints C.

2. At the end of the i-th iteration, if *filter* is *true* then the number of elements in consistent that are *false* is at least i.

The first loop invariant follows easily from the fact that $\mathcal{C}_i$ contains all the constraints of $\mathcal{C}$. The second loop invariant is proved by induction on i. Assume that at the end of the i-th iteration, the number of *false* elements in CONSISTENT is greater than or equal to i. For every $j \in [1..p]$ such that CONSISTENT$[j]$ is *false*, there must exist a constraint $u_{\mathbf{a_j}} = 1$ in $\mathcal{C}_i$. We will show that at the termination of iteration $i+1$, if *filter* is *true*, then the number of elements in CONSISTENT that are *false* is greater than or equal to $i+1$. Since *filter* is *true*, the BIP engine has returned an optimal solution (x^*, u^*) to P_{i+1} such that $u^*_{\mathbf{a_j}} = 0$ for at least some $j \in [1..p]$. Notice that it is not possible that at some previous iteration, CONSISTENT$[j]$ has been assigned *false*. If that were the case, then the constraint $u_{\mathbf{a_j}} = 1$ would be in $\mathcal{C}_{i+1}$, and (x^*, u^*) would not be a solution of $\mathcal{C}_{i+1}$. Therefore, at iteration $i+1$, at least one element in CONSISTENT that has value *true* is changed to *false*.

We show that the algorithm always terminates. The second loop invariant implies in a straightforward manner that the algorithm terminates in at most p iterations. Next, we show that for any $m \in [1..p]$, a potential answer $\mathbf{a_m}$ is a consistent answer if and only if CONSISTENT$[m] = 1$ at termination. In one direction, if $\mathbf{a_m}$ is a consistent answer, then Theorem 2 implies that for every solution $(\hat{x}, \hat{u})$ to the constraints $\mathcal{C}$, it always holds that $\hat{u}_{\mathbf{a_m}} = 1$. Since for every $i \in [1..p]$, every optimal solution to P_i is also a solution to $\mathcal{C}$, we have that the algorithm will never execute line 14 for $j = m$. Hence, the value of CONSISTENT$[m]$ will always remain *true*.

In the other direction, if CONSISTENT$[m]$ is *true* after the algorithm has terminated, then assume towards a contradiction that $\mathbf{a_m}$ is not a consistent answer. If the algorithm terminates at the i-th iteration, then every solution to P_i is such that it assigns value 1 to every variable $u_{\mathbf{a_j}}$, for $j \in [1..p]$. So, the minimal value that the objective function of P_i can take is p. If $\mathbf{a_m}$ were not a consistent answer, we know from Theorem 2 that there must be a solution $(\hat{x}, \hat{u})$ to $\mathcal{C}$ such that $\hat{u}_{\mathbf{a_m}} = 0$. We will reach a contradiction by showing that we can construct from $(\hat{x}, \hat{u})$ a solution (x^*, u^*) to P_i such that $\sum_{j \in [1..p]} u^*_{\mathbf{a_j}} < p$. The vectors $\hat{x}, \hat{u}$ are defined as follows: $x^* = \hat{x}$, $u^*_{\mathbf{a_m}} = \hat{u}_{\mathbf{a_m}}$ and $u^*_{\mathbf{a_j}} = 1$ for $j \neq m$. Because the equality constraints (a) in $\mathcal{C}_i$ and in $\mathcal{C}$ are the same, we have that x^* will satisfy the constraints (a) in $\mathcal{C}_i$. For $j \neq m$ and since $u^*_{\mathbf{a_j}} = 1$, all inequalities in (b) that involve $u^*_{\mathbf{a_j}}$, where $j \neq t$, are trivially satisfied. In addition, the left-hand-side of every inequality in the constraints (b) that involves $u^*_{\mathbf{a_m}}$ will evaluate to the same value under $(\hat{x}, \hat{u})$ and (x^*, u^*) (this follows directly from the fact that $x^* = \hat{x}$ and $u^*_{\mathbf{a_m}} = \hat{u}_{\mathbf{a_m}}$). Finally, all equality constraint that may have been added to $\mathcal{C}$ during previous iterations are satisfied since $u^*_{\mathbf{a_j}} = 1$ for $j \neq m$, and the equality $u^*_{\mathbf{a_m}} = 1$ cannot be in $\mathcal{C}_i$. Now, it is clear that (x^*, u^*) is a solution to P_i such that $\sum_{j \in [1..p]} u^*_{\mathbf{a_j}} = p - 1$. This contradicts the assumption that all solutions to P_i yield minimal value p of the objective function. $\square$

3.4 Architecture of EQUIP

We have developed a system, called EQUIP, for computing the consistent query answers to conjunctive queries under primary key constraints. Our system, at its core, implements Algorithm ELIMINATEPOTENTIALANSWERS. Our strategy in the implementation of EQUIP has been to push as much processing as possible to the RDBMS. This strategy is motivated by two main reasons. First, SQL queries can be efficiently evaluated by the underlying database system, whereas large BIP programs could potentially take a long time to evaluate given the inherent hardness of Binary Integer Programming. Secondly, by partly relying on the database system to compute the consistent query answers, we can take advantage of RDBMS features such as database statistics and indexes for query optimization.

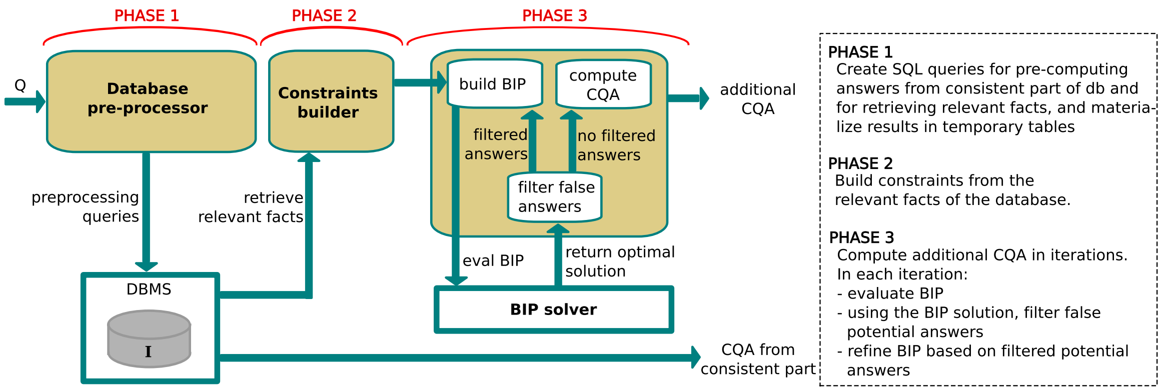
EQUIP executes in three phases (see Figure 3.2), corresponding to the modules: 1) database pre-processor, 2) constraints builder, and 3) consistent query answers (CQA) evaluator.

- PHASE 1: Compute answers of the query from the consistent part of the database and extract database facts that may be relevant for computing the additional answers to the query.

- PHASE 2: Construct the set of constraints based on Theorem 2 with the set of database facts retrieved in PHASE 1.

- PHASE 3: Run Algorithm ELIMINATEPOTENTIALANSWERS to eliminate the potential answers that are not consistent answers.

PHASE 1: The database pre-processor performs two main tasks in PHASE 1 as an attempt to optimize subsequent phases: (i) it retrieves answers of q from the consistent part of the database, and (ii) it retrieves the database facts that are *relevant* for computing additional consistent answers of q on I. We describe these two steps next.



PHASE 1
Create SQL queries for pre-computing answers from consistent part of db and for retrieving relevant facts, and materia-lize results in temporary tables

PHASE 2
Build constraints from the relevant facts of the database.

PHASE 3
Compute additional CQA in iterations.
In each iteration:
- evaluate BIP
- using the BIP solution, filter false potential answers
- refine BIP based on filtered potential answers

Figure 3.2: Architecture of EQUIP

(i) Given an inconsistent database, it is reasonable to assume that there are facts in the database that do not participate in any violations. In fact, we expect that in most real-life inconsistent databases the majority of the facts are not involved in conflicts. Let I_C we the subset of the facts of I that are not involved in any conflict. Then, as a preliminary optimization step, we compute a portion of the consistent answers by evaluating q on I_C. It is easy to see that the result of q over the consistent part of the database I_C is a subset of the consistent query answers of q on I. Computing the answers of q on I_C does not necessarily require that we materialize I_C and then evaluate q on top of I_C. In fact, the answers to q on I_C can be computed using SQL queries.

(ii) After having computed $q(I_C)$, every tuple in $q(I) - q(I_C)$ is an additional potential answer which may or may not be a consistent answer. For a given potential answer $\mathbf{a}$, not all facts in the database are needed to check if $\mathbf{a}$ is a consistent answer or not. For instance, if K is a key-equal group of facts such that none of the facts of K participates in a minimal witness of $q[\mathbf{a}]$, then repairs of this violation have no effect whatsoever on whether **a** is a consistent answer or not. This observation suggests that it is enough to focus on only a subset of the facts in I in order to check the additional potential answers. The notion of *relevant database facts* is formalized in Definition 3:

Definition 3 *Let q be a boolean conjunctive query and let I be a database. A fact f in I is relevant to q if there exists a fact g that is key-equal with f, and a minimal witness S of q on I such that $g \in S$.*

Note that the above definition does not require that f and g are distinct. Every fact is key-equal with itself. Therefore, a fact f that is not key-equal with any other fact in the database, may still be a relevant fact as long is it participates in a minimal witness of the query. Obviously, all facts involved in some minimal witness are considered relevant.

Next, in Proposition 1, we show that to compute the consistent answers, it suffices to focus on the relevant facts.

Proposition 1 *Let q be a boolean conjunctive query, and let I be a database instance. Let $I_{R,q}$ be the set of facts in I that are relevant to q. Then, q is false on some repair of I if and only if q is false on some repair of $I_{R,q}$.*

Proof. First, we make the following observation about I and $I_{R,q}$: for every repair r of I, there exists a repair r_1 of $I_{R,q}$ and a repair r_2 of $I - I_{R,q}$ such that $r = r_1 \cup r_2$; and vice-versa. It follows directly from the definition of $I_{R,q}$, that for every two key-equal facts f and g, either they are both in $I_{R,q}$, or neither of them is in $I_{R,q}$. Therefore, every repair r of I can be obtained by combining a repair of $I_{R,q}$ with a repair of $I - I_{R,q}$.

In one direction, because $I_{R,q}$ is a subset of I, it is obvious that if q is false on some repair of I then it is also false on some repair of $I_{R,q}$.

In the other direction, assume there exists r_1, a repair of $I_{R,q}$ such that q is false on r. Let r_2 be any repair of $I_{R,q}$, and let $r = r_1 \cup r_2$. We will argue that r is a repair of I such that $r \not\models q$. From the observation made above, it follows that r is a repair of I. Assume towards a contradiction that $r \models q$. then, there exists a minimal witness S of q on $I_{R,q}$. Note that no fact in $I - I_{R,q}$ participates in a minimal witness of q on I. Then, it can only happen that $S \subseteq r_1$, thus, contradicting the assumption that $r_1 \not\models q$. $\square$

It follows from Proposition 3, that to check if a tuple **a** is a consistent answer of a non-boolean query q, it suffices to check only the repairs of $I_{R,q[\mathbf{a}]}$.

In Figure 3.3 we describe at a high level the main steps followed in PHASE 1. A more detailed description of PHASE 1 is provided in Appendix A.1. Steps 1 and 2 compute the consistent answers from the consistent part of the database, I_C. While there are many different ways of computing $q(I_C)$, the strategy we follow in our implementation of PHASE 1 is to first compute the key-values associated with key-equal groups of size greater than 1, and then compute the answers to q on I that are obtained from minimal witnesses that involve only facts whose key-values are not found in the previous step. This is achieved in step 1 and step 2 in Figure 3.3. More specifically, for every relation R_i mentioned in q, we create a view KEYS_R$_i$ that holds all key-values that are not unique in R_i. The view ANS_FROM_CON holds all tuples in $q(I_C)$, and it can be computed by selecting all tuples in $q(I)$ that can be obtained via some minimal witness S, where every R_i-fact in S is such that its key value is not found in KEYS_R$_i$. The set ANS_FROM_CON is immediately returned as part of the consistent answers of q on I (see bottom of Figure 3.2). The computation of the relevant database facts is described in step 3 and step 4 of Figure 3.3. In step 3, we compute all minimal witnesses to the potential answers into a new relation called WITNESSES. This is achieved by evaluating a query on top of I that selects all facts that together form a witness to some potential answer $\mathbf{a}$, where $\mathbf{a}$ is not in ANS_FROM_CON. Finally, for every relation R_i that is mentioned in q, we compute the relevant R_i-facts into a relation RELEVANT_R$_i$. Again, we use an SQL query to achieve this.

```
PHASE 1: DATABASE PRE-PROCESSING

Input: $\mathbf{R}$ : Schema with relation names $\{R_1, \cdots, R_i, \cdots, R_l\}$
    $q(\mathbf{z}) : -R_{p_1}(\underline{\mathbf{x_1}}, \mathbf{y_1}), \cdots, R_{p_j}(\underline{\mathbf{x_j}}, \mathbf{y_j}), \cdots, R_{p_k}(\underline{\mathbf{x_k}}, \mathbf{y_k})$ for $j \in [1..k]$ and $1 \leq p_j \leq l$
    $I$ : database over $\mathbf{R}$

1. for all $R_i, 1 \leq i \leq l$
   create view KEYS_R$_i$ that contains all primary key values $\mathbf{d}$ s.t. there exists more
   than one fact of the form R$_i$($\mathbf{d}$,_) in $I$

2. create view ANS_FROM_CON that contains all tuples $\mathbf{t}$ s.t.
   - $\mathbf{t} \in q(I)$, and
   - there exists a minimal witness $S$ for $q(\mathbf{t})$, and s.t. no fact $R_i(\mathbf{d}, \_) \in S$ has its
     key-value $\mathbf{d}$ in KEYS_R$_i$

3. create view WITNESSES that contains all tuples $(\mathbf{t_{p_1}}, \cdots, \mathbf{t_{p_j}}, \cdots, \mathbf{t_{p_k}})$ s.t.
   - $R_{p_j}(\mathbf{t_{p_j}}) \in I$ for $1 \leq j \leq k$, and
   - the set $S = \{R_{p_j}(\mathbf{t_{p_j}}) \ : \ 1 \leq j \leq k\}$ forms a minimal witness for $q(\mathbf{a})$, where $\mathbf{a}$
     is a potential answer that is not in ANS_FROM_CON

4. for all $R_i, 1 \leq i \leq l$
   create view RELEVANT_R$_i$ that contains all tuples $\mathbf{t}$ s.t.
     - $R_i(\mathbf{t}) \in I$, and
     - there exists an atom $R_{p_j}(\underline{\mathbf{x_j}}, \mathbf{y_j})$ in $q$ such that $p_j = i$ and there exists a tuple
       $(\mathbf{t_{p_1}}, \cdots, \mathbf{t_{p_j}}, \cdots, \mathbf{t_{p_k}})$ in WITNESSES s.t. $R_{p_j}(\mathbf{t_{p_j}})$ and $R_i(\mathbf{t})$ are key-equal
```

Figure 3.3: Description of PHASE 1

Example 5 illustrates PHASE 1 and all of the above mentioned optimizations.

Example 5 Let $q(z) : -R_1(\underline{x}, y, z)$, $R_2(\underline{x'}, y, w)$ be a query over the schema $\{R_1(\underline{A_1}, B_1, C_1)$, $R_2(\underline{A_2}, B_2, C_2)\}$. The following views will be generated in PHASE 1:

KEYS_R$_1(A_1)$: select A$_1$ from R$_1$ group by A$_1$ having count(*)>1

KEYS_R$_2(A_2)$: select A$_2$ from R$_2$ group by A$_2$ having count(*)>1

ANS_FROM_CON(C_1):

```
    select R₁.C₁
    from R₁ inner join R₂ on R₁.B₁=R₂.B₂
    where R₁.A₁ not in (select * from KEYS_R₁) and
          R₂.A₂ not in (select * from KEYS_R₂)
```

WITNESSES$(A_1, B_1, C_1, A_2, B_2, C_2)$:

```
    select A₁,B₁,C₁,A₂,B₂,C₂
    from R₁ inner join R₂ on R₁.B₁=R₂.B₂
    where R₁.C₁ not in (select * from ANS_FROM_CON)
```

RELEVANT_R$_1(A_1)$: select * from R$_1$ inner join WITNESSES on R$_1$.A$_1$=WITNESSES.A$_1$

RELEVANT_R$_2(A_2)$: select * from R$_2$ inner join WITNESSES on R$_2$.A$_2$=WITNESSES.A$_2$ □

Finally, an additional optimization is implemented in each of the steps of Phase 1, which is not accounted for in Figure 3.3, but can be found in the detailed description of this phase in Appendix A.2. This optimization consists in identifying the *relevant variables* and subsequently, the *relevant attributes* for computing the consistent answers of a given query q. The intuition behind this optimization is the following: If x is an existentially quantified variable that occurs only once in q, and such that the only occurrence of x is at a non-key position, then it is not important to know the exact values that occur at the position of x in the database. More specifically, if two key-equal facts f and g disagree only on the the value in the position of x, then keeping f or g in a repair, has no effect on the query answers on that repair. Thus, repairing such violation is irrelevant w.r.t. the computation of the consistent answers. We elaborate further on this optimization.

Definition 4 *Let q be a boolean conjunctive query over a database schema $\mathbf{R}$. Let R be a relation symbol mentioned in some atom in q, and let A be an attribute of R. The attribute A is relevant for q if at least one of the following conditions holds:*

- *A is in the primary key of R.*
- *There exists an atom $R(\mathbf{x})$ in q such that the variable x that appears in $\mathbf{x}$ in the position of A, occurs more than once in q.*

Otherwise, A is irrelevant.

Notice that in general, because the query may contain repeated relation names, if A is an attribute of a relation R, it could happen that different variables appear in the position of A in different atoms that mention R. For instance, if $q() - : R(\underline{x}, x), R(\underline{x}, y)$, where the schema of R is $R(A_1, A_2)$, then x appears in the position of A_2 in the atom $R(\underline{x}, x)$, and y appears in the position of A_2 in the atom $R(\underline{x}, y)$. In this example, it is obvious that A_1 is a relevant attribute. Also, A_2 is a relevant attribute as well because in atom $R(\underline{x}, x)$, the variable x, which is mentioned more than once, appears in the position of A_2. Given a boolean conjunctive query q, a variable $v \in vars(q)$ is called *relevant* if it appears in some atom of q in the position of a relevant attribute. Next, we will show that given any boolean conjunctive query q over a schema $\mathbf{R}$, we can "ignore" the irrelevant attributes of the database when reasoning about $\textsc{certainty}(q)$. This idea is formalized in Proposition 2.

Proposition 2 *Let q be a boolean conjunctive query. Let q' be the query formed by removing from q the irrelevant variables. Then, there is a first-order reduction from $\textsc{certainty}(q)$ to $\textsc{certainty}(q')$.*

Proof. We make the following trivial observations about q and q':

Observation 1: If $R(\mathbf{x})$ is an atom in q and $R(\mathbf{x'})$ is an atom in q' such that the variables in $\mathbf{x'}$ are the relevant variables of q in $\mathbf{x}$, then $key(R(\mathbf{x})) = key(R(\mathbf{x'}))$.

Observation 2: If $R_i(\underline{\mathbf{x_i}}, \mathbf{y_i})$, $R_j(\underline{\mathbf{x_j}}, \mathbf{y_j})$ are two distinct atoms in q (the relation symbols R_i and R_j need not be different), and $R_i(\underline{\mathbf{x'_i}}, \mathbf{y'_i})$, $R_j(\underline{\mathbf{x'_j}}, \mathbf{y'_j})$ are atoms in q' such that the variables in $\mathbf{x'_i}, \mathbf{y'_i}, \mathbf{x'_j}, \mathbf{y'_j}$ are the relevant variables of q in $R_i(\underline{\mathbf{x_i}}, \mathbf{y_i})$, $R_j(\underline{\mathbf{x_j}}, \mathbf{y_j})$, then $vars(R_i(\underline{\mathbf{x_i}}, \mathbf{y_i})) \cap vars(R_j(\underline{\mathbf{x_j}}, \mathbf{y_i})) = vars(R_i(\underline{\mathbf{x'_i}}, \mathbf{y'_i})) \cap vars(R_j(\underline{\mathbf{x'_j}}, \mathbf{y'_j}))$.

Observations 1 and 2 follow trivially from the definition of relevant variables.

Let I be a database over the schema of q. We construct a database I' over the schema of q' as follows: for every fact $f = R_i(t)$ in I, we generate a fact $g = R_i(s)$ in I' such that t and s agree in the positions of the relevant variables of R. In other words, we copy from t to s only the values that appear in the positions of relevant variables. To denote that f has generated g, we write $f \rightrightarrows g$. Next, we argue that there is a repair r of I that falsifies q if and only if there is a repair r' of I' that falsifies q'.

The vice-versa is also true, i.e., if g_1 and g_2 are not key-equal and $f_1 \rightrightarrows g_1$, $f_2 \rightrightarrows g_2$, then f_1 and f_2 are not key-equal.

($\Rightarrow$) In this direction, we assume that there exists a repair r of I such that $r \not\models q$. Let r' be the instance that contains all facts g such that $f \rightrightarrows g$ for some fact $f \in r$. We show that r' is a repair of I' and $r' \not\models q'$. From Observation 1, it follows that r' is a repair. Assume towards a contradiction that $r' \models q'$. Then, r' contains a minimal witness S' of q'. Let S be the set of facts $\{f \ : f \in r \textit{ and exists } g \in S' \textit{ s.t. } f \rightrightarrows g\}$. From Observation 2 it follows that S is a minimal witness for q.

($\Leftarrow$) Assume that there exists a repair r' of I' such that $r' \not\models q'$. Let r be an instance constructed by choosing from every key-equal group of facts, a single fact f such that $f \rightrightarrows g$, for some $g \in r'$. Notice that there could be multiple facts from I that have generated the same fact g in I'. The construction of r from r' demands that we arbitrarily pick only one of these facts. Again, from Observation 1 it follows that r is a repair. Assume towards a contradiction that $r \models q$. Then, r contains a minimal witness S of q. Let S' be the set of facts $\{g \ : g \in r' \textit{ and exists } f \in S \textit{ s.t. } f \rightrightarrows g\}$. From Observation 2 it follows that S' is a minimal witness for q'.

It is easy to see that the reduction is first-order, since the database I' can be computed from I by simply doing a projection over each relation R of I, on those attributes of R that are relevant to q. $\square$

Proposition 2 can be easily adapted for non-boolean q, if in addition, we consider the free variables in the query to be relevant variables.

PHASE 2: This phase uses the temporary tables created in PHASE 1 to build the set of constraints, as described in Theorem 2. Every tuple in RELEVANT_R$_i$ is represented by a variable in the integer program. From every group of facts in RELEVANT_R$_i$ that have the same values in the positions of the key attributes of R$_i$, we add a constraint of type (a) as described in Theorem 2. Every tuple in WITNESSES represents a minimal witness to a potential answer to the query. So, from every tuple in WITNESSES we construct a constraint of the type (b) as described in Theorem 2.

PHASE 3: In this phase, the additional potential answers are first retrieved by simply taking a projection over the view WITNESSES on the attributes that appear in the head of the query q. With this input, the set of constraints built in PHASE 2, and the database instance, we run Algorithm ELIMINATEPOTENTIALANSWERS next. In the first iteration, we start with a binary integer program whose objective function is the sum of all variables $u_{\mathbf{a_j}}$, where each $u_{\mathbf{a_j}}$ represents a distinct potential answer $\mathbf{a_j}$, and whose constraints are those from PHASE 2. In every iteration, the program is evaluated using a BIP solver. The first optimal solution that is returned by the optimizer is used to filter out potential answers that are not consistent answers. In each iteration, the integer program is augmented with more constraints. As the added constraints make some of the existing constraints of type (b) (see Theorem 2) trivially satisfiable, this results in a program (in the next iteration) that is simpler to evaluate than the one evaluated in the current iteration.

3.5 Experimental Evaluation

We have conducted an extensive set of experiments with EQUIP. Our goal is to analyze the performance of EQUIP on conjunctive queries whose consistent answers have varying complexity and on databases of varying size and of varying degree of inconsistency (i.e., the percentage of tuples involved in conflicts). We also seek to understand how EQUIP compares with earlier consistent query answering systems, such as ConQuer and ConsEx.

3.5.1 Experimental Setting

Our experiments have been carried out on a machine running on a Dual Intel Xeon 3.4GHz Linux workstation with 4GB of RAM. We use the 64 bit Ubuntu v11.04, DB2 Express C v10.1 as the underlying DBMS, and IBM's ILOG CPLEX v12.3 [1] for solving the binary integer programs. Our system is implemented in Java v1.6.

No established benchmark for consistent query answering exists. Moreover, as noted in [29], conjunctive queries obtained from TPC-H queries after removing aggregates, grouping, and subqueries are first-order rewritable and, in fact, most of them are in the class C_{forest}, hence their consistent answers can be computed using the ConQuer system. We have conducted experiments using such queries and data derived from TPC-H to compare EQUIP with ConQuer. However, since we are interested in understanding the performance of EQUIP on conjunctive queries whose consistent answers are of varying complexity (from first-order rewritable to coNP-complete), we compiled a list of queries of varying complexity and generated synthetic inconsistent databases for our experiments. We now discuss this experimental setting in some detail.

Benchmark queries Table 3.3 contains the list of queries we compiled. The queries vary according to the number of atoms, the number of free variables, and the computational complexity of consistent query answering, which can be: first-order rewritable; in PTIME but not first-order rewritable; coNP-complete. Queries Q_1 to Q_{14} are shown not to be first-order rewritable based on the characterization of first-order expressibility provided in [61]. For the two-atom queries $Q_1, Q_2, Q_3, Q_8, Q_9, Q_{10}$, their complexity (coNP-hard or PTIME) can be immediately derived from our dichotomy for two-atom queries [42], which we will present in Section 4.2. The complexity of the three-atom queries Q_4 to Q_7, and Q_{11} to Q_{14} can be derived from the sufficient conditions for intractability and tractability that we will present later in Section 4.3. First-order rewritability of queries Q_{15} to Q_{21} can be shown using the results in [61]. Moreover, queries Q_{15} to Q_{18} are also in the class C_{forest} [32], which provides another proof that they are first-order rewritable.

Complexity: coNP-complete

$Q_1() : -R_5(\underline{x}, y, z), R_6(\underline{x'}, y, w)$
$Q_2(z) : -R_5(\underline{x}, y, z), R_6(\underline{x'}, y, w)$
$Q_3(z, w) : -R_5(\underline{x}, y, z), R_6(\underline{x'}, y, w)$
$Q_4() : -R_5(\underline{x}, y, z), R_6(\underline{x'}, y, y), R_7(\underline{y}, u, d)$
$Q_5(z) : -R_5(\underline{x}, y, z), R_6(\underline{x'}, y, y), R_7(\underline{y}, u, d)$
$Q_6(z, w) : -R_5(\underline{x}, y, z), R_6(\underline{x'}, y, w), R_7(\underline{y}, u, d)$
$Q_7(z, w, d) : -R_5(\underline{x}, y, z), R_6(\underline{x'}, y, w), R_7(\underline{y}, u, d)$

Complexity: PTIME, not first-order rewritable

$Q_8() : -R_3(\underline{x}, y, z), R_4(\underline{y}, x, w)$
$Q_9(z) : -R_3(\underline{x}, y, z), R_4(\underline{y}, x, w)$
$Q_{10}(z, w) : -R_3(\underline{x}, y, z), R_4(\underline{y}, x, w)$
$Q_{11}() : -R_3(\underline{x}, y, z), R_4(\underline{y}, x, w), R_7(\underline{y}, u, d)$
$Q_{12}(z) : -R_3(\underline{x}, y, z), R_4(\underline{y}, x, w), R_7(\underline{y}, u, d)$
$Q_{13}(z, w) : -R_3(\underline{x}, y, z), R_4(\underline{y}, x, w), R_7(\underline{y}, u, d)$
$Q_{14}(z, w, d) : -R_3(\underline{x}, y, z), R_4(\underline{y}, x, w), R_7(\underline{y}, u, d)$

Complexity: First-order rewritable

$Q_{15}(z) : -R_1(\underline{x}, y, z), R_2(\underline{y}, v, w)$
$Q_{16}(z, w) : -R_1(\underline{x}, y, z), R_2(\underline{y}, v, w)$
$Q_{17}(z) : -R_1(\underline{x}, y, z), R_2(\underline{y}, v), R_7(\underline{v}, u, d)$
$Q_{18}(z, w) : -R_1(\underline{x}, y, z), R_2(\underline{y}, v), R_7(\underline{v}, u, d)$
$Q_{19}(z) : -R_1(\underline{x}, y, z), R_8(\underline{y, v}, w)$
$Q_{20}(z) : -R_5(\underline{x}, y, z), R_6(\underline{x'}, y, w), R_9(\underline{x}, y, d)$
$Q_{21}(z) : -R_3(\underline{x}, y, z), R_4(\underline{y}, x, w), R_{10}(\underline{x}, y, d)$

Table 3.3: Benchmark queries used in our experiments

Database generation In our experiments with the above queries, we use synthetic databases, which we generate in two steps: (a) generate a consistent database; and (b) generate an inconsistent database from the consistent database by inserting tuples that would violate some key constraints. Instead of using available database generators, such as *datagen* in TPC-H for (a), we have generated our own databases, because it is difficult to express over the TPC-H schema a variety of meaningful queries that are not first-order rewritable. This difficulty arises because the meaningful joins one can perform between two tables in the TPC-H schema, are joins between attributes that are related via a foreign key constraint. As a result, all meaningful conjunctive queries one might express over the TPC-H schema are first-order rewritable.

a) Generation of the consistent databases Each relation in the generated consistent database has the same number (denoted as *r_size*) of facts. The values of the non-key attributes are generated so that for every two atoms R_i, R_j that share variables in any of the queries, approximately 25% of the facts in R_i join with some fact in R_j, and vice-versa. The third attribute in all of the ternary relations, which is sometimes projected out and never used as a join attribute in Table 3.3, takes values from a uniform distribution in the range [1, *r_size*/10]. Hence, in each relation, there are approximately *r_size*/10 distinct values in the third attribute, each value appearing approximately 10 times. The choice of the distributions is made with the purpose to simulate reasonably high selectivities of the joins and large numbers of potential answers.

b) Generation of the inconsistent databases From the consistent database, an inconsistent database is obtained by adding, for each relation, tuples that violate some key constraints. The inconsistency generator takes as input a consistent relation and two additional parameters: (a) *nr_conflicts*: the total number of violations to be added to the consistent relation; and (b) *c_size*: the size of each key-equal group. The inconsistent version of a consistent relation r is obtained by repeatedly adding tuples that violate some key value. Specifically, a key value from the key values of the tuples in r, where the number of violations for the key value is less than *c_size* - 1, is first selected. Subsequently, additional distinct tuples with the same key value are added, so that there is a total of *c_size* distinct tuples with the same key value. The non-key attributes of these newly generated tuples are obtained by using the non-key attributes of some randomly selected tuples in r. The purpose of reusing the non-key attributes is to preserve the existing data distributions as we augment r with new tuples. This process is repeated until a total of *nr_conflicts* is obtained. In fact, the inconsistency generator that we have just described is similar to the one used in ConQuer [32], except that we have added the parameter *c_size* to control the sizes of the key-equal groups.

Other experimental parameters Unless otherwise stated, our experiments use databases with 10% conflicts in which each key-equal group contains two facts. The *evaluation time* of a consistent query answering system is the time between receiving the query to the time it takes for the system to return the last consistent answer to the query. We always generate five random databases with the same parameters, i.e., with the same size, the same degree of inconsistency, and the same number of facts per key-equal group. For each database, we run each query five times and take the average of the last three runs, and finally, we take the averages of the query evaluation times over the five databases. One of the CPLEX parameters that can be varied is the *relative GAP parameter*. This parameter is used to specify the level of "tolerance" on the optimality of the solution that is found. The GAP parameter can be set to allow the optimizer to return a less-than-optimal, but "good enough" solution. In other words, the solution that is returned may not be optimal, but will have an estimated gap from the optimal solution that is within the tolerance limit given by the GAP parameter. Having a large GAP parameter usually allows the optimizer to find a solution much earlier. Since Algorithm 3.1 remains correct if suboptimal solutions are returned, we have set the GAP parameter to 0.1. In this way, CPLEX may avoid the long running times that may otherwise be incurred by searching for an optimal solution. Finally, to make sure vectors of exact integer (0/1) solutions are returned, we set parameter EpInt to 0, where EpInt is the tolerance for integral solutions.

3.5.2 Experimental Results and Comparisons

Our first goal was to determine how the EQUIP system compares against existing systems that are capable of computing the consistent answers of conjunctive queries that are not first-order rewritable. The only other systems that have this capability are the ones that rely on the logic programming technique, such as Infomix and ConsEx. Since ConsEx is the latest system in this category and additionally, it implements an important optimization based on *magic sets*, we have compared EQUIP against ConsEx.

Comparison with ConsEx Our findings, depicted in Figure 3.4, show that EQUIP outperforms ConsEx by a significant margin, especially on larger datasets and even on small queries whose consistent answers are coNP-hard (i.e., queries Q_1 and Q_2), as well as on the small queries whose consistent answers are in PTIME, but not first-order rewritable (i.e., queries Q_8 and Q_9).

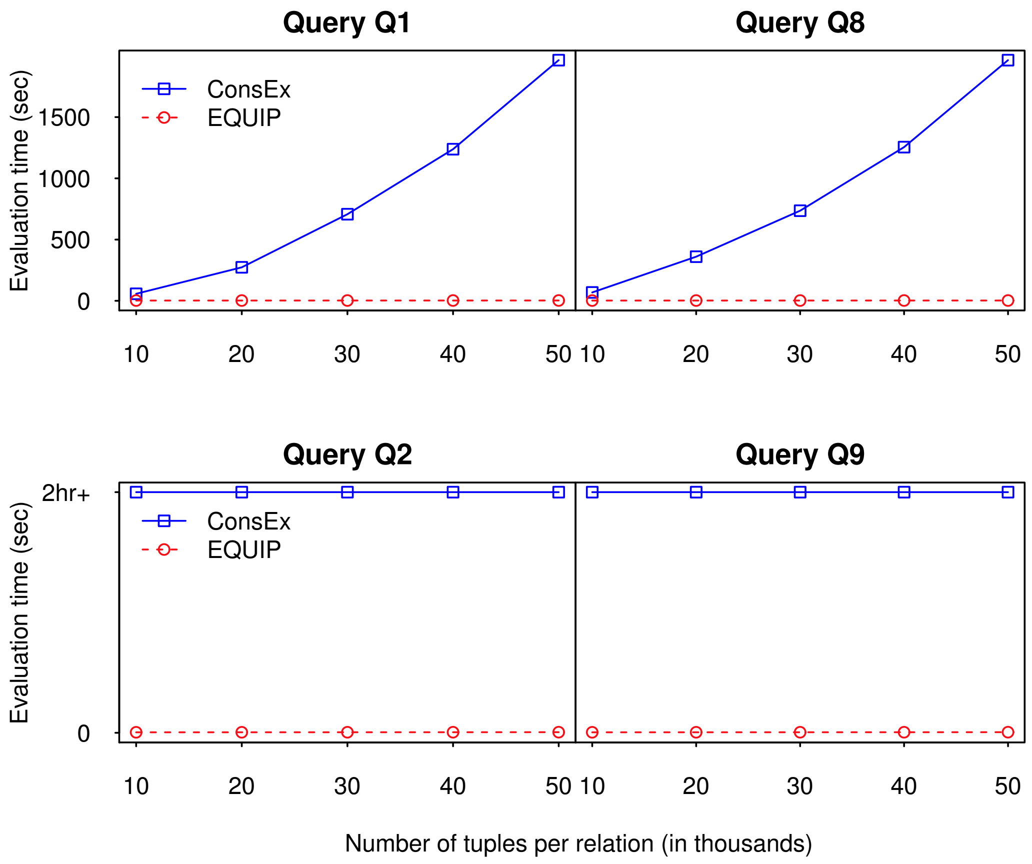
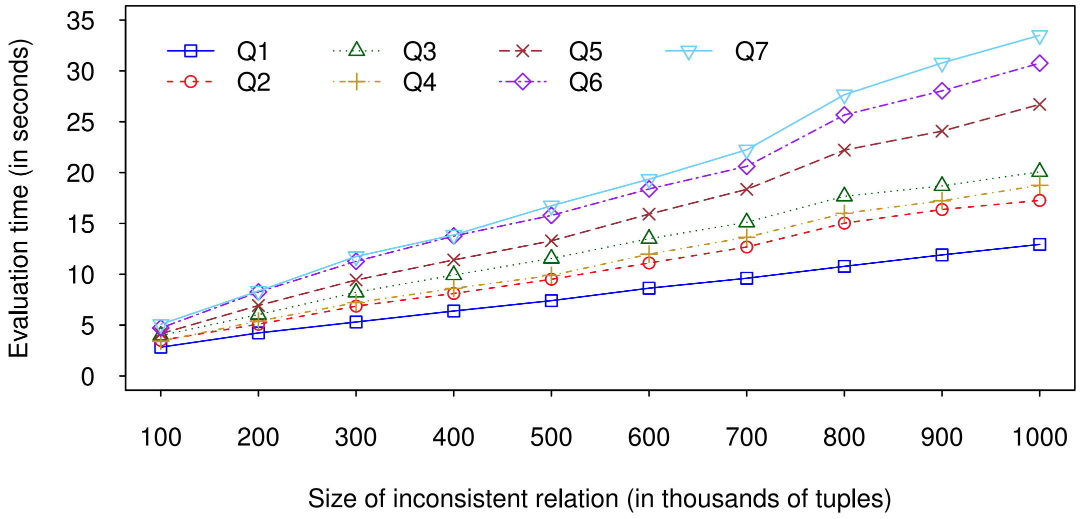


Figure 3.4: Comparison of EQUIP with ConsEx.

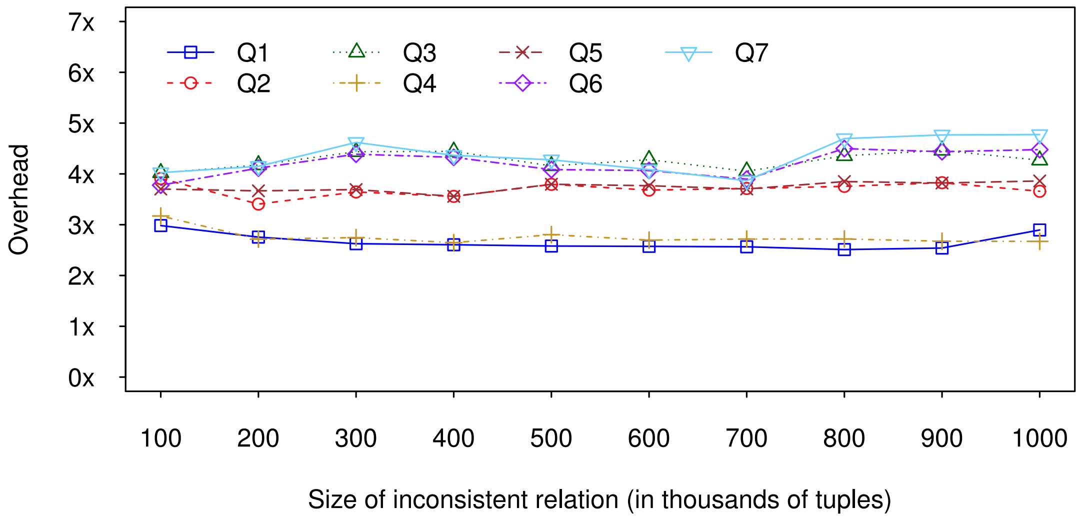
Our second goal was to investigate the total evaluation time and the overhead of EQUIP on conjunctive queries whose consistent answers have varying complexity.

Evaluation time for coNP-hard queries In Figure 3.5a, we show the total evaluation time in seconds, using EQUIP, of each query $Q_1 - Q_7$ as the size of the inconsistent database is varied. In Figure 3.5b, we show the overhead of computing the consistent answers, relative to the time for evaluating the query over the inconsistent database (i.e., the time for evaluating the potential answers). More precisely, if t_1 is the time needed to evaluate q on I, and t_2 is the time needed to compute the consistent answers to q on I, the overhead is defined as $\frac{t_2}{t_1}$.

Figure 3.5a shows a sub-linear behavior of the evaluation of consistent query answers, as we increase the database size. Note that the overhead for computing consistent answers is rather constant and no more than 5 times the time for usual evaluation. In particular, the boolean queries Q_1 and Q_4 incur noticeably less overhead than the rest of the queries. The reason is that these queries happen to evaluate to true over the consistent part of the constructed databases, and hence, since there are no additional potential answers to check, no binary integer program is constructed and solved.



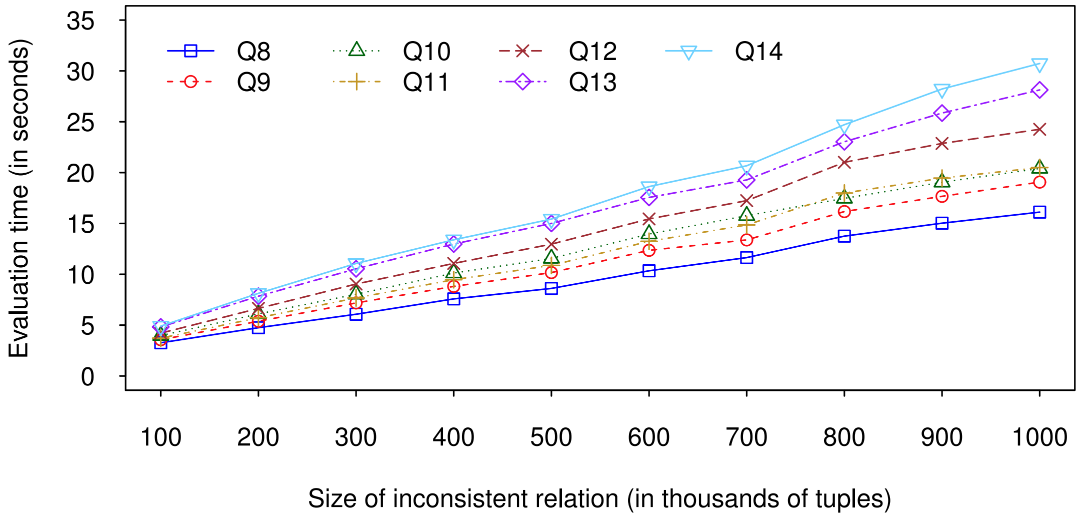
(a)



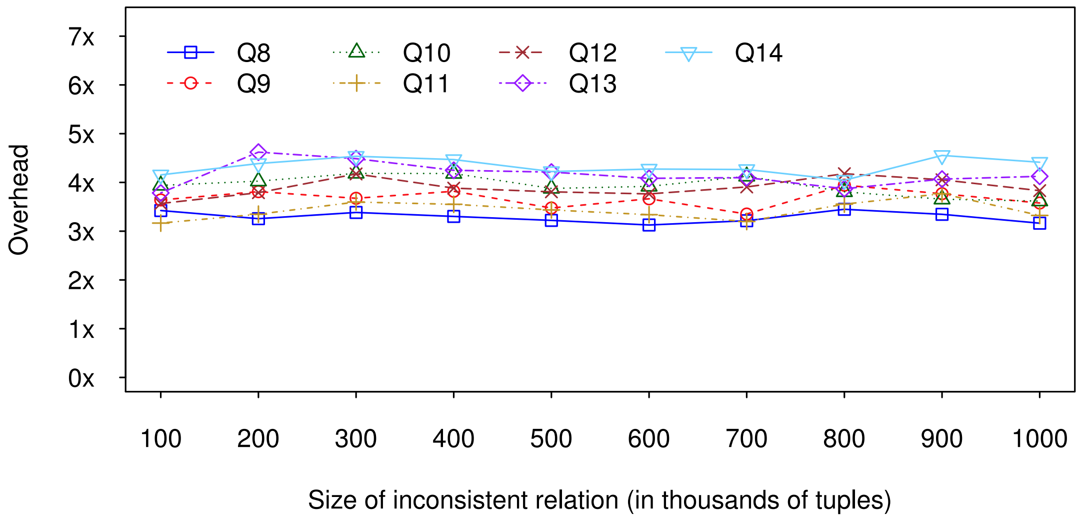
(b)

Figure 3.5: Evaluation time and overhead of EQUIP for computing consistent answers of coNP-hard queries Q_1 to Q_7.

Evaluation time for PTIME, not first-order rewritable queries In Figure 3.6, we show the performance of EQUIP on conjunctive queries whose consistent answers are in PTIME, but are not first-order rewritable. Since EQUIP is based on Binary Integer Programming, which is an NP-complete problem, we do not expect EQUIP to perform better on such queries as compared to queries that are coNP-hard. Indeed, Figure 3.5 and Figure 3.6 show that the performance of EQUIP on the tractable queries Q_8 to Q_{14} is comparable to the performance of EQUIP on the queries Q_1 to Q_7 that are intractable.



(a)

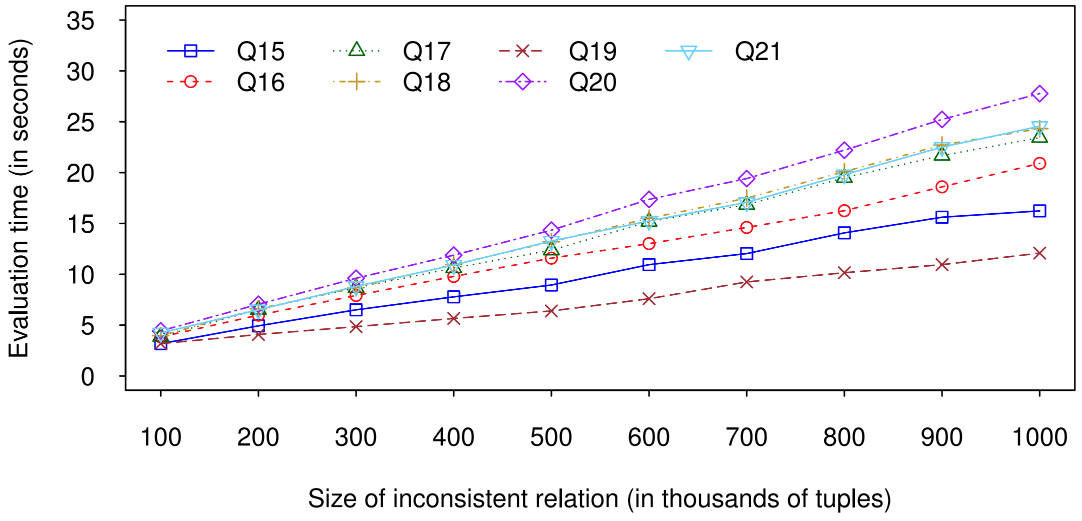


(b)

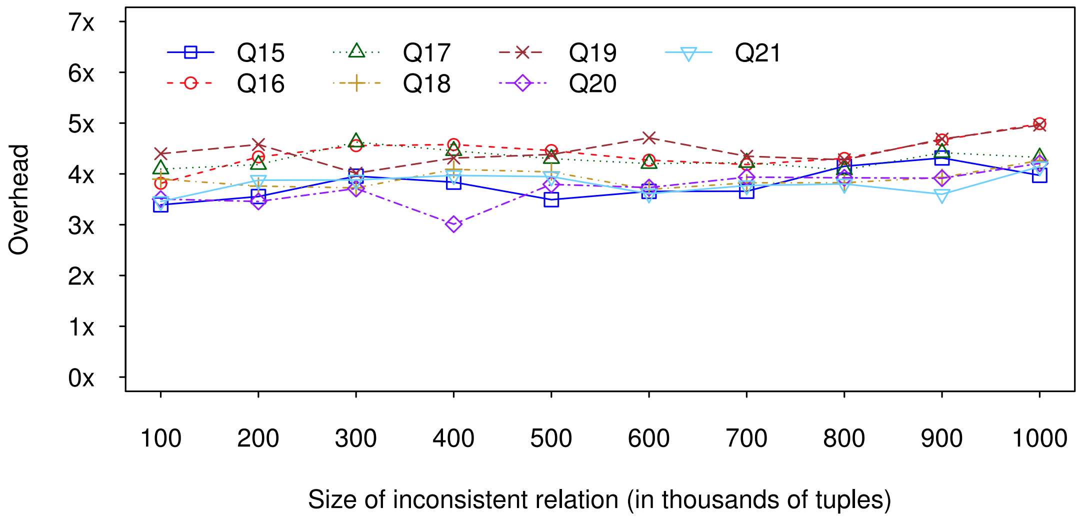
Figure 3.6: Evaluation and overhead of EQUIP for computing consistent answers of PTIME, but not-first-order rewritable queries Q_8 to Q_{14}.

Evaluation time for first-order rewritable queries Next, we evaluate EQUIP on first-order rewritable queries. As Figure 3.7 shows, the performance of EQUIP on such queries is comparable to that on queries in the other classes (see Figure 3.5 and Figure 3.6).

We also compare EQUIP with ConQuer on queries Q_{15} to Q_{18}, since these queries are in C_{forest}, hence their consistent answers can be computed using ConQuer. The results of this comparison are shown in Figure 3.8. ConQuer performs better than EQUIP on each of the queries Q_{15} to Q_{18}. This is not surprising, because EQUIP is agnostic to the complexity of the query and "blindly" reduces queries into binary integer programs to be solved by CPLEX. This reduction is exponential in the size of the query and instance, even though these queries are first-order rewritable. These findings suggest that ConQuer is preferred whenever the query is in C_{forest}.



(a)



(b)

Figure 3.7: Evaluation and overhead of EQUIP for computing consistent answers of first-order rewritable queries Q_{15} to Q_{21}.

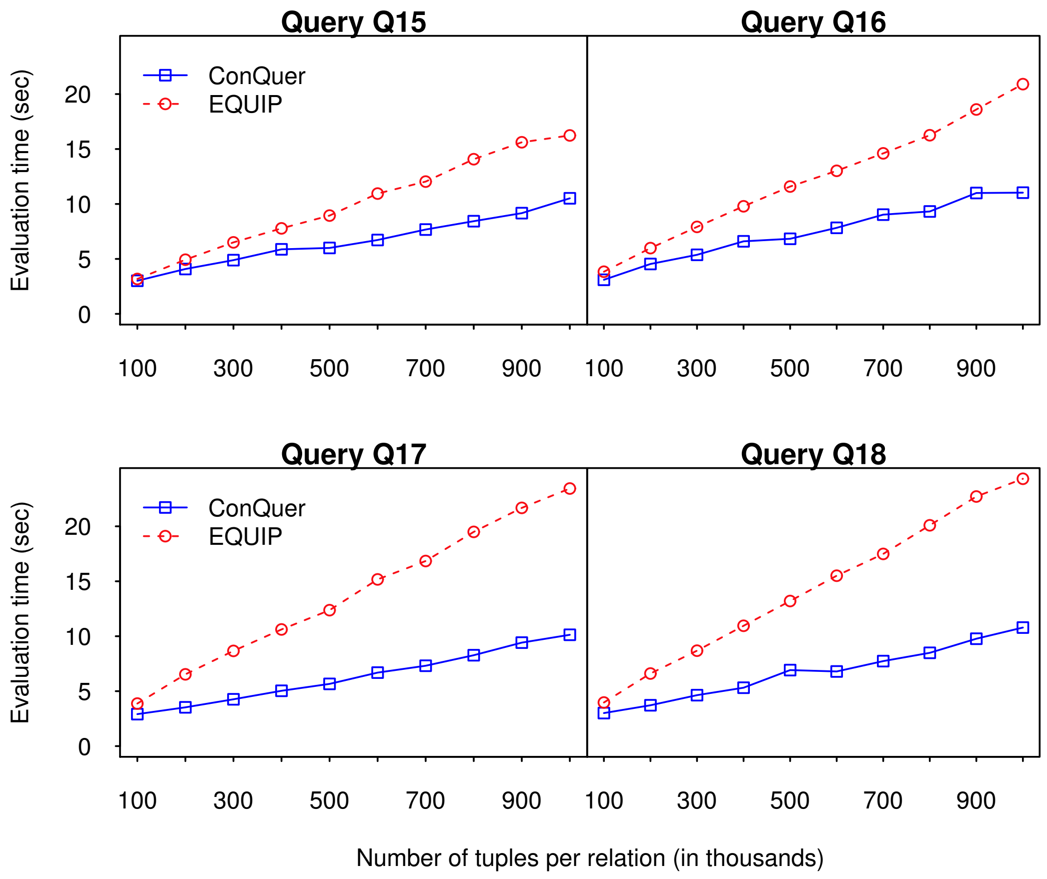


Figure 3.8: Comparison of EQUIP with ConQuer on queries Q_{15} to Q_{18}, over the database with 1 million tuples/relation.

Experiments with data and queries derived from the TPC-H benchmark

We used the data generator of the TPC-H benchmark to generate a consistent database of size 1GB. From this database we created an inconsistent database with 10% conflicts, where each key equal group has size 2. The experiments use queries derived from existing TPC-H queries Q_2, Q_3, Q_4, Q_{10}, Q_{11}, Q_{20}, Q_{21} by removing aggregates, grouping, and sub-queries. The exact queries we ran, can be found in Appendix B. Also, to avoid naming conflicts with our benchmark queries in Table 3.3, we shall henceforth denote

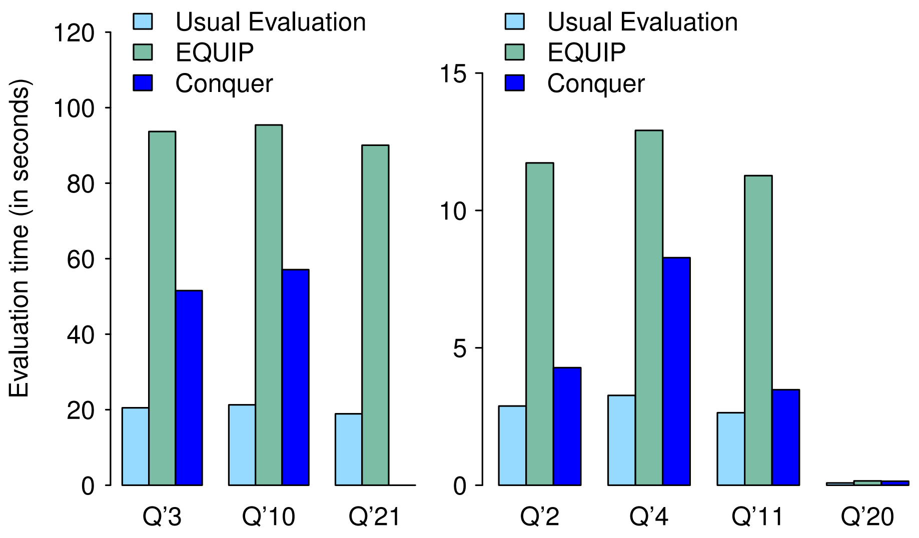


Figure 3.9: Evaluation times of the simplified TPC-H queries over "as-is" query evaluation, EQUIP, and ConQuer.

these queries as Q'_2, Q'_3, Q'_4, Q'_{10}, Q'_{11}, Q'_{20}, and Q'_{21}.

Figure 3.9 shows the comparison between EQUIP and ConQuer on the more expensive queries Q'_3, Q'_{10}, Q'_{21}, and the less expensive queries Q'_2, Q'_4, Q'_{11}, Q'_{20}. Due to a bug, ConQuer could not evaluate query Q'_{21}. The overheads incurred by EQUIP on experiments with data derived from TPC-H are approximately in the same range as the overheads we observed when evaluating the system with our own benchmark queries, and we omit the graphs here. As expected, ConQuer performs better than EQUIP, about twice as fast as EQUIP. These results reinforce our earlier findings for first-order rewritable synthetic queries and synthetic data.

Evaluation time per phase Figure 3.10 shows the evaluation times for queries $Q_1 - Q_{21}$, split into the three phases: Phase 1, Phase 2, and Phase 3. The results show that Phase 1, the pre-processing step executed over DB2, dominates the overall evaluation time. The results also show that the construction and evaluation times of the integer program takes more time for non-boolean queries. Moreover, by looking at Q_6, Q_7, Q_{13}, Q_{14}, we see that Phase 2 and Phase 3 take more time for queries that have more variables in the head of the query. In contrast, for the boolean queries, the BIP solver is not even invoked.

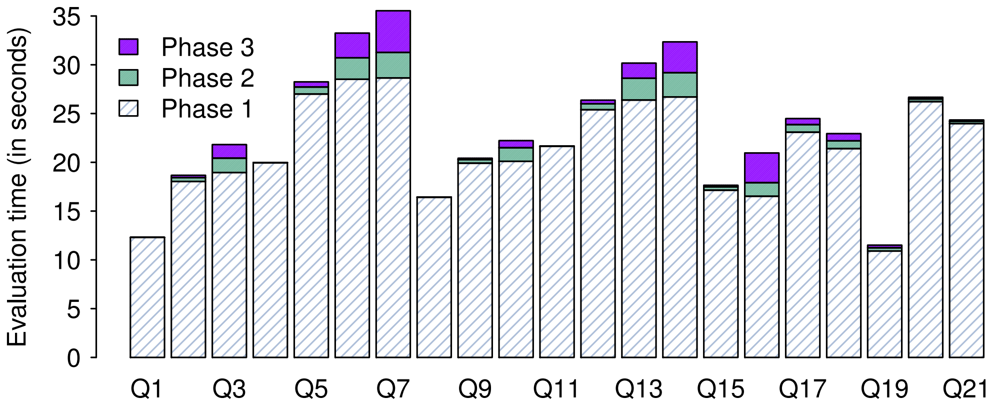


Figure 3.10: Evaluation time for Phase 1, Phase 2, and Phase 3 of EQUIP over a database with 1 million tuples/relation.

Pre-computation from the consistent part of the database Since we have approximately 10% of the database involved in conflicts, it is natural to expect that a significant portion of the consistent answers can be computed from the consistent part of the database. Hence, we expect that the optimization we have implemented in Phase 1 for computing some of the consistent answers from the consistent part of the database to play a role in reducing the overall evaluation time. Figure 3.11 shows the evaluation times of EQUIP, where part of the consistent answers are computed from the

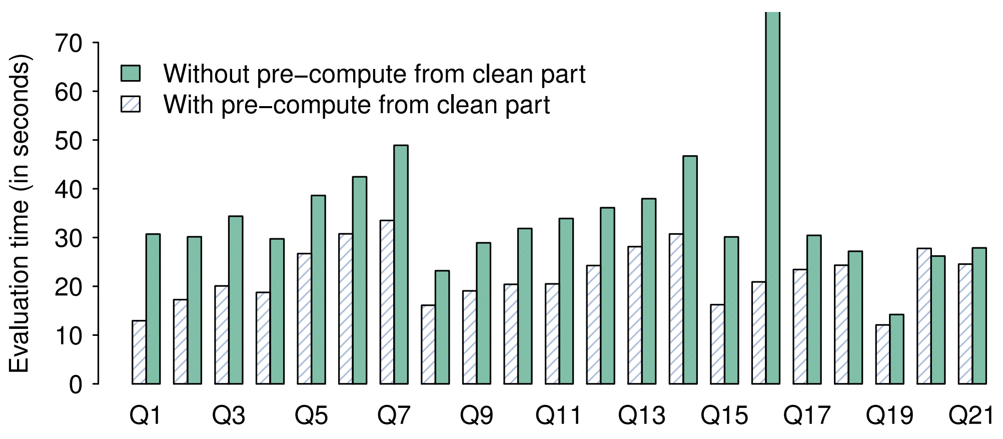


Figure 3.11: Performance of EQUIP with and without pre-computing the consistent answers from the consistent part of the database, over a database with 1 million tuples/relation.

consistent part of the database and additional answers are computed using Algorithm EliminatePotentialAnswers, versus the evaluation time when all of the consistent answers are computed using Algorithm EliminatePotentialAnswers. Our experiments show that the latter technique (without optimization), on 12 of the queries it is between 1.1 and 1.5 times slower; on 6 of the queries it is between 1.6 and 2 times slower; on 2 queries it is over 2 times slower; and only on one query it is slightly faster.

Sizes of the binary integer programs In Table 3.4, we tabulate the sizes of the binary integer programs that were constructed from a database with 1 million tuples per relation. As this table indicates, for some of the queries, large binary integer programs containing over 100,000 variables and constraints are solved. Observe that the number of variables is much smaller than the number of tuples. The reason is that the binary integer programs are built only from facts relevant to computing the additional consistent query answers. Recall that PHASE 1 retrieves only the relevant facts. The boolean queries Q_1, Q_4, Q_8 and Q_{11} evaluate to true over the consistent part of the databases used for these experiments; hence, there are no relevant facts, and no integer program is constructed. So far, we have not run into the potential limitation that the binary integer program generated may be too large to fit into main memory. This is true even for experiments conducted with data derived from TPC-H. In general, instead of building a single program for checking all the potential answers at once, we can, in fact, partition the set of potential answers into buckets and construct a "smaller" binary integer program for each bucket. As part of future work, we plan to further investigate this technique.

Query	Number of variables	Number of constraints	Query	Number of variables	Number of constraints
Q_1, Q_4, Q_8, Q_{11}	0	0	Q_{13}	148K	126K
Q_2	13K	12K	Q_{14}	170K	137K
Q_3	89K	71K	Q_{15}	9K	8K
Q_5	26K	26K	Q_{16}	109K	86K
Q_6	141K	124K	Q_{17}	31K	28K
Q_7	160K	138K	Q_{18}	53K	45K
Q_9	11K	9K	Q_{19}	20K	18K
Q_{10}	86K	64K	Q_{20}	19K	17K
Q_{12}	22K	120K	Q_{21}	16K	17K

Table 3.4: Sizes of BIP programs over a database with 1 million tuples per relation.

Varying the degree of inconsistency Figure 3.12 shows the effect of varying the degree of inconsistency (10%, 15%, 20%) with a database with 1 million tuples per relation. Our results indicate that in the worst case, the evaluation time increases linearly with the degree of inconsistency. This is not unexpected as EQUIP has to manage a larger number of relevant tuples as the degree of inconsistency increases. Not surprisingly, the degree of inconsistency affects the percentage of consistent answers out of all the potential answers. Specifically, when the degree of inconsistency is 10%, the percentage of consistent answers, depending on the query, ranges from 80% (for Q_7, Q_{14}) to 99% (for Q_{15}). When the degree of inconsistency is increased to 20%, the percentage of consistent answers ranges from 63% to 97%.

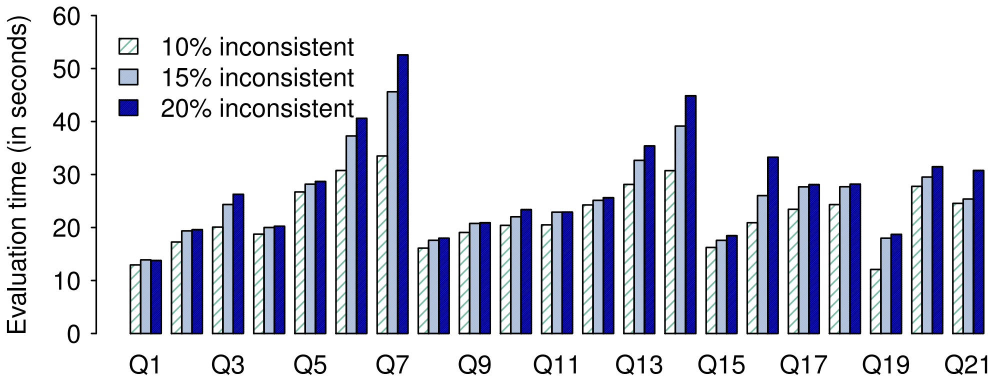


Figure 3.12: Evaluation time of EQUIP over databases with 1 million tuples per relation and different degrees of inconsistency.

Varying the size of key-equal groups In Figure 3.13 we show the effect of increasing the size of key-equal groups from 2 to 5, on a database with 1 million tuples per relation and 10% of tuples involved in violations. Figure 3.13 does not reveal any particular behavior of EQUIP with respect to the size of key-equal groups. As we increase the size of the key-equal groups, PHASE 1 becomes slightly less expensive, because there are fewer duplicated key-values that need to be materialized. Moreover, the BIP program contains the same number of variables but fewer constraints. On the other hand, when the key-equal groups have larger size, the constraints become more involved. The combined effect of these different factors, results in stable performance of EQUIP with respect to the size of key-equal groups.

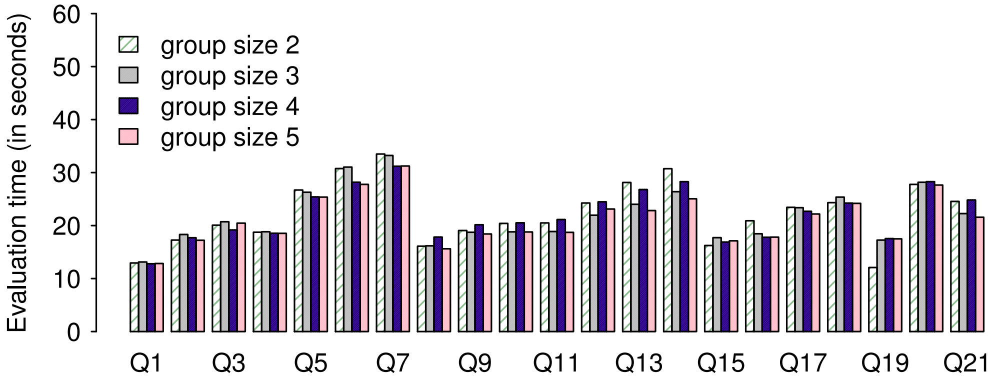


Figure 3.13: Evaluation time of EQUIP over databases with 1 million tuples per relation, degree of inconsistency 10%, and different sizes of key-equal groups.

The use of indices We have found out that the majority of the time for computing the consistent query answers goes to Phase 1. Since Phase 1 consists of evaluating SQL queries on top of the underlying DBMS, it is natural to consider making use of DBMS features, such as indices. For this reason, we also conducted experiments in which we added a number of indices. For every relation, we have created a clustered index on the key attributes. Figure 3.14 shows the evaluation of EQUIP in the presence of indices. The improved performance from using indices is clear, as for all the queries the evaluation time has decreased significantly. One may also be able to obtain better performance by using a different configuration of indices, or by a better tuning of DBMS parameters.

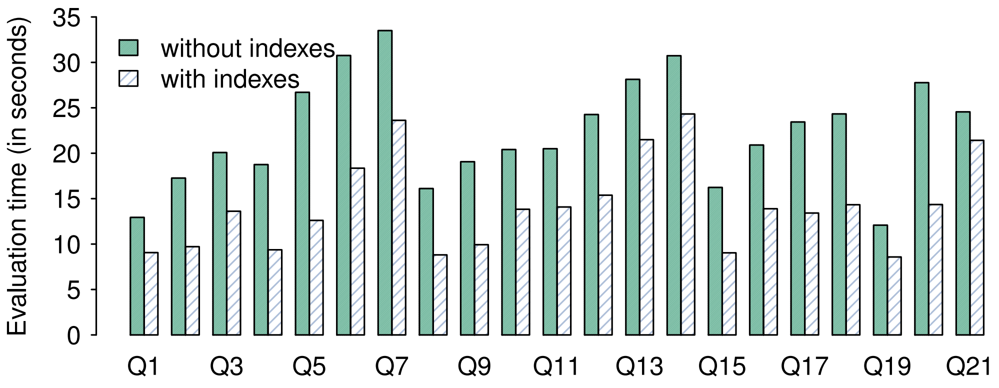


Figure 3.14: Performance of EQUIP in the presence of indices, over a database with 1 million tuples/relation.

Experiments with a cyclic query and a query with a self-join EQUIP can handle arbitrary conjunctive queries, and in particular it can handle cyclic queries and queries with self-joins, which are typically considered "problematic" with regard to the computation of consistent answers. We evaluate the following two additional queries with EQUIP:

$$Q_{22}(v) : -R_5(\underline{x}, y, z), R_6(\underline{x'}, y, z), R_{11}(\underline{x, x'}, v)$$
$$Q_{23}(z) : -R_5(\underline{x}, y, z), R_5(\underline{x'}, y, w)$$

Table 3.5: A cyclic query and a query with self-join used in our experiments.

In Table 3.5, query Q_{22} is cyclic, and query Q_{23} has a self-join. For both queries, CERTAINTY(q) is coNP-hard. For query Q_{22}, we provide a coNP-hardness proof in Appendix C.2. For query Q_{23}, later in Section 4.2 we establish that CERTAINTY(Q_{23}) is coNP-hard.

Figure 3.15 shows the performance of EQUIP on the queries Q_{22} and Q_{23}. While in theory, several difficulties are associated with computing the consistent answers to cyclic queries or queries that have repeated relation names, in practice, EQUIP does not exhibit any distinguishable characteristic in performance when evaluating these queries. In fact, the evaluation times of Q_{22} are similar to the evaluation times of other queries in our benchmark that have three atoms and one free variable. Also, the evaluation times of query Q_{23} are about the same as the evaluation times of the very similar query Q_2. The reason for such behavior of EQUIP is because our reduction to BIP is generic and treats all conjunctive queries in the same way.

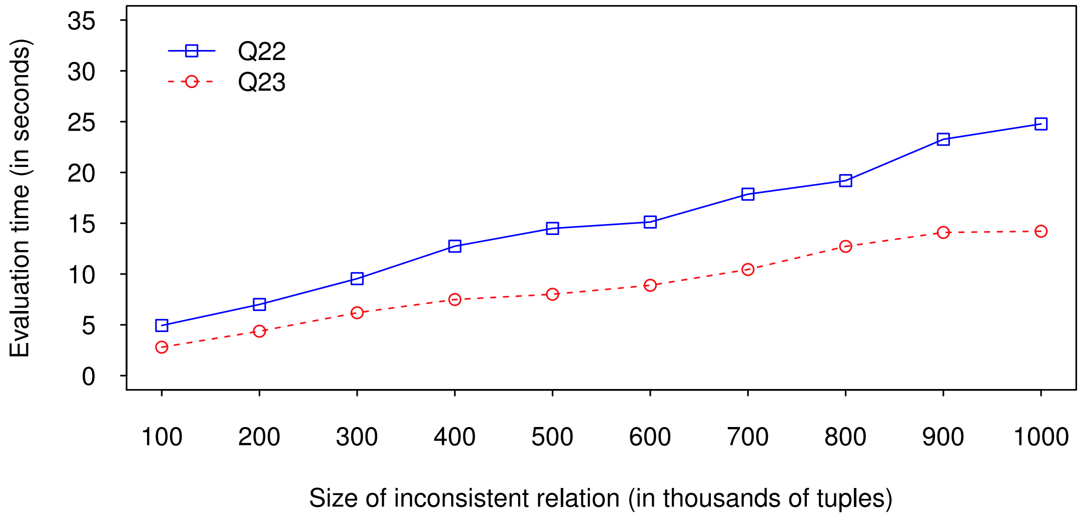


Figure 3.15: Performance of EQUIP in on queries Q_{22} and Q_{23}.

Comparison with DLV The comparison with ConsEx in Figure 3.4 reveals that EQUIP significantly outperforms ConsEx. We also pointed out that ConsEx can compute the consistent query answers to broader classes of queries and constraints than

EQUIP. One may wonder if it is possible to build better logic programs by following some strategy specific to conjunctive queries and primary key constraints, instead of the more generic strategy that ConsEx follows. For this purpose, we evaluate with DLV a different set of logic programs, generated independently from ConsEx. These programs were provided to us by Leone et al. via personal communication [49]. These programs also contain rules that internally implement our optimization for pre-computing consistent answers from the clean part of the database. The results of our experiments are summarized in Table 3.6. These programs are evaluated by DLV much more efficiently than the programs generated by ConsEx. See for instance the query Q_2 on which ConsEx does not terminate even with only 10,000 tuples per relation. Many of the queries are evaluated by DLV within a few seconds. In fact, over the database with 10,000 tuples per relation, DLV is slightly faster than EQUIP (a few hundred milliseconds faster). However, we should point out that the running times shown for DLV only include the time for evaluating the programs, but they do not include the time for parsing the query and the constraints, nor the time for generating the programs. On the other hand, the running times for EQUIP include every computation. As we go to 20,000 tuples per relation, EQUIP becomes much faster than DLV at evaluating the consistent answers. On some of the queries ($Q_2, Q_9, Q_{12}, Q_{17}, Q_{20}, Q_{21}$), the performance of DLV is particularly poor. Such behavior of DLV is in fact surprising since these queries are not any more complex than the rest of the queries. This experiment provides further evidence that as of now, our BIP-based approach for consistent query answering is far more efficient, stable and scalable compared to a DLV-based approach.

Running time of DLV and EQUIP (in seconds)

	10K		20K		30K		40K		50K	
Query	DLV	EQUIP	DLV	EQUIP	DLV	EQUIP	DLV	EQUIP	DLV	EQUIP
Q1	1.4	2.0	3.8	2.1	4.7	2.1	6.0	2.2	7.3	2.4
Q2	1.5	1.9	46.6	2.2	2hr+	2.2	2hr+	2.4	2hr+	2.5
Q3	1.7	2.0	5	2.3	6.6	2.3	8.6	2.7	10.8	3.1
Q4	1.4	2.1	4.6	2.1	5.7	2.3	7.3	2.5	8.8	2.8
Q5	1.7	2.1	5.0	2.3	6.5	2.4	8.3	2.7	10.2	3.3
Q6	2.1	2.5	5.8	2.6	7.8	2.7	10.2	2.9	12.7	3.9
Q7	2.3	2.3	6.3	2.7	8.6	2.7	11.2	3.4	14	3.5
Q8	1.4	2.1	4.2	2.2	4.7	2.2	6.1	2.3	7.1	2.5
Q9	233	2.1	873	2.1	2hr+	2.2	2hr+	2.4	2hr+	2.6
Q10	1.6	2.1	4.8	2.2	6.2	2.3	8	2.7	9.6	3.1
Q11	1.3	2.3	4.6	2.3	5.6	2.4	7.2	2.5	8.7	2.8
Q12	225	2.4	925	2.4	2hr+	2.5	2hr+	2.8	2hr+	2.9
Q13	1.9	2.3	5.5	2.5	7.2	2.7	9.4	3.1	11.5	3.4
Q14	2.1	2.3	6.0	2.4	8.0	2.6	10.2	3.1	12.7	3.5
Q15	1.4	2.1	4.3	2.1	5.3	2.1	6.7	2.2	8.3	2.4
Q16	1.6	2.1	4.8	2.3	6.1	2.5	7.9	2.7	9.8	2.9
Q17	2hr+	2.3	2hr+	2.3	2hr+	2.4	2hr+	2.7	2hr+	3.1
Q18	1.9	2.2	5.5	2.4	7.2	2.4	9.2	2.7	11.6	3.1
Q19	1.3	2.3	3.4	2.4	3.7	2.4	4.5	2.4	8.9	2.7
Q20	2hr+	2.5	2hr+	2.6	2hr+	2.7	2hr+	2.9	2hr+	3.1
Q21	2hr+	2.5	2hr+	2.6	2hr+	2.6	2hr+	2.7	2hr+	2.8

Table 3.6: Performance of DLV and EQUIP at evaluating the consistent answers of queries Q_1 to Q_{21}

Chapter 4

Data Complexity of Consistent Query Answering

In this chapter, we present our analysis on the data complexity of consistent query answering for acyclic and self-join free conjunctive queries under primary key constraints. First, we introduce some preliminary notions that are relevant to this chapter. Next, we will present a dichotomy in the complexity of CERTAINTY(q) where q is a boolean conjunctive query with two atoms and without self-joins. Later, we will present a sufficient condition for CERTAINTY(q) to be in P, and a sufficient condition for CERTAINTY(q) to be coNP-complete, where q is an arbitrary boolean conjunctive query that is acyclic and has no self-joins. Finally, we will state our conjecture for a dichotomy theorem on the complexity of CERTAINTY(q) for the class of acyclic and self-join free conjunctive queries and primary key constraints. Since our work towards a dichotomy for the complexity of CERTAINTY(q) has developed in parallel with similar efforts by

Wijsen [62], and Koutris and Suciu [46], we review this related work, and show how it compares to our results. In fact, we will argue that this related work serves as evidence that supports our dichotomy conjecture.

4.1 Preliminaries

The results in this chapter concern acyclic and self-join free conjunctive queries. Therefore, we will use the relation names to refer to the atoms in a query. Our complexity analysis builds on top of Wijsen's characterization of first-order expressibility of acyclic and self-join free conjunctive queries in [61]. Hence, we give next a succinct recount of this work. The necessary and sufficient condition for first-order expressibility is formulated on top of a graph, called the *attack* graph, that is constructed from the query and the constraints together.

Every atom R in a conjunctive query q gives rise to a functional dependency among the variables occurring in R. For example, the atom $R(\underline{x, y}, z)$ gives rise to $\{x, y\} \to z$. As a special case, the atom $R(\underline{a}, x)$, where a is a constant, gives rise to the functional dependency $\{\} \to x$.

Definition 5 *Let q be a boolean conjunctive query. Let $vars(q)$ be the set of variables occurring in q, and let $Atoms(q)$ be the set of atoms of q.*

- *$K(q)$ is the set of all functional dependencies that arise from the atoms of q. In symbols, $K(q) = \{key(R) \to vars(R) : R \in q\}$.*
- *If R is an atom of q, then R^+ denotes the attribute closure of the set $key(R)$ w.r.t.*

the set of all functional dependencies that arise in the atoms $Atoms(q) - \{R\}$. In symbols, $R^+ = \{x \in vars(q) : K(Atoms(q) - \{R\}) \models key(R) \rightarrow x\}$.

To illustrate Definition 5 with an example, let $q() : -R_1(\underline{x}, y), R_2(\underline{x, z}, y')$. Then $R_1^+ = \{x\}$ and $R_2^+ = \{x, z, y\}$. Notice that in R_2^+ we include the variable y because of the dependency $\{x\} \rightarrow y$ which holds in R_1.

Definition 6 *Let ρ be an intersection tree for a boolean conjunctive query q. The* attack graph *of ρ is the directed graph whose vertices are the atoms of q, and there is a directed edge from an atom R to an atom S if for every label L on the unique path from R to S in ρ, we have that $L \not\subseteq R^+$.*

We write $R \rightsquigarrow S$ to denote that there is a directed edge from R to S in the attack graph, and we say that R *attacks* S. A cycle of length n in the attack graph is a sequence of edges $R_0 \rightsquigarrow R_1 \rightsquigarrow ... \rightsquigarrow R_{n-1} \rightsquigarrow R_0$.

Figure 4.1 illustrates the construction of the attack graph for queries $q_1() : -R_1(\underline{x}, y), R_2(\underline{y}, z)$, $q_2() : -R_1(\underline{x}, y), R_2(\underline{y}, x)$, and $q_3() : -R_1(\underline{x}, y), R_2(\underline{x'}, y)$ from the Introduction. Because these queries have two atoms only, each of them has a unique join-tree with a single edge. For q_1, for instance, the variables shared between R_1 and R_2 are $L = \{y\}$. Moreover, $R_1^+ = \{x\}$ and $R_2^+ = \{y\}$. Since $L \not\subseteq R_1^+$, there is an edge from R_1 to R_2. However, there is no edge from R_2 to R_1 because $L \subseteq R_2^+$. One can construct the attack graphs for q_2 and q_3 in a similar way.

We are now ready to formulate the main result in [61]:

$$\begin{tikzcd} R_1(\underline{x}, y) \arrow[r, bend left] & R_2(\underline{y}, z) \end{tikzcd}$$

(a) Attack graph of query q_1

$$\begin{tikzcd} R_1(\underline{x}, y) \arrow[r, bend left] & R_2(\underline{y}, x) \arrow[l, bend left] \end{tikzcd}$$

(b) Attack graph of query q_2

$$\begin{tikzcd} R_1(\underline{x}, y) \arrow[r, bend left] & R_2(\underline{x'}, y) \arrow[l, bend left] \end{tikzcd}$$

(c) Attack graph of query q_3

Figure 4.1: Attack graphs of queries $q_1() : -R_1(\underline{x}, y), R_2(\underline{y}, z)$, $q_2() : -R_1(\underline{x}, y), R_2(\underline{y}, x)$, and $q_3() : -R_1(\underline{x}, y), R_2(\underline{x'}, y)$

Theorem 4 ([61]) *Let q be an acyclic self-join free boolean conjunctive query and let τ be a join tree for q. Then the following two statements are equivalent:*

1. CERTAINTY(q) *is first-order expressible.*

2. *The attack graph of τ is acyclic.*

To illustrate Theorem 4, query q_1 in Figure 4.1 has an acyclic attack graph; hence CERTAINTY(q_1) is first-order expressible. On the other hand, q_2 and q_3 have cyclic attack graphs; therefore, CERTAINTY(q_2) and CERTAINTY(q_3) is not first-order expressible. As mentioned in the Introduction, CERTAINTY(q_2) is in P and CERTAINTY(q_3) is coNP-complete. However, their attack graphs look exactly the same.

It was shown in [61] that if the attack graph is cyclic then it contains some cycle of length two. Hence, if the query is not first-order expressible, there is always a cycle of length two in the attack graph. The construction of the attack graph, as presented in [59], depends on the join-tree. Because a query can be described by different join-trees, one may wonder how the attack graphs constructed from the different join-trees compare with each-other. It is in fact the case, as it was proven in [61], that the attack graph of a query is always the same, regardless of the join-tree.

The results that we present in this chapter apply to boolean self-join free conjunctive queries that may contain constants. These results can be straightforwardly extended to non-boolean queries as well. Specifically, let q be a query of arity k, for some $k \geq 1$, and let I be an instance. A k-tuple t with values from the active domain of I is in the consistent answers of q on I if and only for every repair r of I, we have that t is in $q(r)$. This is the same as the boolean query $q(t)$ being true in every repair r of I, where $q(t)$ is the query obtained from q by substituting the free variables of q (i.e., the variables that are not existentially quantified) with corresponding constants from t.

4.2 A Dichotomy for Conjunctive Queries with Two Atoms

We have proven a dichotomy theorem for CERTAINTY(q), where q is a boolean and self-join free conjunctive query with two atoms, such that CERTAINTY(q) is not first-order expressible. Intermediate results were initially published in [43], and the complete result was later published in [42]. The motivation for initially focusing on this restricted class came from the fact that even for queries with two atoms, there was no clear understanding or intuition regarding the complexity of CERTAINTY(q), and the dichotomy was not trivial. The three examples in the Introduction involve conjunctive queries with exactly two atoms; in fact, most of the conjunctive queries analyzed in [32, 60] have exactly two atoms. Proving the dichotomy even for such a restricted class, was an important step towards tackling the general dichotomy. In fact, the proof techniques that we use to prove sufficient conditions for tractability and intractability, which we will present in Section 4.3 and Section 4.4, are fundamentally the same as the ones used to prove the dichotomy for queries with two atoms.

More precisely, we have proven the following result for CERTAINTY(q), where q is a boolean conjunctive query with exactly two atoms[1], without self-joins, and such that CERTAINTY(q) is not first-order expressible.

Theorem 5 *Let q be a self-join free boolean conjunctive query with two atoms R_1 and R_2 such that* CERTAINTY(q) *is not first-order expressible. Then either* CERTAINTY(q) *is in P or* CERTAINTY(q) *is coNP-complete. Moreover, the complexity of* CERTAINTY(q) *is determined by the following criterion:*

1. *If $key(R_1) \cup key(R_2) \subseteq L$, then* CERTAINTY($q$) *is in P;*
2. *If $key(R_1) \cup key(R_2) \not\subseteq L$, then* CERTAINTY($q$) *is coNP-complete,*

where R_1, R_2 are the two atoms of q, and L is the set of variables shared by R_1 and R_2.

To illustrate Theorem 5, let us consider again the queries $q_2() : -R_1(\underline{x}, y), R_2(\underline{y}, x)$ and $q_3() : -R_1(\underline{x}, y), R_2(\underline{x'}, y)$ from the Introduction. We know that CERTAINTY(q_2) and CERTAINTY(q_3) is not first-order expressible. For the query $q_2() : -R_1(\underline{x}, y) \wedge R_2(\underline{y}, x)$, we have that $key(R_1) = \{x\}$, $key(R_2) = \{y\}$, and $L = \{x, y\}$; since $key(R_1) \cup key(R_2) \subseteq L$, it follows that CERTAINTY(q_2) is in P. In contrast, for the query $q_3() : -R_1(\underline{x}, y) \wedge R_2(\underline{x'}, y))$, we have that $key(R_1) = \{x\}$, $key(R_2) = \{x'\}$, and $L = \{y\}$; since $key(R_1) \cup key(R_2) \not\subseteq L$, it follows that CERTAINTY(q_3) is coNP-complete. If

[1]Every query with two atoms is trivially acyclic.

CERTAINTY(q) is first-order expressible, then CERTAINTY(q) is in P. Consequently, Theorem 5 yields the following dichotomy theorem for self-join free boolean conjunctive queries with exactly two atoms.

Corollary 1 *If q is a self-join free boolean conjunctive query with exactly two atoms, then either* CERTAINTY(q) *is in P or* CERTAINTY(q) *is coNP-complete.*

Every self-join free conjunctive query with two atoms is acyclic and has only one join tree, namely, a single edge that connects the two atoms. Therefore, if the attack graph is cyclic, it is a cycle of length 2, which arises precisely when $L \not\subseteq R_1^+$ and $L \not\subseteq R_2^+$. Thus, Theorem 4 yields the following corollary.

Corollary 2 *Let q be a self-join free Boolean conjunctive query with two atoms R_1 and R_2, and let L be the set of variables shared by R_1 and R_2. Then the following two statements are equivalent:*

1. CERTAINTY(q) *is first-order rewritable.*
2. $L \subseteq R_1^+$ *or* $L \subseteq R_2^+$.

Furthermore, by combining Theorem 5 with Corollary 2, we obtain the following *trichotomy* result.

Corollary 3 *Let q be a self-join free boolean conjunctive query with two atoms R_1 and R_2, and let L be the set of variables shared by R_1 and R_2. Then, the following two statements hold:*

1. *If $L \subseteq R_1^+$ or $L \subseteq R_2^+$, then* CERTAINTY(q) *is first-order rewritable.*

2. *If $L \not\subseteq R_1^+$, $L \not\subseteq R_2^+$, and $key(R_1) \cup key(R_2) \subseteq L$, then* CERTAINTY$(q)$ *is in P but is not first-order rewritable.*

3. *If $L \not\subseteq R_1^+$, $L \not\subseteq R_2^+$, and $key(R_1) \cup key(R_2) \not\subseteq L$, then* CERTAINTY$(q)$ *is coNP-complete.*

Before embarking on the proof of Theorem 5, we describe briefly our strategy. Let q be a self-join free boolean conjunctive query with two atoms such that CERTAINTY(q) is not first-order expressible. In Section 4.2.1, we prove the intractability side of the dichotomy, that is, we show that if the query q is such that $key(R_1) \cup key(R_2) \not\subseteq L$, then CERTAINTY$(q)$ is coNP-hard. As a stepping stone, we show that CERTAINTY(q') is coNP-hard, where q' is the query $q'() : -S_1(\underline{x, z}, y), S_2(\underline{y}, x)$; this is done via a polynomial-time reduction from MONOTONE SAT. After this, we show that if q is a query such that $key(R_1) \cup key(R_2) \not\subseteq L$, then CERTAINTY$(q')$ can be reduced in polynomial time to CERTAINTY(q).

In Section 4.2.2, we prove the tractability side of the dichotomy, that is, we show that if the query q is such that $key(R_1) \cup key(R_2) \subseteq L$, then CERTAINTY$(q)$ is in P. To this extent, we introduce the notion of the *conflict-join* graph and show that CERTAINTY(q) can be reduced in polynomial time to the problem of finding an independent set of a certain size in a *conflict-join* graph. In the general case, the problem of finding an independent set of a certain size in a given graph is NP-complete. However, there are families of graphs on which this problem can be solved in polynomial time. One such family is the class of all *claw-free* graphs. We show that the *conflict-join* graph of queries with two atoms that satisfy the condition $key(R_1) \cup key(R_2) \subseteq L$ is always *claw-free*.

4.2.1 The Intractability Side of the Dichotomy

We begin by observing that if q is a query such that CERTAINTY(q) is not first-order rewritable and moreover, $key(R_1) \cup key(R_2) \not\subseteq L$, then the variables of q exhibit a particular pattern.

Proposition 3 *Let q be a self-join free boolean conjunctive query with two atoms such that* CERTAINTY(q) *is not first-order rewritable. Let R_1, R_2 be the two atoms of q, and let L be the set of variables shared by R_1 and R_2. Then the following hold:*

1. *There exist four variables u, v, w, w' with the property that $u \in key(R_1) \setminus key(R_2)$, $v \in key(R_2) \setminus key(R_1)$, $w \in L \setminus key(R_1)$, and $w' \in L \setminus key(R_2)$.*
2. *If, in addition, $key(R_1) \cup key(R_2) \not\subseteq L$, then u can be chosen to also satisfy $u \in key(R_1) \setminus L$ or v can be chosen to also satisfy $v \in key(R_2) \setminus L$.*

PROOF. Since CERTAINTY(q) is not first-order rewritable, Corollary 2 tells that $L \not\subseteq R_1^+$ and $L \not\subseteq R_2^+$. We claim that $key(R_1) \not\subseteq key(R_2)$ and $key(R_2) \not\subseteq key(R_1)$. Indeed, if $key(R_1) \subseteq key(R_2)$, then $key(R_1) \subseteq R_2^+$ and also $nkey(R_1) \subseteq R_2^+$. Consequently, $L \subseteq R_2^+$, which contradicts the hypothesis. A similar argument shows that $key(R_2) \not\subseteq key(R_1)$. Thus, there are variables u and v such that $u \in key(R_1) \setminus key(R_2)$ and $v \in key(R_2) \setminus key(R_1)$. Since $L \not\subseteq R_1^+$ and $key(R_1) \subseteq R_1^+$, there is a variable w such that $w \in L \setminus key(R_1)$. Similarly, since $L \not\subseteq R_2^+$ and $key(R_2) \subseteq R_2^+$, there is a variable w' such that $w' \in L \setminus key(R_2)$.

Assume that, in addition, $key(R_1) \cup key(R_2) \not\subseteq L$ holds. This means that $key(R_1) \not\subseteq L$ or $key(R_2) \not\subseteq L$. In the first case, there exists a variable $u \in key(R_1) \setminus L$ (hence, also $u \in key(R_1) \setminus key(R_2)$). In the second case, there exists a variable $v \in key(R_2) \setminus L$ (hence, also $v \in key(R_2) \setminus key(R_1)$). $\square$

Let q' be the query $q'() :- S_1(\underline{x, z}, y), S_2(\underline{y}, x)$. Corollary 2 implies that CERTAINTY(q) is not first-order rewritable. Moreover, we have that $key(S_1) \cup key(S_2) \not\subseteq L$. It is easy to verify directly that the variables of q' exhibit the pattern described in Proposition 3. Specifically, the role of u is played by z, the roles of both v and w are played by y, and the role of w' is played by x.

Lemma 1 *Let q' be the query $q'() :- S_1(\underline{x, z}, y), S_2(\underline{y}, x)$. Then* CERTAINTY($q'$) *is coNP-hard.*

PROOF. We will reduce MONOTONE SAT to CERTAINTY(q') in polynomial time. Let φ be a boolean formula in conjunctive normal form such that each clause has either positive literals only (positive clause) or negative literals only (negative clause); without loss of generality, assume that each variable of φ occurs in some positive clause and in some negative clause. Construct an instance I over the schema of q' as follows:

- For every positive clause c_i and variable p in it, generate a fact $S_1(1, c_i, p)$ in I.

- For every negative clause c_j and variable p in it, generate a fact $S_1(0, c_j, p)$ in I.

- For every variable p, generate two facts $S_2(p, 0)$ and $S_2(p, 1)$ in I.

We will now prove that φ is satisfiable if and only if there is a repair of I that does not satisfy q'.

($\Rightarrow$) Assume first that there exists a satisfying assignment θ for φ. Construct the following instance r:

- For every positive clause c_i of φ, choose a variable p in c_i such that $\theta(p) = 1$. Add the fact $S_1(1, c_i, p)$ to r.

- For every negative clause c_j of φ, choose a variable p in c_j such that $\theta(p) = 0$. Add the fact $S_1(0, c_j, p)$ to r.

- For every variable p of φ, if $\theta(p) = 1$, then add the fact $S_2(p, 0)$ to r; otherwise, add $S_2(p, 1)$ to r.

To see that r is a repair, notice that for every clause in φ we add a single S_1-fact in r, and for every variable p, we add a single S_2-fact in r. For a variable p, either $\theta(p) = 0$ or $\theta(p) = 1$. If $\theta(p) = 0$ then in r there exists no fact of the form $S_1(0, \_, p)$, and no fact $S_2(p, 0)$. If $\theta(p) = 1$ then in r there exists no fact of the form $S_1(1, \_, p)$, and no fact $S_2(p, 1)$. In either case, $r \not\models q$.

($\Leftarrow$) Next, assume that r is a repair of I such that $r \not\models q'$. Let θ be the following truth assignment.

- For every fact $S_1(1, c_i, p)$ in r, set $\theta(p) = 1$.

- For every fact $S_1(0, c_j, p)$ in r, set $\theta(p) = 0$.

- For every variable p for which there is no fact $S_1(\_,\_,p)$ in r, assign to p value 0 or value 1 arbitrarily.

From $r \not\models q$, it follows that there cannot exist two facts $S_1(1, c_i, p)$ and $S_1(0, c_j, p)$. Therefore, θ is well-defined. Moreover, since r is a repair, every clause is satisfied by θ because for every clause c there exists a fact $S_1(1, c, p)$. □

We will use the following terminology and notation. If a and b are constants, then $a \cdot b$ is a new constant encoding the pair (a, b) in a unique way; in other words, the function that maps every pair of constants (a, b) to $a \cdot b$, is injective.

Lemma 2 *Let q be a self-join free Boolean conjunctive query with two atoms such that* CERTAINTY(q) *is not first-order expressible. Let R_1, R_2 be the two atoms of q, and let L be the set of variables shared by R_1 and R_2. If $key(R_1) \cup key(R_2) \not\subseteq L$, then* CERTAINTY$(q)$ *is coNP-hard.*

PROOF. Let q' be the query $\exists x, y, z.S_1(\underline{x, z}, y) \wedge S_2(\underline{y}, x)$ of Lemma 1. We will show that CERTAINTY(q') can be reduced to CERTAINTY(q) in polynomial time. To this effect, given an instance I' over the schema of q', we will construct an instance I over the schema of q such that there is a repair of I' on which q' is false if and only if there is a repair of I on which q is false.

Assume that the two atoms of q are $R_1(s_1, \ldots, s_n)$ and $R_2(t_1, \ldots, t_m)$, where each s_i and each t_j is a variable or a constant (clearly, these variables need not be pairwise distinct). Let V be the set of variables occurring in q. From Proposition 3, there are variables u, v, w, w' such that $u \in key(R_1) \setminus key(R_2)$, $v \in key(R_2) \setminus key(R_1)$,

$w \in L \setminus \mathit{key}(R_1)$, $w' \in L \setminus \mathit{key}(R_2)$. Moreover, $u \in \mathit{key}(R_1) \setminus L$ or $v \in \mathit{key}(R_2) \setminus L$ holds. Assume that $u \in \mathit{key}(R_1) \setminus L$ (the other case is similar). Let $P = \{u, v, w, w'\}$, let $Q = V \setminus P$, and c be a fixed constant. We are now ready to describe the construction of the instance I from I'. The intuition behind this construction is that the variable u in q plays the role of the variable z in q', the variable w' in q plays the role of the variable x in q', while the variables v and w in q play the role of the variable y in q'.

Consider first the atom $R_1(s_1, \ldots, s_n)$ of q. Every fact $S_1(a_1, a_3, a_2)$ of I' generates a fact $R_1(b_1, \ldots, b_n)$ of I, where each b_i is defined as follows:

1. If $s_i = u$, then $b_i = a_1 \cdot a_3$.
2. If $s_i = v$, then $b_i = a_2$.
3. If $s_i = w = w'$, then $b_i = a_1 \cdot a_2$.
4. If $s_i = w$ and $w \neq w'$, then $b_i = a_2$.
5. If $s_i = w'$ and $w' \neq w$, then $b_i = a_1$.
6. If s_i is a constant, then $b_i = s_i$.
7. In all other cases, $b_i = c$.

Next, consider the atom $R_2(t_1, \ldots, t_m)$ of q. Every fact $S_2(a_2, a_1)$ of I' generates a fact $R_2(b_1, \ldots, b_m)$ of I, where each b_i is defined by the preceding conditions 2 to 7 and with t_i in place of s_i. Note that the first condition is not applicable because $u \in \mathit{key}(R_1) \setminus L$, hence u cannot be among the variables occurring in the atom $R_2(t_1, \ldots, t_m)$.

In the preceding construction, each b_i is defined in a unique way. The reason is that u is different from v, w, and w', and also v is different from w'; thus, no s_i can meet two of the conditions 1 to 7 at the same time.

Let f be an S_i-fact of I' and let g be an R_i-fact of I, $i = 1, 2$. We write $f \rightrightarrows g$ to denote that g has been generated by f in the way described above. Thus, $I = \{g$: there is a fact f of I' such that $f \rightrightarrows g\}$. The preceding construction ensures two important properties that we now state and prove.

Property 1 *For $i = 1, 2$, let f_1, f_2 be S_i-facts of I' and let g_1, g_2 be R_i-facts of I such that $f_1 \rightrightarrows g_1$ and $f_2 \rightrightarrows g_2$. The following statements are equivalent.*

1. *The facts f_1 and f_2 are key-equal.*

2. *The facts g_1 and g_2 are key-equal.*

To verify that Property 1 holds, assume first that f_1, f_2 are S_1-facts and that g_1, g_2 are R_1-facts. Assume that $g_1 = R_1(b_1, \ldots, b_n)$ and $g_2 = R_1(b'_1, \ldots, b'_n)$. If f_1 and f_2 are key equal, then they must be of the form $S_1(a_1, a_3, a_2)$ and $S_1(a_1, a_3, a'_2)$, respectively. The preceding construction implies that the values of the keys of R_1-facts depend only on the value of the variable u or only on the values of the variables u and w', provided $w' \neq w$ (if $w' = w$, then $w' \notin key(R_1)$). If $s_i = u$, then, by construction, we have that $b_i = a_1 \cdot a_3 = b'_i$; furthermore, if $w' \neq w$, then, by construction, we have that $b_i = a_1 = b'_i$. Consequently, g_1 and g_2 are key-equal facts. For the other direction, assume that the facts g_1 and g_2 are key-equal. Assume that $f_1 = S_1(a_1, a_3, a_2)$ and $f_2 = S_1(a'_1, a'_3, a'_2)$. Since $u \in key(R_1)$, there is some i such that $u = s_i$, hence $b_i = b'_i$.

Furthermore, by construction, we have that $b_i = a_1 \cdot a_3$ and $b'_i = a'_1 \cdot a'_3$. Consequently, $a_1 \cdot a_3 = a'_1 \cdot a'_3$, which implies that $a_1 = a'_1$ and $a_3 = a'_3$. Thus, f_1 and f_2 are key-equal facts. A similar argument shows that Property 1 holds also when f_1, f_2 are S_2-facts and g_1, g_2 are R_2-facts.

Property 2 *For $i = 1, 2$, if f_1, f_2 are S_i-facts of I' and g is an R_i-fact of I such that $f_1 \rightrightarrows g$ and $f_2 \rightrightarrows g$, then $f_1 = f_2$.*

To verify that Property 2 holds, assume first that $f_1 = S_1(a_1, a_3, a_2)$, $f_2 = S_1(a'_1, a'_3, a'_2)$, and $g = R_1(b_1, \ldots, b_n)$. By Property 1, the facts f_1 and f_2 are key-equal, hence $a_1 = a'_1$ and $a_3 = a'_3$. Since $w \in L$, there is some i such that $s_i = w$. If $w = w'$, then, by construction, $a_1 \cdot a_2 = b_i = a_1 \cdot a'_2$, hence $a_2 = a'_2$. If $w \neq w'$, then $a_2 = b_i = a'_2$. In either case, we have that $a_2 = a'_2$ and so $f_1 = f_2$. A similar argument shows that Property 2 holds also for the case in which f_1, f_2 are S_2-facts and g is an R_2-fact.

We continue with the proof of the lemma. We will show that there is a repair of I' on which q' is false if and only if there is a repair of I on which q is false.

Assume that r' is a repair of I' such that $r' \not\models q'$. Let $r = \{g \in I :$ there is a fact $f \in r'$ such that $f \rightrightarrows g\}$. We claim that r is a repair of I such that $r \not\models q$.

Properties 1 and 2 imply that r is a repair of I. Indeed, to show that r is a consistent instance, let g_1, g_2 be two key-equal facts of r. Let f_1, f_2 be two facts of r' such that $f_1 \rightrightarrows g_1$ and $f_2 \rightrightarrows g_2$. Property 1 implies that the facts f_1 and f_2 are key equal. Since r is a consistent instance, it follows that $f_1 = f_2$, hence $g_1 = g_2$. To show that r is a maximal consistent subinstance of I, let g be a fact of I such that $r \cup \{g\}$ is consistent. Let f be a fact of I' such that $f \rightrightarrows g$. We claim that $r' \cup \{f\}$ is consistent. Indeed, assume that f' is a fact of r' such that f and f' are key-equal, and let $g' \in r$ be such that $f' \rightrightarrows g'$. By Property 1, we have that g and g' are key-equal facts, hence (since $r \cup \{g\}$ is consistent) $g = g'$. Property 2 implies that $f' = f$, hence $g \in r$; this completes the proof that r is a repair of I.

Next, we show that r does not satisfy q. Towards a contradiction, assume that $R_1(b_1, \ldots, b_n)$ and $R_2(b'_1, \ldots, b'_m)$ are two facts of r that satisfy q. Let $S_1(a_1, a_3, a_2)$ and $S_2(a'_2, a'_1)$ be two facts of r' such that $S_1(a_1, a_3, a_2) \rightrightarrows R_1(b_1, \ldots, b_n)$ and $S_2(a'_2, a'_1) \rightrightarrows R_2(b'_1, \ldots, b'_m)$. Consider the variables w, w' and recall that $w \in L$ and $w' \in L$. Let i and j be such that $s_i = w$ and $t_j = w$. We distinguish two cases. If $w = w'$, then $b_i = a_1 \cdot a_2$ and $b'_j = a'_1 \cdot a'_2$. Since the facts $R_1(b_1, \ldots, b_n)$ and $R_2(b'_1, \ldots, b'_m)$ satisfy q, we have that $b_i = b'_j$, hence $a_1 = a'_1$ and $a_2 = a'_2$, which implies that the facts $S_1(a_1, a_3, a_2)$ and $S_2(a'_2, a'_1)$ satisfy q', a contradiction. If $w \neq w'$, then $b_i = a_1$ and $b'_j = a'_1$. Since $b_i = b'_j$, we have that $a_1 = a'_1$. Furthermore, let k and l be such that $s_k = w'$ and $t_l = w'$. Then $b_k = a_2$ and $b'_l = a'_2$. Since $b_k = b'_l$, we have that $a_2 = a'_2$, which implies that the facts $S_1(a_1, a_3, a_2)$ and $S_2(a'_2, a'_1)$ satisfy q', a contradiction.

In the other direction, assume that r is a repair of I such that $r \not\models q$. Let $r' = \{f \in I' : \text{there is a fact } g \in r \text{ such that } f \rightrightarrows g\}$. We claim that r' is a repair of I' such that $r' \not\models q'$.

As before, Properties 1 and 2 imply that r' is a repair of I'. Indeed, if f_1 and f_2 are two key-equal facts of r', then, by Property 1, the facts g_1 and g_2 of r are key-equal, where $f_1 \rightrightarrows g_1$ and $f_2 \rightrightarrows g_2$. It follows that $g_1 = g_2$ and so, by Property 2, we have that $f_1 = f_2$. Similarly, if $r' \cup \{f\}$ is consistent and $f \rightrightarrows g$, then $r \cup \{g\}$ is consistent, hence $g \in r$ and so $f \in r'$. Finally, we show that r' does not satisfy q'. Towards a contradiction, assume that $S_1(a_1, a_3, a_2)$ and $S_2(a_2, a_1)$ are two facts of r' that satisfy q'. Let g_1 and g_2 be the facts of r such that $S_1(a_1, a_3, a_2) \rightrightarrows g_1$ and $S_2(a_2, a_1) \rightrightarrows g_2$. By the construction of I from I', and because $u \notin L$, we have that the facts g_1 and g_2 of r agree on all values corresponding to variables in L. Consequently, the facts g_1 and g_2 satisfy q, contrary to the hypothesis. This completes the proof of the lemma. $\square$

As an illustration of Lemma 2, it follows that CERTAINTY(q_3) is coNP-hard, where $q_3() : -R_1(\underline{x}, y), R_2(\underline{x'}, y)$ is the query mentioned in the Introduction. In addition, CERTAINTY(q) is coNP-hard if q is one of the following queries:

- $q() : -R_1(\underline{x, z}, x', y), R_2(\underline{x'}, x, y)$
- $q() : -R_1(\underline{x, w}, z, y), R_2(\underline{x, z}, y)$
- $q() : -R_1(\underline{x, z}, y, w), R_2(\underline{y}, x, w)$

4.2.2 The Tractability Side of the Dichotomy

In this section, we introduce the notion of the *conflict-join graph* and use it to study when CERTAINTY(q) is tractable, where q is a self-join free boolean conjunctive query with two atoms. As before, we assume that there is one key constraint for each relation symbol.

Definition 7 *Let q be a self-join free boolean conjunctive query with two atoms. If I is an instance, then the* conflict-join graph *$H_{I,q} = (V, E)$ is defined as follows:*

- *The set V of the nodes of $H_{I,q}$ consists of all* facts *of I.*
- *For every pair of key-equal facts, add an edge in E connecting these two facts.*
- *For every pair of facts that form a minimal witness to q, add an edge in E connecting these two facts.*

For every fixed query q, the size of the conflict-join graph $H_{I,q}$ is polynomial (in fact, quadratic) in the size of the instance I. If D is a set of pairwise key-equal facts of I, then every two distinct elements of D are key-equal, which implies that D induces a clique in $H_{I,q}$. A maximal set of pairwise key-equal facts of I must contain all facts that are key-equal to one of its members; moreover, if D and D' are distinct key-equal groups, then $D \cap D' = \emptyset$. Consequently, the set V of nodes of $H_{I,q}$ can be partitioned into pairwise disjoint sets $V_1, \ldots, V_n$ such that each V_i is a maximal set of key-equal facts of I. Also, by construction, the set E of edges of $H_{I,q}$ can be partitioned into two disjoint sets E_1 and E_2, where E_1 consists of all edges whose endpoints form a conflict in I, and E_2 consists of all edges whose endpoints join in the query q.

In what follows, we will establish a connection between the existence of a maximum independent set of a particular size in the conflict-join graph $H_{I,q}$ and the existence of a repair r of I such that $r \not\models q$. Recall that an *independent set* in a graph G is a set of nodes with no edges between them. A *maximum independent set* is an independent set of maximum cardinality. The *independent set number* $\alpha(G)$ of a graph G is the cardinality of a maximum independent set of G.

We now focus on the independent set number $\alpha(H_{I,q})$ of the conflict-join graph associated with an instance I. It is easy to see that $\alpha(H_{I,q}) \leq n$, where n is the number of the maximal sets $V_1, \ldots, V_n$ of pairwise key-equal facts of I. Indeed, this holds because each V_i induces a clique in $H_{I,q}$, so an independent set in $H_{I,q}$ can contain at most one node from each V_i, $1 \leq i \leq n$.

Example 6 Let q_2 be the query $q_2() : -R_1(\underline{x}, y), R_2(\underline{y}, x)$ from the Introduction. Consider the instance $I = \{R_1(a,b), R_1(a,b'), R(a,b''), R_1(a',b),\ R_2(b,a), R_2(b,a')\}$. Figure 4.2 depicts the conflict-join graph H_{I,q_2}. Note that H_{I,q_2} is partitioned into three key-equal groups. Note also that the set $r = \{R_1(a,b'), R_1(a',b), R_2(b,a)\}$ has size three and is a maximum independent set of H_{I,q_2}. Furthermore, viewed as an instance, r is a repair of I and $r \not\models q_2$. The next lemma tells that this is no accident. □

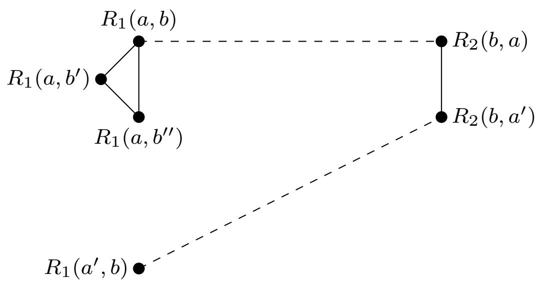


Figure 4.2: The conflict-join graph H_{I,q_2} for the query and instance of Example 6. Edges drawn as continuous lines connect pairs of facts that conflict; edges drawn as dashed lines connect facts that together satisfy q.

Lemma 3 *Assume that q is a self-join free boolean conjunctive query with two atoms and I is an instance. Let $H_{I,q}$ be the conflict-join graph associated with I and q, let*

$\alpha(H_{I,q})$ be the independent set number of $H_{I,q}$, and let n be the number of key-equal groups of facts in I. Then the following statements are equivalent:

1. *There is a repair r of I such that $r \not\models q$.*

2. *$\alpha(H_{I,q}) = n$.*

PROOF. Assume first that r is a repair of I such that $r \not\models q$. Let M be the set of all facts of r. We claim that M is an independent set in $H_{I,q}$ and has size n. To see that M is an independent set in $H_{I,q}$, consider two distinct facts f_1 and f_2 of r. If they involve the same relation symbol, then they cannot be key-equal because r is a consistent instance, hence there is no edge between them in $H_{I,q}$. If they involve different relation symbols, then they cannot join in q because $r \not\models q$; hence there is no edge between them in $H_{I,q}$. To see that M has size n, notice that, since r is a repair of I, it must contain one fact of each different key value, which means that r must contain one fact from each of the n key-equal groups of I. Since $\alpha(H_{I,q}) \leq n$, it follows that $\alpha(H_{F,I,q}) = n$.

For the other direction, assume that M is an independent set of $H_{I,q}$ of size n. Let r be the sub-instance of I formed by the facts of M. We claim that r is a repair of I such that $r \not\models q$. Indeed, since M is an independent set of $H_{I,q}$, we have that r is consistent and also $r \not\models q$. Also, since M is of size n, we have that r must contain a fact of each different key value, hence r is a maximal consistent sub-instance of I. $\square$.

The proof of Lemma 3 actually establishes something stronger, namely, that the repairs of I that falsify q are precisely the independent sets of $H_{I,q}$ of size n.

It is well known that the problem of computing the independent set number of a given graph is NP-hard [34]. However, it is also known that there are restricted classes of graphs for which this problem is solvable in polynomial time. In particular, this holds true for *claw-free* graphs, *chordal* graphs, and *perfect* graphs. Claw-free graphs will turn out to be of particular interest to us. A graph is *claw-free* if it does not contain a claw as an induced subgraph, where the *claw* is the complete bipartite graph $K_{1,3}$ (see Figure 4.3). Equivalently, a graph is claw-free if no node has three pairwise non-adjacent neighbors. Claw-free graphs form a broad class of graphs that enjoy good algorithmic properties. In particular, a polynomial-time algorithm for computing the independent set number on claw-free graphs was given by Minty [52].

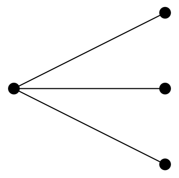


Figure 4.3: The claw graph $K_{1,3}$

Lemma 4 *Assume that q is a self-join free Boolean conjunctive query with exactly two atoms. Let R_1, R_2 be the two atoms of q, and let L be the set of variables shared by R_1 and R_2. If $key(R_1) \cup key(R_2) \subseteq L$, then, for every instance I, the conflict-join graph $H_{I,q}$ is claw-free. Consequently, if $key(R_1) \cup key(R_2)) \subseteq L$, then* CERTAINTY$(q)$ *is in P.*

PROOF. Let I be an instance. We first observe the following regarding the conflict-join graph $H_{I,q}$.

- If f_1, f_2, f_3 are three facts of I such that (f_1, f_2) and (f_1, f_3) are edges in E_1, then (f_2, f_3) is also an edge in E_1. Indeed, since (f_1, f_2) and (f_1, f_3) are in E_1, it follows that f_1 is key-equal to both f_2 and f_3; hence, f_2 is key-equal to f_3, which implies that (f_1, f_3) is an edge in E_1.

- If f_1, f_2, f_3 are three facts of I such that (f_1, f_2) and (f_1, f_3) are edges in E_2, then (f_2, f_3) is an edge in E_1. To see this, we distinguish two cases, depending on whether f_1 is an R_1-fact or an R_2-fact. Assume first that f_1 is an R_1-fact. Then, f_2 and f_3 must be R_2-facts. Since f_1 and f_2 satisfy q, they must agree on all values corresponding to variables in L. Given that $\mathit{key}(R_2) \subseteq L$, we have that f_1 and f_2 agree on all values corresponding to variables in $\mathit{key}(R_2)$. Similarly, f_1 and f_3 agree on all values corresponding to variables in $\mathit{key}(R_2)$. It follows that f_2 and f_3 are key-equal. The argument in the case that f_1 is an R_2-fact is similar.

We now prove that the conflict-join graph $H_{I,q}$ is claw-free. Let f_1, f_2, f_3, and f_4 be four facts of I such that (f_1, f_2), (f_1, f_3), (f_1, f_4) are edges in E. Then either at least two of these three edges are in E_1 or at least two of these three edges are in E_2. If, say, both (f_1, f_2) and (f_1, f_3) are in E_1, then, by the first observation above, we have that (f_2, f_3) is an edge in E_1 (and hence in E). If, say, both (f_1, f_2) and (f_1, f_3) are in E_2, then, by the second observation above, we have that (f_2, f_3) is an edge in E_1 (and hence in E). Therefore, the nodes f_1, f_2, f_3, and f_4 do not induce a claw in $H_{I,q}$.

Finally, assuming that $(\mathit{key}(R_1) \cup \mathit{key}(R_2)) \subseteq L$, there is a polynomial-time algorithm for $\textsc{certainty}(q)$. Specifically, given an instance I, we first construct the conflict-join graph $H_{I,q}$ in polynomial time in the size of I. Since $H_{I,q}$ is claw-free, we can use Minty's algorithm [52] to compute the independent set number $\alpha(H_{I,q})$ in polynomial time in the size of $H_{I,q}$ and, hence, in polynomial time in the size of I. We then compare $\alpha(H_{I,q})$ to the number n of distinct maximal sets of pairwise key-equal facts of I, which can also be computed in polynomial time in the size of I. By Lemma 3, we have that CERTAINTY(q) is true on I if and only if $\alpha(H_{I,q}) < n$. $\square$

It should be noted that Arenas et al. [3] introduced the notion of the *conflict* graph while studying the consistent answers of aggregate queries. The conflict graph is constructed from the constraints and the instance, while our conflict-join graph takes also the query into account. Arenas et al. used the tractability of the maximum independent set number on claw-free graphs to show that if a relational schema with at most two functional dependencies is in Boyce-Codd Normal Form, then there is a polynomial-time algorithm for computing the consistent answers of COUNT(*) queries (see [3, Theorem 12]). The preceding Lemma 4 could also be obtained via a reduction to the problem of computing the consistent answers of COUNT(*) queries and then by appealing to Theorem 12 in [3]. The proof we gave here is direct and self-contained.

Lemma 4 gives a broad sufficient condition for the tractability of CERTAINTY(q) for self-join free boolean conjunctive queries q with exactly two atoms. In particular, it yields a unifying polynomial-time algorithm for CERTAINTY(q) that applies to several interesting queries q for which CERTAINTY(q) is not first-order expressible. To begin with, it implies that CERTAINTY(q_2) is in P, where q_2 is the query $q_2() : -R_1(\underline{x}, y), R_2(\underline{y}, x)$ from the Introduction. Note that the sole focus of [60] was showing that CERTAINTY(q_2)

is in P (using a different algorithm than ours) but is not first-order expressible. Also, Lemma 4 implies that CERTAINTY(q) is in P, where q is one of the following three queries:

- $q() : -R_1(\underline{x, z}, y), R_2(\underline{y}, x, z)$
- $q() : -R_1(\underline{x}, y, z), R_2(\underline{y}, x, z)$
- $q() : -R_1(\underline{x, y}, z), R_2(\underline{x, z}, y)$

The proof of Theorem 5 can now be obtained by combining Lemma 2 with Lemma 4. Thus, if q is a self-join free boolean conjunctive query with two atoms such that CERTAINTY(q) is not first-order expressible, then either CERTAINTY(q) is in P or CERTAINTY(q) is coNP-complete. Moreover, CERTAINTY(q) is in P if and only if $key(R_1) \cup key(R_2) \subseteq L$.

Note that, by Lemma 4, the condition $key(R_1) \cup key(R_2) \subseteq L$ is a sufficient condition for tractability of CERTAINTY(q), even if CERTAINTY(q) is first-order expressible. However, in general, this is not a necessary condition for tractability of CERTAINTY(q). For example, consider again the query $q_1() : -R_1(\underline{x}, y), R_2(\underline{y}, z)$ from the Introduction. Then $key(R_1) \cup key(R_2) = \{x, y\} \not\subseteq L = \{y\}$. However, as seen earlier, CERTAINTY(q_1) is first-order expressible. Similarly, if q is the query $q() : -R_1(\underline{x, y}, z), R_2(\underline{y, u}, w)$, then $key(R_1) \cup key(R_2) = \{x, y, u\} \not\subseteq L = \{y\}$, yet CERTAINTY($q$) is in P, because $L \subseteq R_1^+ = \{x, y\}$, hence CERTAINTY($q$) is first-order expressible.

4.3 Sufficient Condition for Intractability

For the general case when the query contains an arbitrary number of atoms, we have found a sufficient condition for intractability. We will use the following terminology and notation to present our result. If q is a conjunctive query, $R(\mathbf{x})$ is an atom in q, f is a fact of the form $R^I(t)$ and x is a variable in $\mathbf{x}$, we use the notation $t[x] = a$ to denote the fact that the constant a appears in every position of variable x in t. If $(x_1, \cdots, x_k)$ is a tuple of variables such that $x_i \in \mathbf{x}$ for $1 \leq i \leq k$, we write $t[x_1, \cdots, x_k] = (a_1, \cdots, a_k)$ to express that $t[x_i] = a_i$ for $1 \leq i \leq k$. Given two facts $R^I(t)$ and $S^I(t)$, where R and S need not be different, and x a variable that appears in atoms R and S in q, we write $t[x] = s[x]$ to express that there exists a constant a such that $t[x] = a$ and $s[x] = a$. If a and b are constants, then $a \cdot b$ is a new constant formed by concatenating a and b. Finally, we will make use of the notion of the *attribute closure* of an atom with respect to the primary key constraints, defined next.

Definition 8 *Let q be an acyclic and self-join free boolean conjunctive query. Let Σ be a set of primary key constraints over the same schema as q. For every atom R_i in q, we define $vars(R_i)^+$ to be the set $vars(R_i)^+ = \{v \in vars(q) \mid \Sigma \models (key(R_i) \rightarrow v)\}$.*

For example, in the query $q() :- R_1(\underline{x}, y), R_2(\underline{x'}, y), R_3(\underline{y}, x)$, we have that $vars(R_1)^+ = \{x, y\}$, $vars(R_2)^+ = \{x', y, x\}$, and $vars(R_3)^+ = \{y, x\}$.

We are now ready to present our sufficient condition for intractability of CERTAINTY(q), in Theorem 6:

Theorem 6 *Let q be a self-join free boolean acyclic conjunctive query with atoms $R_1, \cdots, R_n$, and such that* CERTAINTY(q) *is not first-order expressible. If there are two atoms R_i, R_j, $1 \leq i, j \leq n$, $i \neq j$, such that:*

- *There is a cycle of length two between R_i, R_j in the attack graph, and*
- *$key(R_i) \nsubseteq (vars(R_j))^+$ or $key(R_j) \nsubseteq (vars(R_i))^+$,*

Then CERTAINTY(q) *is coNP-complete.*

Proof. We will prove this theorem by showing that there is a polynomial reduction from CERTAINTY(q') to CERTAINTY(q), where q' is the query $q'() : -S_1(\underline{x, y}, x'), S_2(\underline{x'}, y)$. W.l.o.g., we will assume that R_i is such that $key(R_i) \nsubseteq vars(R_j)^+$. Given an instance I' over the schema of q', we will construct an instance I over the schema of q. To describe the transformation of I' to I, we will make use of a function f that maps the variables of q to variables of q'. Let Q be the set of variables of q, let P be the set of variables of q', i.e., $P = \{x, x', y\}$, and let c be a fixed constant that is not in the domain of any of the variables of Q and P. We define f to be a function of the form $f : Q \rightarrow 2^{P \cup \{c\}}$. So, f maps every variable appearing in q to a subset of $P \cup \{c\}$. We will first show how to define this function, and then how to construct I from I'. Function f is defined recursively:

```
1. for every $v \in vars(q)$, let $f(v) = \emptyset$

2. repeat until $f$ does not change
  3. for every $R_k$ atom in $q$ do
    4. for every $v \in vars(R_k) \setminus (vars(R_j))^+$ set $f(v) = f(v) \cup \{x\}$
    5. for every $v \in vars(R_k) \setminus R_i^+$ set $f(v) = f(v) \cup \{x'\}$
    6. for every $v \in vars(R_k) \setminus R_j^+$ set $f(v) = f(v) \cup \{y\}$

7. if $f(v)$ is not defined for some $v$, then let $f(v) = \{c\}$
```

Next, we show how to construct the database instance I from I' using f:

- For every fact $g_1 = S_1(a, b, a')$ in I', for every $R_p \neq R_j$, generate a fact $f_p = R_p(t_p)$ in I such that for every $v \in vars(R_p)$ we have that:

  – $t_p[v] = a$ if $f(v) = \{x\}$

  – $t_p[v] = b$ if $f(v) = \{y\}$

  – $t_p[v] = a'$ if $f(v) = \{x'\}$

  – $t_p[v] = a \cdot b$ if $f(v) = \{x, y\}$

  – $t_p[v] = b \cdot a'$ if $f(v) = \{y, x'\}$

  – $t_p[v] = a \cdot a'$ if $f(v) = \{x, x'\}$

  – $t_p[v] = a \cdot b \cdot a'$ if $f(v) = \{x, y, x'\}$

  – $t_p[v] = c$ if $f(v) = \{c\}$

  We use the notation $g_1 \rightrightarrows \{f_1, \cdots, f_{j-1}, f_{j+1}, \cdots f_n\}$ to express that g_1 generates facts $\{f_1, \cdots, f_{j-1}, f_{j+1}, \cdots f_n\}$.

- For every fact $g_2 = S_2(a', b)$ in I', generate a fact $f_j = R_j(t_j)$ in I such that for every $v \in vars(R_j)$ we have that:

– $t_j[v] = a'$ if $f(v) = \{x'\}$

– $t_j[v] = b$ if $f(v) = \{y\}$

– $t_j[v] = b \cdot a'$ if $f(v) = \{y, x'\}$

– $t_j[v] = c$ if $f(v) = \{c\}$

We say that $g_2 \rightrightarrows f_j$.

Next, we prove the following properties of the function f:

Property 1: $\{x, y\} \subseteq f(key(R_i)) \subseteq \{x, y, c\}$ and $\{x'\} \subseteq f(nkey(R_i)) \subseteq \{x, x', y\}$

Property 2: $\{x'\} \subseteq f(key(R_j)) \subseteq \{x', c\}$ and $\{x', y\} \subseteq f(nkey(R_j)) \subseteq \{x', y, c\}$

Proof of Property 1 and 2 Let τ be a join tree of q. In τ there must be a path $R_i, R_{i+1}, ...R_{j-1}, R_j$ between R_i and R_j. Because there is a cycle of length two in the attack graph, it must happen that for $1 \leq p < j-1$ we have that $L_{p,p+1} \not\subseteq R_i^+$ and $L_{p,p+1} \not\subseteq R_j^+$. Notice that $key(R_i) \not\subseteq R_j^+$, because otherwise, from $key(R_i) \subseteq R_j^+$ would follow $vars(R_i) \subseteq R_j^+$, and in particular $L_{i,i+1} \subseteq R_j^+$. Similarly, we can reason that $key(R_j) \not\subseteq R_i^+$. Since $key(R_i) \not\subseteq vars(R_j)^+$ then, there is a variable $u \in key(R_i)$ such that $u \notin vars(R_j)^+$. Obviously, it cannot happen that $u \in R_j^+$. Hence, $\{x, y\} \subseteq f(key(R_i))$. Moreover, from $key(R_i) \subseteq R_i^+$ it follows that $x' \notin f(key(R_i))$. So far we have that $\{x, y\} \subseteq f(key(R_i)) \subseteq \{x, y, c\}$. We also have that $nkey(R_i) \not\subseteq R_i^+$. Assume $nkey(R_i) \subseteq R_i^+$. Then, $vars(R_i) \subseteq R_i^+$, and therefore, $L_{i,i+1} \subseteq R_i^+$. This contradicts the assumption. Hence, $nkey(R_i) \not\subseteq R_i^+$. Therefore, $x' \in f(nkey(R_i))$. Both x and y are allowed to appear in the $f(nkey(R_i))$. So, we have that $\{x'\} \subseteq f(nkey(R_i)) \subseteq \{x, x', y\}$. Because $key(R_j) \not\subseteq R_i^+$, it follows that there is a variable in the key of R_j that is mapped by f to a set that contains x'. So, $x' \in f(key(R_j))$. Because $key(R_j) \subseteq R_j^+$ and also $key(R_j) \subseteq vars(R_j)^+$, we have that $y \notin f(key(R_j))$ and $x \notin f(key(R_j))^+$. So far, we have established that $\{x'\} \subseteq f(key(R_j)) \subseteq \{x', c\}$. Since $nkey(R_j) \subseteq (vars(R_j))^+$, we have that $x \notin f(nkey(R_j))$. Assume $nkey(R_j) \subseteq R_j^+$. Then, $vars(R_j) \subseteq R_j^+$, and therefore, $L_{j-1,j} \subseteq R_j^+$. This contradicts our assumption that every label in the path between R_i and R_j is not included in R_j^+. From $nkey(R_j) \not\subseteq R_j^+$ it follows that $y \in f(nkey(R_j))$. From $nkey(R_j) \subseteq (vars(R_j))^+$ it follows that $x \notin f(nkey(R_j))$. Hence, $\{x', y\} \subseteq f(nkey(R_j)) \subseteq \{x', y, c\}$.

Next, we prove an important property of the construction of I from I':

Property 3: Every R_p^I for $p \neq i, j$, is a consistent relation.

Proof of Property 3 For a given set of variables S, we define $f(S)$ to be the set $f(S) = \bigcup_{v \in S} f(v)$. Let R_p be any atom different from R_i, R_j.

- Case I) $key(R_p) \not\subseteq R_i^+$ and $key(R_p) \not\subseteq R_j^+$

  There is a variable $v \in key(R_p)$ such that $v \notin R_i^+$ and there is a variable $w \in key(R_p)$ such that $w \notin R_j^+$. Then, $x' \in f(v)$ and $y \in f(w)$. So, $\{x', y\} \subseteq f(key(R_k))$. If $nkey(R_p) \subseteq (vars(R_j))^+$, then $x \notin f(nkey(R_p))$. Hence, $f(nkey(R_p)) \subseteq f(key(R_p)) \cup \{c\}$. If there is a variable $v \in nkey(R_p)$ such that $v \in (vars(R_j))^+$ then, there must be a variable $w \in key(R_p)$ such that $w \in (vars(R_j))^+$. Otherwise, if $key(R_p) \subseteq (vars(R_j))^+$ then also w would be in $vars(R_j)^+$. Therefore, $\{x, x', y\} \subseteq f(key(R_p))$, and $f(nkey(R_p)) \subseteq f(key(R_p)) \cup \{c\}$. From $f(nkey(R_p)) \subseteq f(key(R_p)) \cup \{c\}$ it follows that in R_p^I there cannot exist two key-equal facts.

- Case II) $key(R_p) \subseteq R_i^+$ and $key(R_p) \nsubseteq R_j^+$

  Since there is a variable $v \in key(R_p)$ such that $v \notin R_j^+$ then, $y \in f(v)$. So, $y \in f(key(R_p))$. Since $key(R_p) \subseteq R_i^+$, then $vars(R_p) \subseteq R_i^+$. It follows that $x' \notin f(key(R_p))$ and $x' \notin f(nkey(R_p))$. If $nkey(R_k) \subseteq (vars(R_j))^+$, then $x \notin f(nkey(R_p))$. Hence, $f(nkey(R_p)) \subseteq f(key(R_p)) \cup \{c\}$. If there is a variable $v \in nkey(R_p)$ such that $v \in (vars(R_j))^+$ then, there must be a variable $w \in key(R_p)$ such that $w \in (vars(R_j))^+$. Otherwise, if $key(R_p) \subseteq (vars(R_j))^+$ then also w would be in $vars(R_j)^+$. Again we have that $f(nkey(R_p)) \subseteq f(key(R_p)) \cup \{c\}$. In R_p^I there cannot exist two key-equal facts.

- Case III) $key(R_p) \nsubseteq R_i^+$ and $key(R_p) \subseteq R_j^+$

  Since there is a variable $v \in key(R_p)$ such that $v \notin R_i^+$ then, $x' \in f(v)$. So, $x' \in f(key(R_k))$. Since $key(R_p) \subseteq R_j^+$, then $vars(R_p) \subseteq R_j^+$. For every variable v in R_j^+ we have that $f(v) \subseteq \{x', c\}$. So, $\{x'\} \subseteq f(key(R_p))$ and $f(nkey(R_p)) \subseteq \{x'.c\}$. In any case, $f(nkey(R_p)) \subseteq f(key(R_p)) \cup \{c\}$, and therefore, R_p^I is consistent.

- Case IV) $key(R_p) \subseteq R_i^+$ and $key(R_p) \subseteq R_j^+$

  Here we have that $key(R_p) \subseteq R_i^+ \cap R_j^+$. Then, also $nkey(R_p) \subseteq R_i^+ \cap R_j^+$. A variable $v \in R_i^+$ cannot contain x'. A variable $v \in R_j^+$ cannot contain y. Hence, a variable $v \in R_i^+ \cap R_j^+$ cannot contain x' or y. Since every variable $v \in vars(R_k)$ is in R_j^+ then, we also have that v is in $(vars(R_j))^+$. Hence, $x \notin f(v)$. It remains that for every $v \in vars(R_p)$ we have that $f(v) = \{c\}$. Again, we have that $f(nkey(R_p)) \subseteq f(key(R_p)) \cup \{c\}$ and R_p^I is a consistent relation.

($\Rightarrow$) Next we prove that from each repair of I' on which q' is false, we can construct a repair of I on which q is false.

Let r' be a repair of I' such that $r' \not\models q'$. We construct r doing the following:

- for every fact $g_1 \in S_1^{r'}$ add to R_i^r the fact f_i such that $g_1 \rightrightarrows f_i$.
- for every fact $g_2 \in S_2^{r'}$ add to R_j^r the fact f_j such that $g_2 \rightrightarrows f_j$.
- for every p such that $1 \leq p \leq n$ and $p \neq i, j$, add to R_p^r every fact from R_p^I.

We show that r is a repair of I and that $r \not\models q$.

It is easy to argue that r is a repair. From Property 1 it follows that for every key-equal pair of facts in S_1 generates two key-equal facts in R_i. Similarly, it follows from Property 2 that every pair of key-equal facts in S_2 generates two key-equal facts in R_j. From Property 3 we have that every other relation is consistent.

Next, we show that r does not satisfy q. Assume there is a minimal witness $\{f_1, ..., f_i, ...f_j, ..., f_n\}$ of q in r. Assume also that the path between R_i and R_j in τ is $R_i, R_{i+1}, \cdots, R_{j-1}, R_j$. We know that for every $L_{p,p+1}$ in the path between R_i and R_j, where $i \leq p \leq j-1$, we have that $L_{p,p+1} \not\subseteq R_i^+$ and $L_{p,p+1} \not\subseteq R_j^+$. Hence, in every $L_{p,p+1}$ there are variables m, l such that $m \notin R_i^+$ and $l \notin R_j^+$. It follows that $\{x', y\} \subseteq f(L_{p,p+1})$. Then, the facts $f_i = R_i(t_i)$ and $f_j = R_j(t_j)$ are such that for every $v \in L_{i,j}$ for which $f(v) \subseteq \{x', y\}$, it holds that $t_i[v] = t_j[v]$. Let $g_1 = S_1(w_1)$ and $g_2 = S_2(w_2)$ be the facts in r' that generate, respectively, f_1 and f_2. Because S_1 and S_2 share only the variables x' and y, it is obvious now that the facts g_1 and g_2 form a minimal witness of the query q' on r'.

($\Leftarrow$) In the other direction, let r be a repair of I such that $r \not\models q$. We construct r' doing the following:

- For every fact $f_i \in R_i^r$ add to $S_1^{r'}$ the fact g_1 such that $g_1 \rightrightarrows \{\cdots, f_i, \cdots\}$
- For every fact $f_j \in R_j^r$ add to $S_2^{r'}$ the fact g_2 such that $g_2 \rightrightarrows \{\cdots, f_j, \cdots\}$

Again, it is easy to argue that r' is a repair using Properties 1-3.

Next, we show that r' does not satisfy q'. Assume there are facts $g_1 = S_1(w_1)$ and $g_2 = S_2(w_2)$ in r' that form a minimal witness for q'. Then, it must hold that $w_1[x'] = w_2[x']$ and $w_1[y] = w_2[y]$. Let $f_i = R_i(t_i)$ and $f_j = R_j(t_j)$ be the facts of r such that $g_1 \rightrightarrows \{f_1 \cdots, f_i, \cdots, f_{j-1}, f_{j+1}, \cdots, f_n\}$, $g_2 \rightrightarrows f_j$. Notice that the set $\{f_1 \cdots, f_i, \cdots f_{j-1}, f_{j+1}, \cdots, f_n\}$ is a minimal witness of the query $q \backslash \{R_j\}$. Since for all $p \neq i, j$ we have that $R_p^r = R_p^I$, all of the facts $\{f_1 \cdots, f_i, \cdots, f_{j-1}, f_j, \cdots, f_n\}$ appear in r. For every relation R_p that joins with R_j, we have that $L_{p,j} \subseteq (vars(R_j))^+$. So, it cannot happen that $x \notin f(L_{p,j})$. Therefore, $f(L_{p,j}) \subseteq \{x', y, c\}$. From Property 2 we know that $x \notin f(vars(R_j))$. It follows that for every relation R_p that joins with R_j, the fact f_p will join with the fact f_j. Hence, the set of facts $f_1, ..., f_n$ forms a minimal witness for q. This contradicts the assumption that $r \not\models q$. $\square$

A few examples of queries that illustrate the application of Theorem 6 are:

- $q() : -R_1(\underline{x}, y), R_2(\underline{x'}, y), R_3(\underline{x''}, y)$
- $q() : -R_1(\underline{x, z}, y), R_2(\underline{y}, x), R_3(\underline{x}, y)$

4.4 Sufficient Condition for Tractability

In Section 4.2.2 we showed how to reduce CERTAINTY(q), where q is a self-join free and boolean conjunctive query with two atoms, to the problem of finding a maximum independent set in a claw-free graph. We used simple edges in the conflict-join graph to represent minimal witnesses of q, since every minimal witness involves exactly two facts from the database. If we consider a query with an arbitrary number of atoms, then each minimal witness of the query involves as many facts as there are atoms in q. In this case, the edges in the conflict-join graph that represent minimal witnesses, become hyper-edges, and the conflict-join graph becomes a hypergraph. The problem of finding a maximum independent set of vertices in a hypergraph is even more complicated compared to simple graphs. Little is known about tractability of this problem in hypergraphs. However, for conjunctive queries that are acyclic and self-join free, we show that one can still reduce CERTAINTY(q) to the problem of finding a maximum independent set in a simple graph. In this section, we show how to generalize the conflict-join graph construction to acyclic self-join free boolean conjunctive queries with an arbitrary number of atoms. Then, we show that it suffices to solve the maximum independent set problem on this graph in order to check CERTAINTY(q). Finally, we provide a syntactic condition on the query, such that for every query that satisfies this condition, the conflict-join graph is claw-free; hence, CERTAINTY(q) is in P.

In order to define the generalized version of the conflict-join graph, we will need the following notation. Given a graph G and a vertex v in it, the set $N_G(v)$ is the neighborhood of v in G, i.e., the set of all vertices that are adjacent with v in G. Given a self-join free conjunctive query q, and R_i, R_j two atoms in q, the query $q_{\{R_i,R_j\}}$ is the query formed by removing from q all atoms but R_i and R_j.

Definition 9 *Let q be an acyclic Boolean conjunctive query without self-join, and with atoms $R_1, R_2, \cdots, R_k$. Let τ be a join tree for q. Let I be an instance over the same schema as q. The* conflict-join graph*, denoted $H_{I,q,\tau}$, is the graph constructed as follows:*

- *for every key-equal group K in R_i, where $1 \leq i \leq k$, add a clique with vertices $\{f^{R_j} : f \in K \text{ and } R_j \in N_\tau(R_i)\}$*
- *for every two atoms R_i, R_j connected with an edge in τ, for every R_i-fact f and R_j-fact g such that $\{f, g\} \models q_{\{R_i,R_j\}}$, add an edge between f^{R_j} and g^{R_i}*

Example 7 Let $q() : -R_1(\underline{x}, y), R_2(\underline{y}, x), R_3(\underline{x, y}, z), R_4(\underline{z}, x, y)$. In Figure 4.4, on the left we show an inconsistent database instance over the same schema as q, in the middle we show a join tree τ for the query, and on the right we show the conflict-join graph $H_{I,q,\tau}$. Notice that the fact $R_1(a, b)$ is represented by a single vertex $R_1(a, b)^{R_3}$, where the superscript R_3 is the name of R_1's only neighbor. On the other hand, the fact $R_3(a, b, c)$ is represented by three different vertices, one for every neighbor of R_3. Observe also that the facts $R_1(a, b)$ and $R_3(a, b, c)$ together satisfy q_{R_1,R_3}. Therefore, we add an edge between $R_1(a, b)^{R_3}$ and $R_3(a, b, c)^{R_1}$. $\square$

For a fixed query and constraints, the size of $H_{I,q,\tau}$ is polynomial in the size of the database. Assume that q has k atoms, and assume that each relation in the

R_1

A	**B**
a	b

R_2

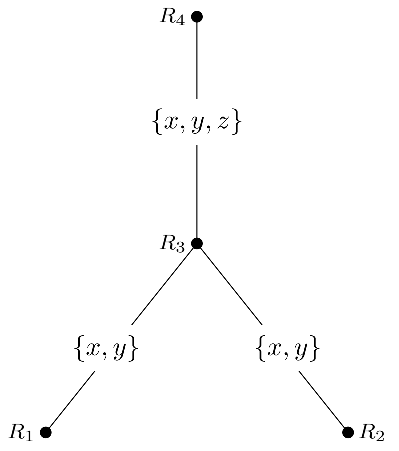
B	**A**
b	a

R_3

A	**B**	**C**
a	b	c
a	b	d

R_4

C	**A**	**B**
c	a	b
c	a	d



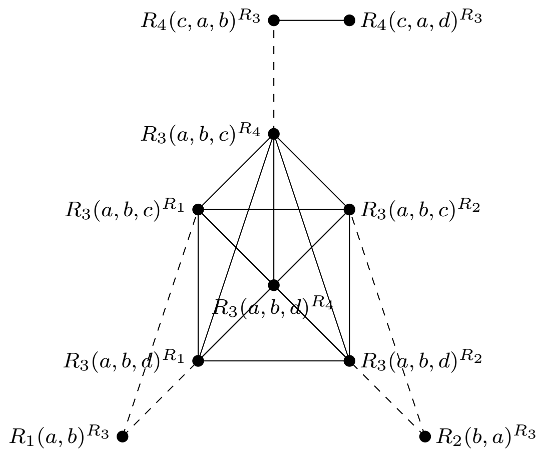


Figure 4.4: Top: Database instance I. Bottom left: A join tree τ for query $q() :- R_1(\underline{x}, y), R_2(\underline{y}, x), R_3(\underline{x, y}, z), R_4(\underline{z}, x, y)$. Bottom right: Conflict-join graph $H_{I,q,\tau}$

database I contains at most n tuples. Observe that for every two atoms R_i and R_j that are neighbors in τ, for every R_i-fact f there is a vertex f^{R_j}, and for every R_j-fact g there is a vertex g^{R_i}. Therefore, for each edge in τ we generate at most $2 \times n$ vertices. There are $k - 1$ edges in any join-tree τ. So, $H_{I,q,\tau}$ has at most $(k - 1) \times 2n$ vertices.

The relationship between the existence of a repair in which q is false and the existence of a maximum independent set of a particular size in $H_{I,q,\tau}$ still holds. For instance, in Figure 4.4 there are four key-equal groups in the database. Observe that the set of vertices $S = \{R_1(a,b)^{R_3},\ R_2(b,a)^{R_3},\ R_3(a,b,c)^{R_4}, R_4(c,a,d)^{R_3}\}$ is a maximum independent set with four vertices. This independent set can be mapped into a database instance $r = \{R_1(a,b), R_2(b,a), R_3(a,b,c), R_4(c,a,d)\}$. Indeed, r is a repair that does not satisfy q. On the other hand, if we consider the set $T = \{R_3(a,b,d)^{R_1}, R_2(b,a)^{R_3}, R_4(c,a,b)^{R_3}\}$, which is also an independent set, but of size three, then the database instance $r' = \{R_3(a,b,d), R_2(b,a), R_4(c,a,b)\}$, while it does not satisfy q, it is not a repair because it is possible to add $R_1(a,b)$ without violating any primary key constraints. In Lemma 5 we formally prove the relationship between $\alpha(H_{I,q,\tau})$ and the number of key-equal groups n.

In what follows, we will make use of some additional notation and terminology. Every key-equal group in I, generates a clique in $H_{I,q,\tau}$. So, the set of vertices of the conflict-join graph can be partitioned into disjoint sets $C_1, C_2, \cdots, C_n$, where each set C_i, for $1 \leq i \leq n$, induces a clique. We will refer to edges that appear in these cliques as *conflict edges*. The rest of the edges are generated from pairs of facts that satisfy some query q_{R_i,R_j} for R_i and R_j that are neighbors in τ. We will refer to these edges by the name *join edges*.

Lemma 5 *Let q be an acyclic Boolean conjunctive query without self-join, and with atoms $R_1, R_2, \cdots, R_k$. Let τ be a join tree for q. Given an instance I over the same schema as q, the following are equivalent:*

- *There exists a repair r of I such that $r \not\models q$*
- *$\alpha(H_{I,q,\tau}) = n$, where n is the number of key-equal groups in I*

PROOF. For the purpose of this proof, we will fix an atom to be the root of τ. After fixing a root, we can infer child-parent relationships between atoms in τ. Assuming we have fixed a root in τ, we will denote by q_{R_i} the conjunctive query with atoms the descendants of R_i in τ. To simplify the presentation of the proof, we will use the following terminology to refer to vertices from the set $\{f^{R_j} : R_j \in N_\tau(R_i)\}$, where f is an R_i-fact: the vertex f^{R_p} where R_p is the parent of R_i will be called the *parent vertex of* f; any of the other vertices will be called a *child vertex* of f.

($\Rightarrow$) In one direction, assume there is a repair r such that $r \not\models q$. We will show how to construct an independent set M of size n. To construct M we use the algorithm in Figure 4.5.

```
Input: q, I, $\tau$, $H_{I,q,\tau}$, a repair $r$ of $I$ such that $r \not\models q$
1. let $M = \emptyset$
2. let $R_1$ be the root of $\tau$
3. for every $R_i$ in $q$
4.    if $R_i \neq R_1$ then let $R_p$ be the parent of $R_i$ in $\tau$
5.    let $children(R_i)$ be the set of children of $R_i$ in $\tau$
6.    for every $R_i$-fact $f$ in $r$
7.       if there exists a minimal witness $S$ of $q_{R_i}$ in $r$ s.t. $f \in S$
8.          if $R_i \neq R_1$ then add $f^{R_p}$ to $M$
9.       else
10.         let $R_j \in children(R_i)$ be s.t. for every minimal witness
                $S$ of $q_{R_j}$, $S \cup \{f\} \not\models q_{R_j} \cup R_i$
11.         add $f^{R_j}$ to $M$
12.return $M$
```

Figure 4.5: Algorithm for constructing a maximal independent set from a repair that falsifies the query.

Next, we argue that the algorithm in Figure 4.5, indeed constructs a set of vertices M that is independent and has size n. First, we argue that M is independent.

Assume instead that M induces some edge. If M induces a conflict edge, then the two endpoints of this edge are of the form f^{R_i} and g^{R_j}, where f and g are key-equal. Because for every fact f from r the algorithm picks a single vertex of the form f^{R_i}, where $i \in \{1, \cdots, k\}$, then it is not possible that $f = g$. Moreover, f and g cannot be distinct key-equal facts because r is consistent. If M induces a join edge, then the endpoints of this edge must be of the form f^{R_i} and g^{R_j}, where R_i and R_j are neighbors in τ, f is an R_j-fact, g is an R_i-fact and $\{f, g\} \models q_{R_i,R_j}$. Assume w.l.o.g. that R_i is the parent of R_j. Then, the vertex f^{R_i} must have been added to M in line 8, and therefore, there must exist in r a minimal witness S of q_{R_j} such that $f \in S$. But we also know that $\{f, g\} \models q_{R_i,R_j}$. It follows that $S \cup \{g\} \models q_{R_j} \cup R_i$. Then R_j does not satisfy the condition required by the algorithm in line 10. Therefore, g^{R_j} could not have been added to M. So far, we have argued that M is independent. Next, we argue that M has size n. It is easy to see that for every non-root relation R_i, the algorithm adds a distinct vertex for each R_i-fact of r. On the other hand, it is not obvious that this is the case for the root relation as well. Assume towards a contradiction that for some R_1-fact f in r, the algorithm failed to add in M a vertex of the form f^{R_j}, where $R_j \in \mathit{children}(R_1)$. This means that no child of R_1 satisfied the condition in line 10. Then, for every $R_j \in \mathit{children}(R_1)$, there must exist a minimal witness S_j of q_{R_j} such that $S \cup \{f\} \models (q_{R_j} \cup R_1)$. But the set $S = \bigcup_{j:R_j \in \mathit{children}(R_1)} S_j$ has the property that $S \cup \{f\} \models q$, which contradicts the assumption that $r \not\models q$.

($\Leftarrow$) In the opposite direction, we will show how, given an independent set M of size n, we can construct a repair r that falsifies q. We construct r by adding a fact f to r if and only if there exists a vertex f^{R_j} in M, where $j \in \{1, \cdots, k\}$. Every key-equal group generates a clique in $H_{I,q,\tau}$. Thus, there are n such cliques in $H_{I,q,\tau}$, and exactly one vertex per clique must be in M. Therefore, r contains exactly one fact from every key-equal group. We have so far argued that r is a repair. Next, we argue that $r \not\models q$. Assume instead that there exists a minimal witness S of q in r. In our argument towards a contradiction, we will make use of the following property of the repair constructed from M:

Property * Let R_i be an atom such that $R_i \neq R_1$ and $r \models q_{R_i}$. Let S be a minimal subset of r with the property that it satisfies q_{R_i}. Then for every $f \in S$, the parent vertex of f is in M.

*Proof of Property ** We will prove this property by structural induction on τ.

Base step: If R_i is a leaf node, then every R_i-fact f is represented by a single vertex, which is the parent vertex.

Inductive step: Let R_i be a non-leaf node in τ, and assume S is a minimal witness of q_{R_i} in r. Obviously, for every $R_j \in \mathit{children}(R_i)$, it also holds that $S \models R_j$. For every $R_j \in \mathit{children}(R_i)$, let S_j be the minimal subset of S that satisfies q_{R_j}. By induction hypothesis, for every $R_j \in \mathit{children}(R_i)$, for every fact $f \in S_j$, the parent vertex of f is in M. Let g be the R_i-fact in S. If R_j is a child of R_i, and f is the R_j-fact in S, then in $H_{I,q,\tau}$ there must be an edge between g^{R_j} and f^{R_i}. Notice that f^{R_i}, being the parent vertex of f, it is in M. It follows that for every child R_j, if the vertex g^{R_j} would be in M it would induce an edge. Therefore, the parent vertex of g has to be in the set M.

Going back to the proof of the lemma, we will assume that S is a minimal subset of r that satisfies q. Then, for every $R_j \in children(R_1)$, it also holds that $S \models q_{R_j}$. Let f be the R_1-fact in S. From Property * it follows that for every $g \in S$, where $g \neq f$, the parent vertex of g must be in M. Then, none of the facts f^{R_j} can be in M because it would induce an edge. This conclusion contradicts the fact that r was taken to contain all and only those facts f such that there is some vertex of the form f^{R_j} in M. $\square$

Lemma 5 establishes that the largest independent set in $H_{I,q,\tau}$ has size equal to n if and only if q is false in some repair of I. Moreover, the repairs of I that falsify q are precisely the independent sets of $H_{I,q,\tau}$ of size n. We observe that for a subclass of acyclic and self-join free conjunctive queries with primary key constraints, $H_{I,q,\tau}$ is always claw-free. Next, in Theorem 7 we state and prove our result concerning the tractability of a sub-class of acyclic and self-join free conjunctive queries.

Theorem 7 *Let q be an acyclic boolean conjunctive query without self-join, and with atoms $R_1, R_2, \cdots, R_k$. If q has a join tree τ such that for every pair of atoms R_i, R_j connected with an edge in τ, it holds that $key(R_i) \subseteq vars(R_j)$ and $key(R_j) \subseteq vars(R_i)$, then for every database I, the conflict-join graph $H_{I,q,\tau}$ is claw-free. Consequently,* CERTAINTY(q) *is in P.*

PROOF. Let e_1, e_2, e_3 be three edges in $H_{I,q,\tau}$ that are adjacent in the same vertex. Two cases are possible for these edges:

Case 1: At least two are conflict edges.

Assume e_1 and e_2 are conflict edges. Each conflict edge appears in exactly one clique induced by a set C_i, for some $i \in \{1, \cdots n\}$. Since e_1 and e_2 are adjacent, then they must appear in the same clique. Therefore, in $H_{I,q,\tau}$ there is an edge from every endpoint of e_1 to every endpoint of e_2

Case 2: At least two are join edges

Assume e_1 and e_2 are join edges. Let e_1 be an edge between f^{R_i} and g^{R_j}, where f is an R_j-fact and g is an R_i-fact. Notice that f^{R_i} can connect via a join edge only to vertices with the superscript R_j. Let e_2 be an edge between f^{R_i} and h^{R_j}, where h is an R_i-fact, and $h \neq g$. Because the $key(R_i) \subseteq vars(R_j)$, then the variables in $key(R_i)$ are shared between the atoms R_i and R_j. Since $\{f, g\} \models q_{R_i,R_j}$ and $\{f, h\} \models q_{R_i,R_j}$ then the facts g and h must have the same key value. $\square$

A few examples of queries that illustrate the application of Theorem 7 are:

- $q() : -R_1(\underline{x, y}, z), R_2(\underline{x, z}, y), R_3(\underline{y, z}, x)$
- $q() : -R_1(\underline{x}, y), R_2(\underline{y}, x, z), R_3(\underline{z}, y)$

There is a simple generalization of our tractability result to a slightly broader class of queries, using Proposition 2. Using Proposition 2 we get the following corollary of Theorem 7.

Corollary 4 *Let q be a self-join free, acyclic boolean conjunctive query. Let q' be the query formed from q by removing all variables that are irrelevant. If q' has a join tree τ*

with the property that for every pair R_i, R_j of neighbors in τ, $(key(R_i) \cup key(R_j)) \subseteq L_{i,j}$, then $H_{I,q,\tau}$ is claw-free. Consequently, CERTAINTY*(q') is in P.*

Given a query q, it can happen that some join tree satisfies the condition of Theorem 7, and some other join tree does not. For example, let $q() : -R_1(\underline{x}, y)$, $R_2(\underline{y}, x, x'), R_3(\underline{x'}, x, y)$. One join tree for this query is the path $\langle R_1, R_3, R_2\rangle$. In the given join tree, the atoms R_1, R_3 are such that $key(R_3) \not\subseteq vars(R_1)$; hence, Theorem 7 does not apply. Another join tree for q is the path $\langle R_1, R_2, R_3\rangle$. For this join tree, we have that $key(R_1) \in vars(R_2)$, $key(R_2) \in vars(R_1)$ and $key(R_2) \in vars(R_3)$ and $key(R_3) \in vars(R_2)$. So, the condition of Theorem 7 is satisfied.

Queries are known to exist for which CERTAINTY(q) is in P and not first-order expressible, but the condition of Theorem 7 is not satisfied. One such query is $q() : -R_1(\underline{x}, y), R_2(\underline{z}, x, y), R_3(\underline{y}, z)$. For this query, we show in Appendix C that CERTAINTY(q) is in P, via a rather special algorithm.

4.5 General Dichotomy: Conjecture

4.5.1 A Conjecture on the Dichotomy of Consistent Query Answering for Acyclic and Self-Join Free Conjunctive Queries.

Based on the intuition we have acquired from the results presented in Section 4.3 and Section 4.4, we conjecture that there exists a dichotomy on the complexity of CERTAINTY(q), where q is a self-join free acyclic boolean conjunctive query, and that the complexity of CERTAINTY(q) can be classified by the following condition:

Conjecture 1 *Let q be a boolean acyclic and self-join free conjunctive query with k atoms. Let q be such that* CERTAINTY(q) *is not first-order expressible. The complexity of* CERTAINTY(q) *is determined by the following criterion:*

- *If there are two atoms R_i, R_j such that:*

  1. *There is a cycle of length two between R_i, R_j in the attack graph, and*
  2. *$key(R_i) \not\subseteq (vars(R_j))^+$ or $key(R_j) \not\subseteq (vars(R_i))^+$, where the closures are taken in Σ,*

  then CERTAINTY(q) *is coNP-complete.*

- *Otherwise,* CERTAINTY(q) *is in P.*

Observe that the necessary and sufficient condition for CERTAINTY(q) to be coNP-complete, as stated by Conjecture 1, coincides with sufficient condition for intractability proven in Theorem 6. Therefore, if our dichotomy conjecture is correct, then Theorem 6 takes care of the intractability side of the dichotomy, and the gap remains on the side of queries with more than two atoms for which CERTAINTY(q) is in P but not first-order expressible. In fact, there are queries with more than two atoms for which CERTAINTY(q) is not first-order expressible and CERTAINTY(q) is expected to be in P according to Conjecture 1, but for which it has not been proven that CERTAINTY(q) is in P. One such example is the query $q() : -R_1(\underline{x}, y), R_2(\underline{y}, x), R_3(\underline{x, v, w}, y), R_4(\underline{x}, z, v)$.

4.5.2 Evidence to the Dichotomy Conjecture

As a first piece of evidence that our conjecture might be correct, observe that the dichotomy for two-atom queries presented in Section 4.2, and the sufficient conditions for tractability and intractability of CERTAINTY(q) presented in Section 4.3 and Section 4.4 are implied by Conjecture 1. In fact, the sufficient conditions for tractability and intractability imply our dichotomy for queries with two atoms, which we explicitly proved in Section 4.2.

Interestingly, Wijsen, independently in [62], has conjectured a condition for the dichotomy of acyclic and self-join free conjunctive queries under primary key constraints, that coincides with our Conjecture 1. He proves a similar result as our Theorem 6. In the tractability side of the dichotomy, he gives a sufficient condition for CERTAINTY(q) to be in P. His tractability result can be briefly summarized as follows: An attack $R_i \rightsquigarrow R_j$ in the attack graph of a given query is called *strong* if $key(R_j) \not\subseteq vars(R_i)^+$; it is called *weak* otherwise. A directed cycle in the attack graph is a strong cycle if at least one attack in the cycle is strong; it is called weak otherwise. A directed cycle in the attack graph is called *terminal* if the attack graph contains no directed edge from a vertex in the cycle to a vertex outside the cycle; it is *nonterminal* otherwise. Finally, Wijsen proves that if all cycles in the attack graph of a given query q are weak and terminal, then CERTAINTY(q) is in P. There are examples of queries whose attack graph contains cycles that are all weak and terminal, but whose complexity cannot be determined by Theorem 7. One such example is the query $q() : -R_1(\underline{x, z}, y), R_2(\underline{y, z}, x), R_3(\underline{u}, z)$. The attack graph of this query contains the attacks $R_1 \rightsquigarrow R_2$, $R_2 \rightsquigarrow R_1$, $R_3 \rightsquigarrow R_1$ and $R_3 \rightsquigarrow R_2$. It is easy to check that the only cycle, the cycle between R_1 and R_2, is both weak and terminal. On the other hand, this query does not satisfy the condition of Theorem 7. It is also the case that some query satisfies the condition of Theorem 7 but its attack graph may contain some weak cycle that is nonterminal. For example, the query $q() : -R_1(\underline{x, y}, z), R_2(\underline{y, z}, x), R_3(\underline{x, z}, y)$ is tractable based on Theorem 7, but its attack graph contains weak cycles that are nonterminal. To see this, notice that the attack graph of this query contains a weak cycle between every two atoms.

The sufficient conditions provided by Theorem 7 and by Wijsen in [62], do not capture all acyclic and self-join free queries for which, based on our dichotomy conjecture, CERTAINTY(q) is expected to be in P. One such example is the query $q() : -R_1(\underline{x}, y), R_2(\underline{z}, x, y), R_3(\underline{y}, z)$. The complexity of CERTAINTY(q) cannot be determined by Theorem 7 for this query. Also, its attack graph contains weak cycles between every two atoms in q; hence, every weak cycle of length two is nonterminal.

Koutris and Suciu [45, 46], present a proof of a dichotomy theorem on the complexity of CERTAINTY(q), where q is a boolean self-join free conjunctive query, under a set of *simple key* constraints. The class of simple key constraints is a special class of primary key constraints that have the following form: in every atom, either the primary key is a single attribute, or all attributes form the key. In addition, they assume that on a given database, it might be the case that some relation does not violate its primary key. If this is the case, the corresponding atom in the query is marked as *consistent*. More precisely, if R is an atom in q and the primary key of R is satisfied in I, then the atom R in the query is marked as R^c. It is interesting that for the class of acyclic and self-join free conjunctive queries and simple key constraints, their dichotomy condition coincides with Conjecture 1. This observation further strengthens our intuition on the boundary between tractability and intractability of CERTAINTY(q) of acyclic and self-join free conjunctive queries under primary key constraints. Next, we will present the dichotomy theorem proven in [45, 46], and argue about its equivalence with Conjecture 1 for the case of acyclic and self-join free conjunctive queries with simple keys.

A Dichotomy for self-join free conjunctive queries and simple key constraints. We present this dichotomy as stated in [45]. Initially, the authors show how to simplify the structure of a given conjunctive query q that is self-join free, by transforming it into another self-join free conjunctive query q', such that q' has only binary atoms and single-attribute keys. They prove a fundamentally important relationship between q and q', which is, that there is a first-order reduction from CERTAINTY(q) to CERTAINTY(q'), and vice-versa. Based on this result, it suffices to focus on queries with binary atoms and single-attribute key constraints, and prove a dichotomy for this class only. Next, they describe how to build a graph representation $G[q']$ for the transformed query q', and provide a criterion that can be efficiently checked on the graph to determine the complexity of CERTAINTY(q') as being either in P or coNP-complete.

We describe next the construction of the *query graph* of a self-join free conjunctive query q with all binary atoms and single-attribute key constraints.

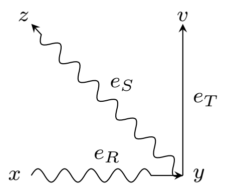


Figure 4.6: Graph of query $q() : -R(\underline{x}, y), S(\underline{y}, z), T^c(\underline{y}, v)$

The graph $G[q]$ is constructed as follows:

- The set of vertexes $V(q)$ is the set of all variables in the query.
- For every atom $R(\underline{u_R}, v_R)$, where R may be inconsistent or not, add a directed edge $e_R(u_R, v_R)$, where u_R is the starting point and v_R is the ending point.

Edges in $G[q]$ that have been generated from inconsistent relations are drawn as curly arrows in the graph. Edges that are generated from consistent relations are drawn as simple arrows. Figure 4.6 shows the query graph of $q() : -R(\underline{x}, y), S(\underline{y}, z), T^c(\underline{y}, v)$.

The notation $x \rightarrow y$ is used to denote the fact that there exists a directed path from variable x to variable y such that all edges in the path are consistent edges. Otherwise, the notation $x \rightsquigarrow y$ is used. The authors introduce the following important notion: given a self-join free conjunctive query q, for every atom R in the query, $fd(R)$ is the set $fd(R) = \{v \in V(G) \mid u_R \rightsquigarrow v \ in \ G - \{e_R\}\}$, where G is the graph of q. Intuitively, $fd(R)$ is the set of all variables that are reachable from u_R in the graph, through a directed path that avoids e_R. An interesting observation is that the set $fd(R)$ is in fact the set R^+ introduced by Wijsen in [61], and also explained in Section 4.1.

Next, two important notions are used to express the dichotomy condition:

- [*source-disjoint edges*] Two inconsistent edges $e_R(u_R, v_R)$ and $e_S(u_S, v_S)$ are source-disjoint if u_R and u_S do not belong in the same strongly connected component (SCC) of G.

- [*Unsplittable*] Two edges e_R and e_S are unsplittable if there exists an undirected path P_R between either endpoint of e_R to either endpoint of e_S such that $V(P_R) \cap fd(R) = \emptyset$, and symmetrically a path P_S such that $V(P_S) \cap fd(R) = \emptyset$ [2].

In Figure 4.6, edges e_R and e_S are source-disjoint because x and y cannot appear in the same strongly connected component. Notice that there is no directed path from y to x. Moreover, these edges are not unsplittable. It is easy to see that $fd(R) = \{x\}$ and $fd(S) = \{y\}$. The path $P_R = (y)$ is such that $P_R \cap fd(R) = \emptyset$. On the other hand, every path from either endpoint of e_R to either endpoint of e_S contains the node y, which is in $fd(S)$.

Finally, the dichotomy is formulated as follows:

Theorem 8 ([45]) *If there exists a pair of source-disjoint and unsplittable edges in $G[q]$, then* CERTAINTY(q) *is coNP-complete; Otherwise,* CERTAINTY(q) *is in P.*

We argue that Conjecture 1 and the condition in Theorem 8 are equivalent for any acyclic and self-join free conjunctive query q, under simple key constraints. Here we provide a concise argument that this equivalence holds true for the case when q has

[2] $V(P_R)$ is the set of vertices mentioned in the path P_R, and $V(P_S)$ is the set of vertices mentioned in the path P_S.

only binary atoms, each with a single attribute as a primary key. This argument can be generalized to arbitrary acyclic and self-join free conjunctive queries and simple keys.

To prove the equivalence between Conjecture 1 and the condition in Theorem 8, assuming that the result proven in [45] is correct, it is enough to show that the following holds true:

Proposition 4 *Let q be a boolean, self-join free and acyclic conjunctive query with all binary atoms, each atom with a single attribute as a primary key. Let Σ be a set of simple key constraints. If $G[q]$ contains a pair of source-disjoint and unsplittable edges, then there exist two atoms R_i and R_j in q such that:*

- *there is a cycle of length two between R_i and R_j in the attack graph of q, and*
- *$key(R_i) \nsubseteq vars(R_j)^+$ or $key(R_j) \nsubseteq vars(R_i)^+$*

Proof. (Sketch) If e_{R_i} and e_{R_j} are two source-disjoint edges, then u_{R_i} and u_{R_j} do not belong in the same strongly connected component. Then, there is no directed path from u_{R_i} to u_{R_j}, or there is no directed path from u_{R_j} to u_{R_i}. It follows that R_i and R_j are atoms in q such that $key(R_i) \nsubseteq vars(R_j)^+$ or $key(R_j) \nsubseteq vars(R_i)^+$.

Next, we argue that if e_{R_i} and e_{R_j} are unsplittable, then there is a cycle of two between R_i and R_j in the attack graph. Let τ be a join-tree of q. If there exists an undirected path P_{R_i} between either endpoint of e_{R_i} to either endpoint of e_{R_j} such that $V(P_{R_i}) \cap fd(R_i) = \emptyset$, then, it can be proven by induction that there is a path between R_i and R_j in τ such that $R_i \rightsquigarrow R_k$ for every R_k in this path. Similarly, if there exists an undirected path P_{R_j} between either endpoint of e_{R_j} to either endpoint of e_{R_i} such that

$V(P_{R_j}) \cap fd(R_j) = \emptyset$, then, it can be proven by induction that there is a path between R_i and R_j in τ such that $R_j \rightsquigarrow R_k$ for every R_k in this path. $\square$

Our extensive study of the complexity of CERTAINTY(q), as well as the related work on proving a dichotomy for CERTAINTY(q), suggest that our Conjecture 1 might be correct. However, considering that even the proof for the case of simple key constraints is quite complex [46], it could take considerable effort before one can prove a more general result for acyclic and self-join free conjunctive queries. More recently, Fontaine [28] has established several intriguing results relating the complexity of CERTAINTY(q) to the constraint satisfaction problem (CSP). Dichotomies for different classes of the Constraints Satisfaction Problem have been studied for quite a while, and some of the established results involve highly complex proofs. Thus, the results in [28] suggest that proving a dichotomy for CERTAINTY(q) could be as challenging, and that it might require new proof techniques other than those investigated so far.

Chapter 5

Combined Complexity of Consistent Query Answering

The work we have presented so far, in the previous chapters, concerns data complexity of consistent query answering, where the database is the only input to the complexity. Many problems in databases are typically studied with respect to data complexity, as in reality, the size of the query and the constraints is much smaller than the size of the database. Therefore, the size of the database is considered as the dominant input parameter to the complexity. However, realistic database applications nowadays employ rich schemas with several relations. Thus, queries involving many atoms are quite common in practice. In the database literature, complexity expressed with respect to the size of the instance and query, is typically referred to as *combined complexity.* The analysis of combined complexity of consistent query answering for conjunctive queries and primary keys is the focus of this chapter. Since we infer the relational database schema and constraints from the given query, we define the combined complexity of consistent query answering as having two input parameters: the size of the database, and the size of the query. The size of the query is the number of atoms in the query. The combined complexity of conjunctive consistent query answering is formalized in Definition 10.

Definition 10 COMB-CERTAINTY *is the following decision problem: Given an instance* I*, a conjunctive query* q *and a tuple* t*, is* t *a consistent answer of* q *on* I*?*

When q *is boolean,* COMB-CERTAINTY *is the following decision problem: Given an instance* I *and a conjunctive query* q*, is "true" the consistent answer to* q *on* I*?*

Next, in Section 5.1 we provide a theoretical discussion of the combined complexity of consistent query answering for conjunctive queries and primary keys.

5.1 Conjunctive Queries and Primary Key Constraints

In Section 3.2.2, in Theorem 2 we gave a reduction from the consistent answers of any arbitrary conjunctive query, to Binary Integer Programming. We also analyzed the size the BIP programs generated by Theorem 2, and argued that these programs have always size polynomial in the size of the database instance. On the other hand, our analysis reveals that the size of the programs is exponential in the number of atoms of the conjunctive query. Note that in Theorem 2, we generate an inequality constraint for every minimal witness, and the number of minimal witnesses is exponential in the number of atoms in the query. The reduction of Theorem 2 is an exponential reduction of COMB-CERTAINTY to Binary Integer Programming. Thus, we do not expect EQUIP to scale well on conjunctive queries with many atoms. This observation motivates the following question: *Is it possible to reduce* COMB-CERTAINTY *to Binary Integer Programming, in polynomial time?* In fact, it is unlikely that a polynomial reduction from COMB-CERTAINTY to BIP exists, as the complexity of COMB-CERTAINTY turns out to be Π_2^P-complete. We state and prove this result in Theorem 9.

Theorem 9 COMB-CERTAINTY *is* Π_2^p*-complete.*

PROOF. Initially, we argue about membership in class Π_2^p. Membership in Π_2^p is a direct consequence of the following two facts: 1) repair checking is in polynomial time for primary key constraints, and 2) given an instance r and a boolean conjunctive query q, checking $r \models q$ is in NP. We prove hardness via a reduction from Π_2^{3SAT}. For a fixed relational schema $\mathbf{R}$ and constraints Σ, given a formula Ψ of the form $\forall p_1, \cdots, p_n$, $\exists p_{n+1}, \cdots, p_m . \Phi_1 \wedge \cdots \wedge \Phi_k \wedge \cdots \wedge \Phi_s$ where each clause Φ_k has three literals, we construct in polynomial time a boolean conjunctive query q and a database I such that Ψ is satisfiable if and only if *true* is the consistent answer to q on I.

The relation schema $\mathbf{R}$ consists of a binary relation U and four ternary relations R_0, R_1, R_2, R_3. The first attribute of U is a primary key. No primary keys are defined on the other relations. Next, we will show how to construct I and q. We will assume that in each clause, literals in negated form appear before literals in un-negated form.

a) Construct I as follows:

- For every universally quantified variable p_i, add to I the facts $U(p_i, 0)$ and $U(p_i, 1)$

- Let $R_0 = \{0,1\}^3 - \{(0,0,0)\}$, $R_1 = \{0,1\}^3 - \{(1,0,0)\}$, $R_2 = \{0,1\}^3 - \{(1,1,0)\}$, and $R_3 = \{0,1\}^3 - \{(1,1,1)\}$ where $\{0,1\}^3$ is the set of ternary tuples with values from $\{0,1\}$.

b) Construct boolean q as follows:

- For every universally quantified variable p_i add atom $U(\text{“}p_i\text{”}, x_i)$
- For every clause Φ_k add an atom $F_k = R_k(x_h, x_i, x_j)$, where:

  $R_k = R_0$ when $\Phi_k = (p_h \vee p_i \vee p_j)$,

  $R_k = R_1$ when $\Phi_k = (\neg p_h \vee p_i \vee p_j)$,

  $R_k = R_2$ when $\Phi_k = (\neg p_h \vee \neg p_i \vee p_j)$,

  $R_k = R_3$ when $\Phi_k = (\neg p_h \vee \neg p_i \vee \neg p_j)$

We argue that Ψ is not satisfiable if and only if q is false in some repair of I.

($\Rightarrow$) In this direction, we assume that Ψ is not satisfiable. Then, there exists an assignment ν to the universally quantified variables $p_1, \cdots, p_n$, such that there exists no assignment θ to the existentially quantified variables $p_{n+1}, \cdots p_m$ so that ν, θ together form a truth assignment for Ψ. From ν, we construct a repair r of I as follows: a) We add to r all facts from relations R_0 - R_3, and b) We add to r all facts $U(\text{“}p_i\text{”}, \nu(p_i))$, for $1 \leq i \leq n$. We argue that r is a repair, and that q is false in r. It is easy to see that r is a repair, since no constraints exist on R_0-R_3, and for every key-value in U exactly one fact $U(\text{“}p_i\text{”}, \nu(p_i))$ exists. Let us assume, towards a contradiction, that q is true on r. Then there exists a mapping θ of variables $x_1, \cdots, x_m$ to $\{0,1\}$ such that: a) for every atom $F_k = R_k(x_h, x_i, x_j)$ in q, where $0 \leq k \leq 3$, the fact $R_k(\theta(x_h), \theta(x_i), \theta(x_j))$ is in r, and b) for every atom $U(“p_i”, x_i)$ in q, the fact $U(“p_i”, \theta(x_i))$ is in r. It is easy to see that θ is a truth assignment to Ψ in this case, and that for every universally quantified variables $p_1, \cdots, p_i, \cdots p_n$, we have that $\theta(x_i) = \nu(x_i)$, for $1 \leq i \leq n$. It is clear now that we have violated the assumption.

($\Leftarrow$) In the opposite direction, assume that r is a repair of I on which q is false. Let ν be an assignment to the universally quantified variables $p_1, \cdots, p_n$ defined as follows: $\nu(p_i) = 1$ if and only $U(“p_i”, 1)$ is in r. We argue that there exists no assignment θ to the existentially quantified variables $p_{n+1}, \cdots p_m$ so that ν, θ together form a truth assignment for Ψ. Assume towards a contradiction that such assignment θ exists. Let S be a set of facts from r constructed as follows: a) S contains all U-facts of r, and b) for every atom $F_k = R_k(x_h, x_i, x_j)$ in q, where $0 \leq k \leq 3$, S contains the fact $R_k(\theta(x_h), \theta(x_i), \theta(x_j))$. It is easy to see that the set S is a minimal witness to q on R, thus contradicting the assumption that q is false in r. $\square$

Theorem 9 tells us that there is little hope in finding a polynomial reduction from COMB-CERTAINTY to BIP. However, one may wonder if such a polynomial reduction is possible for a sub-class of conjunctive queries. Observe that Theorem 9 constructs from Ψ a conjunctive query that may be cyclic. For instance, let $\Psi = \forall p_1, p_2, p_3, \exists p_4, p_5, p_6.(p_4 \vee p_5 \vee p_1) \wedge (\bar{p}_4 \vee p_6 \vee p_2) \wedge (\bar{p}_5 \vee \bar{p}_6 \vee p_3)$. Then q is the query $q = U(“p_1”, x_1), U(“p_2”, x_2), U(“p_3”, x_3), R_0(x_4, x_5, x_6), R_1(x_4, x_6, x_2), R_2(x_5, x_6, x_3)$. It is easy to check that the intersection graph of q contains a cycle between atoms R_0, R_1

and R_2, which is not possible to break without violating the connectedness condition.

As mentioned earlier, acyclic conjunctive queries are a class of conjunctive queries that possess several good properties, one of those being the fact that acyclic conjunctive query evaluation can be done in PTIME combined complexity [8]. This result can be used to argue that, for the class of acyclic conjunctive queries, COMB-CERTAINTY is coNP-complete. The formal argument is presented in Theorem 10. We denote by AC_CERTAINTY the decision problem COMB-CERTAINTY for acyclic conjunctive queries.

Theorem 10 AC_CERTAINTY *is coNP-complete.*

PROOF. The coNP-hardness of AC_CERTAINTY follows from the fact that we can get coNP-hardness even when the query is fixed, i.e., from coNP-hardness of CERTAINTY(q). To show membership in coNP, note that one can check that *true* is not the consistent answer of q, by first guessing a repair r, and then checking if $q(r) = false$. Repair checking can be done in polynomial time. Moreover, $q(r)$ can be evaluated in polynomial time w.r.t. the size of r and q. Hence, the complement of AC_CERTAINTY is in NP. □

Theorem 10, shows that the good properties of acyclic conjunctive queries carry over to the computation of the consistent answers as well, as the complexity of AC_CERTAINTY is in a lower complexity class than COMB-CERTAINTY. It follows from Theorem 10 that there must exist a polynomial reduction from AC_CERTAINTY to BIP. In the remainder of this chapter we investigate polynomial reductions from AC_CERTAINTY to BIP, and rely on them to implement an approach for consistent query answering that scales well on queries with many atoms.

5.2 Better Heuristics for Consistent Answers of Acyclic Conjunctive Queries

In this section, we present a new, alternative strategy for modeling the problem AC_CERTAINTY with BIP. Ultimately, we will show how, given an acyclic conjunctive query and a database instance, one can construct a polynomial-size system of linear equalities and inequalities, whose solutions represent all possible repairs (similarly as in Theorem 2). Then, we will adapt Algorithm EliminatePotentialAnswers to use this new system for the case in which the query is acyclic, instead of System (2) used for arbitrary conjunctive queries. Before presenting formally the reduction, we explain the intuition with a simple example.

5.2.1 Acyclic and Self-Join Free Boolean Conjunctive Queries

We first handle the case of acyclic conjunctive queries that contain no self-joins. In Theorem 11 we will give an alternative reduction of AC_CERTAINTY to BIP, for the case when q is self-join free. Before we present this reduction formally, we bring to the reader's attention Lemma 5 of Section 4.4, where we gave a polynomial reduction from CERTAINTY(q) of acyclic and self-join free conjunctive queries, to the Maximum Independent Set problem. We also argued that the size of the conflict-join graph is polynomial in the size of the database and query. Therefore, for self-join free acyclic conjunctive queries, Lemma 5 is a polynomial reduction of AC_CERTAINTY to the Maximum Independent Set problem. The reduction to BIP that we will present next, in essence, it models the Maximum Independent Set problem on the conflict-join graph $H_{I,q,\tau}$, with Binary Integer Programming.

Theorem 11 *Let q be a self-join free and acyclic boolean conjunctive query. Let τ be a join tree of q. Given a database instance I over the same schema as q, we construct in polynomial time the following system of linear equalities and inequalities.*

System (3):

Variables:
$x_f^{R_i} \in \{0,1\}$ *for every R_j-fact f, and R_i adjacent with R_j in τ*

Constraints:

(a) $\displaystyle\sum_{f\in K;\ R_i\in N_\tau(R_j)} x_f^{R_i} = 1,$ *for every relation R_j in I and K a key-equal group of R_j.*

(b) $x_f^{R_i} + x_g^{R_j} \leq 1,$ *for every edge (R_j, R_i) in τ and $\{f, g\}$ a minimal witness of $q_{\{R_j,R_i\}}$.*

Then the following statements are equivalent:

- *There is a repair r of I such that q is false on r.*
- *System (3) has a solution.*

Proof. To show that System (3) has a solution if and only if there exists a repair on which q is false, we will show that System (3) is an encoding of the problem of finding an independent set on the conflict-join graph $H_{I,q,\tau}$, with size equal to the number of key-equal groups in I. In Lemma 5, we have already proven that such an independent set gives rise to a repair on which q is false, and vice-versa.

Let $H_{I,q,\tau}$ be the conflict-join graph with vertex set $V = K_1 \cup \cdots \cup K_s$ and edge set $E = E_c \cup E_w$, where: $K_1, \cdots, K_s$ are the sets of vertices generated from key-equal groups; E_c is the set of edges between key-equal facts; and E_w is the set of edges generated from minimal witnesses. We can use variables $x_f^{R_i}$ in System (3) to represent vertices f^{R_i} in $H_{I,q,\tau}$. Constraints (a) encode that exactly one vertex from each K_p, $1 \leq p \leq s$, appears in an independent set of size s. Constraints (b) encode that from each minimal witness $\{f, g\}$ of some $q_{\{R_j,R_i\}}$, not both of $x_f^{R_j}$ and $x_g^{R_i}$ are in an independent set. We show that the following statements are equivalent:

1. There is a maximum independent set of $H_{I,q,\tau}$ that has size equal to s.

2. System (3) has a solution.

The proof follows from the fact that there is a one-to-one correspondence between each maximum independent set of size s and each solution of System (3). In one direction, if M is an independent set of size s, let $\hat{x}$ be the solution formed by assigning $\hat{x}_f^{R_i} = 1$ if and only if f^{R_i} is in M. Because M is independent, then for every edge $(f^{R_i}, g^{R_j}) \in E_w$, at most one of f^{R_i}, g^{R_j} is in M. Therefore, $\hat{x}_f^{R_i} + \hat{x}_g^{R_j} \leq 1$ is satisfied. Also, since M is independent and it has size s, from each clique K_p, exactly one vertex is in M. Therefore, constraints (a) are satisfied. In the opposite direction, if $\hat{x}$ is a solution of System (3), let M be the set of vertices such that $f^{R_i} \in M$ if and only if $\hat{x}_f^{R_i} = 1$. Since the constraints (a) are satisfied, then for every K_p, exactly one vertex is in M. Because constraints (b) are satisfied, then for every edge $(f^{R_i}, g^{R_j}) \in E_w$, at most one of f^{R_i}, g^{R_j} is in M. Obviously, M is independent and has size equal to s. $\square$

We illustrate Thoerem 11 with a simple example.

Example 8 Let us consider the query $q() : -R_1(\underline{x}, y, z), R_2(\underline{y}, x, z), R_3(\underline{z}, x, y)$ and database $I = \{f_1 = R_1(a, b, c), f_2 = R_1(a, b', c),\ f_3 = R_2(b, a, c),\ f_4 = R_2(b, a, c'), f_5 = R_3(c, a, b), f_6 = R_3(c, a, b'')\}$. A join tree τ for q is the path $\langle R_1, R_2, R_3\rangle$. We construct the system of constraints as described in Theorem 11:

$$(a)\quad \begin{aligned} &x_{f_1}^{R_2} + x_{f_2}^{R_2} = 1 \\ &x_{f_3}^{R_1} + x_{f_3}^{R_1} + x_{f_4}^{R_3} + x_{f_4}^{R_3} = 1 \\ &x_{f_5}^{R_2} + x_{f_6}^{R_2} = 1 \end{aligned} \qquad\qquad (b)\quad \begin{aligned} &x_{f_1}^{R_2} + x_{f_3}^{R_1} \leq 1 \\ &x_{f_3}^{R_3} + x_{f_5}^{R_2} \leq 1 \end{aligned}$$

The instance $r = \{f_2, f_4, f_5\}$ is a repair of I that does not satisfy q. From r we can assign values to the variables of the program as follows: $x_{f_1}^{R_1} = 0, x_{f_2}^{R_2} = 1, x_{f_3}^{R_1} = 0, x_{f_3}^{R_3} = 0, x_{f_4}^{R_1} = 1, x_{f_4}^{R_3} = 0, x_{f_5}^{R_2} = 1, x_{f_6}^{R_2} = 0$. It is easy to check that the assigned values satisfy all of the constraints. □

5.2.2 Acyclic and Self-Join Free non-Boolean Conjunctive Queries

Next, we shift our attention to the case when q is non-boolean. Again, we will follow a similar strategy as in Section 3.2.2, Theorem 2, to construct a single BIP instance from q and I, where the solutions to the constraints model the repairs of the database and provide information about the potential answers that are not found as query answers in each repair.

Theorem 12 *Let q be a self-join free, acyclic conjunctive query. Let τ be a join tree of q. Given a database instance I over the same schema as q, we construct in polynomial* *time the following system of linear equalities and inequalities:*

System (4):

<u>*Variables*</u>:

$x_f^{R_i} \in \{0,1\}$	*for every R_j-fact f, and R_i adjacent with R_j in τ*
$w_{f,g} \in \{0,1\}$	*for every (R_i, R_j) adjacent in τ, and $\{f,g\}$ a minimal witness of $q_{i,j} = \{R_j, R_i\}$*
$u_{\mathbf{a}} \in \{0,1\}$	*for every $\mathbf{a} \in q(I)$*

<u>*Constraints*</u>:

(a) $\displaystyle\sum_{f \in K;\ R_i \in N_\tau(R_j)} x_f^{R_i} = 1,$	*for every relation R_j in I and every key-equal group K of R_j.*
(b) $x_f^{R_i} + x_g^{R_j} - w_{f,g} \leq 1,$	*for every (R_j, R_i) edge in τ, and $\{f,g\}$ minimal witness of $q_{\{R_j,R_i\}}$.*
(c) $\displaystyle\sum_{\mathbf{a} \in U_{f,g}} u_{\mathbf{a}} \geq \lvert U_{f,g} \rvert \cdot w_{f,g},$	*for every (R_j, R_i) edge in τ, and $\{f,g\}$ a minimal witness of $q_{\{R_j,R_i\}}$. $U_{f,g}$ is the set $\{\mathbf{a} : \text{exists } S \subseteq I \text{ s.t. } q(S) = \{\mathbf{a}\} \text{ and } f, g \in S\}$.*

Then the following statements are equivalent:

- *There is a repair r of I such that $q[\mathbf{a}]$ is false on r.*
- *System (4) has a solution $(\hat{x}, \hat{w}, \hat{u})$ such that $\hat{u}_{\mathbf{a}} = 0$.*

Proof. ($\Rightarrow$) In this direction we argue that if $q[\mathbf{a}]$ is false on some repair r, then System (4) has a solution $(\hat{x}, \hat{w}, \hat{u})$ that assigns value 0 to $\hat{u}_{\mathbf{a}}$. Since $q[\mathbf{a}]$ is a boolean query, then, System (3) of Theorem 11 has a solution $\hat{x}$. Let $\hat{u}$ be such that $\hat{u}_{\mathbf{a}} = 0$ and $\hat{u}_{\mathbf{b}} = 1$ for every $\mathbf{b} \in q(I)$ such that $\mathbf{b} \neq \mathbf{a}$. Let $\hat{w}_{f,g} = 0$ if and only if f and g are two facts that appear in some minimal witness S of $q[\mathbf{a}]$. We will argue that $(\hat{x}, \hat{w}, \hat{u})$ is a solution to System (4). Because the constraints (a) are identical in both systems, we need only argue that constraints (b) and (c) are satisfied. From a pair of facts $\{f,g\}$ that appear together in a minimal witness of q, an inequality of type (b) and an inequality of type (c) is generated. Let f be an R_j-fact and let g be an R_i-fact. One possibility is that there exists a minimal witness S of $q[\mathbf{a}]$ that contains f and g. In this case, there must be an inequality $x_f^{R_i} + x_g^{R_j} \leq 1$ in System (3). Because $\hat{x}$ satisfies this inequality, then $(\hat{x}, \hat{w})$ trivially satisfies the inequality $x_f^{R_i} + x_g^{R_j} - w_{f,g} \leq 1$. Moreover, since we have assigned 0 to $\hat{w}_{f,g}$, then $\sum_{\mathbf{a} \in U} u_{\mathbf{a}} \geq |U_{f,g}| \cdot w_{f,g}$ is trivially satisfied. The other possible scenario for the pair $\{f, g\}$ is that there exists no minimal witness of $q[\mathbf{a}]$ that contains both these facts. In this scenario, it might happen that both $\hat{x}_f^{R_i}$ and $\hat{x}_g^{R_j}$ are equal to 1. But, since we have assigned 1 to $\hat{w}_{f,g}$ in this case, then the inequality $x_f^{R_i} + x_g^{R_j} - w_{f,g} \leq 1$ is satisfied by $(\hat{x}, \hat{w})$. The inequality of type (c) generated from the pair $\{f, g\}$, is such that every variable in the left-hand side is different from $u_{\mathbf{a}}$. Since $\hat{u}_{\mathbf{b}} = 1$ for every $\mathbf{b} \neq \mathbf{a}$, then the sum of terms in the left-hand side equals $|U_{f,g}|$. Because $\hat{w}_{f,g} = 1$, the inequality is satisfied.

($\Leftarrow$) In the opposite direction, we show that if $(\hat{x}, \hat{w}, \hat{u})$ is a solution such that $\hat{u}_{\mathbf{a}} = 0$, then the potential answer $\mathbf{a}$ is not found as an answer to q on some repair. Towards this goal, it is enough to show that $(\hat{x}, \hat{u})$ is a solution to System (3) generated from $q[\mathbf{a}]$ and I. Obviously, $\hat{x}$ satisfies the constraints (a) in System (3). Let f and g be two facts that appear together in some minimal witness of $q[\mathbf{a}]$. Assume f is an R_j-fact and g is an R_i-fact. In System (4) there must exist an inequality $x_f^{R_i} + x_g^{R_j} - w_{f,g} \leq 1$ and an inequality $\sum_{\mathbf{a} \in U} u_{\mathbf{a}} \geq |U_{f,g}| \cdot w_{f,g}$. In the later, the variable $u_{\mathbf{a}}$ must appear in the left-hand side. Since $\hat{u}_{\mathbf{a}} = 0$, then $\hat{w}_{f,g}$ must be 0 in order for this inequality to be satisfied. Since $\hat{w}_{f,g} = 0$ and $\hat{x}_f^{R_i} + \hat{x}_g^{R_j} - \hat{w}_{f,g} \leq 1$, then it must be that $\hat{x}_f^{R_i} + \hat{x}_g^{R_j} \leq 1$. It is obvious now that $(\hat{x}, \hat{u})$ satisfies the constraints (b) in System (3).

Finally, we argue that System (4) in Theorem 12 has size polynomial in the size of the database and the size of the query. We assume that the arity of the query is fixed and the size of the query is the number of atoms in it. Let n be the size of a relation in a given database I. Let q be a query with arity s and with k atoms. If τ is a join tree for q, then the number of edges in τ is $k-1$ [1]. For every two atoms R_i and R_j that are connected by an edge in τ, a distinct variable is used to represent the R_i-facts and the R_j-facts. Since there are $k-1$ edges in τ, there are $2n(k-1)$ variables in System (4) that represent database facts. The number of potential answers to q is n^s. So, there are n^s variables of the form $u_{\mathbf{a_j}}$. The total number of variables is $2n(k-1)+n^s$. Since s is fixed, the number of variables is polynomial in the size of the database and the query. There is a constraint of type (a) for every key-equal group. There are at most n key-equal groups per relation. The number of constraints (a) is kn. One inequality of type (b) and one inequality of type (c) is constructed for every pair of facts f, g such that $f \in R_j^I$, $g \in R_i^I$, there is an edge (R_t, R_m) in τ and $\{f, s\}$ is a minimal witness of q_{R_i,R_j}. Therefore, there are at most $(k-1)n^2$ constraints of type (b). The number of constraints in System (4) is obviously polynomial in the size of the database and the size of the query. $\square$

5.2.3 Acyclic Conjunctive Queries Containing Self-Joins

Theorem 12 applies to acyclic conjunctive queries that have no repeated relation names. Here, we show how to handle repeated relation names.

[1]the number of edges in a tree with k nodes is $k-1$

So far, under the assumption that the query was self-join free we have used relation names to refer to atoms in the query. In the presence of self-joins, we need to distinguish between atoms and relations. Let $\mathbf{R}$ be a relation schema with relation names $\{R_1, \cdots, R_m\}$. Given a conjunctive query q over the schema $\mathbf{R}$, we will denote atoms in q using $F_1, \cdots, F_i, \cdots, F_k$, where each atom F_i is of the form $F_i = R_{p_i}(\mathbf{v_i})$, where $1 \leq p_i \leq m$. Note that the join tree of an acyclic conjunctive query has the atoms of q as nodes. So, $F_1, \cdots, F_k$ are the nodes of a join tree of q. Given a conjunctive query q with atoms $F_1, \cdots, F_k$ and a database instance I over the schema of q, we say that a fact f is an F_i-fact if F_i is an atom $F_i = R_{p_i}(\mathbf{v_i})$ in q and f is an R_{p_i}-fact in I.

Next, we will show how to modify System (4) to handle queries that may contain self-joins. The formulation of the constraints becomes quite involved in this case, as we need to distinguish between atoms and relation names. However, the intuition is quite simple. First, we construct constraints (a), (b) and (c) as in System (4), assuming every relation name is distinct, i.e., assuming each occurrence of the same relation R is conceptually referring to a different copy of R in the database. So far, a solution to (a), (b) and (c) encodes a database instance that does not satisfy q, but that is not necessarily a repair. The reason is because a solution to (a), (b) and (c) may be encoding a repair on which key-equal facts f and g are selected from two independent copies of the same relation. To avoid this scenario, we add more equality constraints that guarantee that in each copy of the same relation, the same fact is chosen from a key-equal group.

Theorem 13 *Let $\mathbf{R}$ be a relation schema with relation names $\{R_1, \cdots, R_m\}$. Let $q(\mathbf{z}) :- F_1, \cdots, F_k$ be a given acyclic conjunctive query with atoms $F_1, \cdots, F_k$, where each atom F_j is of the form $F_j = R_{p_j}(\mathbf{v_j})$, for $1 \le p_j \le m$. Let τ be an arbitrary join tree of q. Let I be a given a database instance. We construct in polynomial time the following system of linear equalities and inequalities:*

System (5):

<u>*Variables*</u>*:*

$x_f^{F_i} \in \{0,1\}$ *for every F_j-fact f, and F_i adjacent with F_j in τ*

$w_{f,g} \in \{0,1\}$ *for every (F_i, F_j) adjacent in τ, and $\{f, g\}$ a minimal witness of $q_{i,j} = \{F_j, F_i\}$*

$u_{\mathbf{a}} \in \{0,1\}$ *for every $\mathbf{a} \in q(I)$*

<u>*Constraints*</u>*:*

(a) $\displaystyle\sum_{\substack{f \in K \\ F_i \in N_\tau(F_j)}} x_f^{F_i} = 1,$ *for every atom $F_j = R_{p_j}(\mathbf{v_j})$ and K a key-equal group of R_{p_j}.*

(a') $\displaystyle\sum_{\substack{f \in K \\ F_i \in N_\tau(F_j)}} x_f^{F_i} = \sum_{\substack{f \in K \\ F_i \in N_\tau(F_t)}} x_f^{F_i}$

for every F_j, F_t s.t. $F_j = R_{p_j}(\mathbf{v_j})$, $F_t = R_{p_t}(\mathbf{v_t})$ and $p_j = p_t$, and for every key-equal group K in R_{p_j} (or, R_{p_t}).

(b) $x_f^{F_i} + x_g^{F_j} - w_{f,g} \le 1,$ *for every (F_j, F_i) adjacent in τ and $\{f, g\}$ a minimal witness of $q_{i,j} = \{F_j, F_i\}$.*

(c) $\displaystyle\sum_{\mathbf{a} \in U_{f,g}} u_{\mathbf{a}} \ge |U_{f,g}| \cdot w_{f,g}$ *for every (F_j, F_i) edge in τ, and $\{f, g\}$ a minimal witness of $\{F_j, F_i\}$, where $U_{f,g} = \{\mathbf{a} :$ exists S a minimal witness of $q[\mathbf{a}]$ s.t. $f \in S$ and $g \in S$ $\}$*

Then the following statements are equivalent:

- *There is a repair r of I such that $q[\mathbf{a}]$ is false on r.*
- *System (5) has a solution $(\hat{x}, \hat{w}, \hat{u})$ such that $\hat{u}_{\mathbf{a}} = 0$.*

Proof. System (5) is different from System (4), mainly because it contains constraints (a'). We will prove this theorem using Theorem 12. From q, we construct a new self-join free query q' with atoms $F_j(\mathbf{v_j})$. i.e., we are using the atom names of q as relation names of q'. Moreover, let I' be the database constructed from I by copying F_j-facts of I into I'. So, each relation of I is copied into I' as many times as the relation name appears in q. We make the following observation about q and q':

Observation 1: There exists a repair r on which $q[\mathbf{a}]$ is false, if and only if there exists a repair r' of I' on which $\mathbf{a}$ is not an answer to q', and r' is such that for every two facts $F_i(t)$ and $F_j(s)$, if F_i and F_j mention the same relation name in q, then $t = s$.

Proof of Observation 1 Let r be a repair of I, on which $\mathbf{a}$ is not found as an answer to q. We construct a repair r' of I' by adding an F_j-fact to r' for every F_j-fact of r. Observe that r' is such that, if F_i and F_j are atoms in q that mention the same relation name, then the relations F_i and F_j in r' have the same tuples. Assume towards a contradiction that there exists a minimal witness S' of $q'[\mathbf{a}]$ in r'. Construct $S \subseteq I$ by adding an F_i-fact if and only if there is an F_i-fact in r. Then, obviously, S is contained in r and S is a minima witness of $q[\mathbf{a}]$. In the opposite direction, let r' be a repair of I' on which $q[\mathbf{a}]$ is false, and such that F_i and F_j in r' have the same tuples whenever F_i and F_j are atoms in q that mention the same relation name. A repair r can be constructed by adding to it an F_i-fact for every F_i-fact in r'. Again, if S is a minimal witness of $q[\mathbf{a}]$, a minimal witness S' of q' can be constructed by adding to S' an F_i-fact for every F_i-fact in r.

To prove our theorem, after proving Observation 1, it suffices to show that System (5) has a solution $(\hat{x}, \hat{w}, \hat{u})$ such that $\hat{u}_{\mathbf{a}} = 0$, if and only if there exists a repair r' of I' on which $\mathbf{a}$ is not an answer to q', and r' is such that for every two facts $F_i(t)$ and

$F_j(s)$, if F_i and F_j mention the same relation name in q, then $t = s$. In one direction, from Theorem 12 we know that every repair r' of I' can be mapped into a solution $(\hat{x}, \hat{w}, \hat{u})$ to System (4). Obviously, $(\hat{x}, \hat{w}, \hat{u})$ is a solution to constraints (a), (b) and (c) of System (5). Additionally, if r' is such that $t = s$ for every two facts $F_i(t)$ and $F_j(s)$, where F_i and F_j mention the same relation name in q, then it follows straightforwardly that constraints (a') are also satisfied by $(\hat{x}, \hat{w}, \hat{u})$. In the other direction, a solution $(\hat{x}, \hat{w}, \hat{u})$ to System (5), is obviously a solution to System (4) constructed for q' and I'. Then, this solution can be mapped into a repair r' of I' on which $q[\mathbf{a}]$ is false. Because $(\hat{x}, \hat{w}, \hat{u})$ satisfies the constraints (a'), it follows that r' has the property that for every two facts $F_i(t)$ and $F_j(s)$, if F_i and F_j mention the same relation name in q, then $t = s$.

Finally, it is easy to see that number of constraints (a') is polynomial in the size of the database and query, since in the worst case, when all atoms mention the same relation name, there will be quadratically many equalities of type (a'). $\square$

5.2.4 Algorithm for Consistent Answers of Acyclic Conjunctive Queries

In Section 3.3, we gave an algorithm for computing the consistent answers to conjunctive queries using BIP. In Algorithm EliminatePotentialAnswers, we used the BIP program generated as described in Theorem 2 as a building block. The System (5) generated as described in Theorem 13, is an alternative method of encoding CERTAINTY(q) when q is acyclic. To be able to still apply the strategy of Algorithm EliminatePotentialAnswers for iteratively filtering false potential answers, we need to show that Algorithm EliminatePotentialAnswers correctly computes the consistent answers, even if we use System (4) instead of System (3), when the input query is acyclic and self-join free. In Figure 5.1 we present Algorithm EliminatePotentialAnswers-AC, which uses Theorem 12 instead of Theorem 2 to compute the consistent query answers to acyclic and self-join free conjunctive queries. In Theorem 14 we prove correctness of the algorithm.

Theorem 14 *Let q be an acyclic and self-join free conjunctive query and I a database instance. Then, Algorithm* EliminatePotentialAnswers-AC *computes exactly the consistent query answers to q on I.*

Proof. The proof will make use of the following two loop invariants:

1. At the i-th iteration, every optimal solution to P_i is also a solution to the constraints C.

2. At the end of the i-th iteration, if *filter* is *true* then the number of elements in consistent that are *false* is at least i.

The first loop invariant follows easily from the fact that $\mathcal{C}_i$ contains all the constraints of $\mathcal{C}$. The second loop invariant is proved by induction on i. Assume that at the end of the i-th iteration, the number of *false* elements in consistent is greater than or equal to i. For every $j \in [1..p]$ such that consistent$[j]$ is *false*, there must exist a constraint $u_{\mathbf{a_j}} = 1$ in $\mathcal{C}_i$. We will show that at the termination of iteration $i+1$, if *filter* is *true*, then the number of elements in consistent that are *false* is greater than or equal to $i+1$. Since *filter* is *true*, the BIP engine has returned an optimal solution

Algorithm **EliminatePotentialAnswers-AC**

```
1. Input
        $q$ : acyclic conjunctive query
        $I$ : database over the same schema as $q$
        $\mathcal{C}$: the set of constraints constructed from $q, I$ as described in Theorem 13
        $\{\mathbf{a_1}, \cdots, \mathbf{a_p}\}$: the set of potential answers to $q$ on $I$

2. let Consistent be a boolean array with subscripts $1, \ldots, p$. Consistent$[j]$
represents the element $a_j$ and every entry is initialized to true.
3. let $i := 1$
4. let filter:=true
5. let $\mathcal{C}_1 := \mathcal{C}$
6. while (filter=true)
7.     let $P_i = \min\{\sum_{j \in [1..p]} u_{\mathbf{a_j}} | \text{subject to } \mathcal{C}_i\}$
8.     Evaluate $P_i$ using BIP engine
9.     let $(x^*, u^*)$ be an optimal solution for $P_i$
10.    let $\mathcal{C}_{i+1} := \mathcal{C}_i$
11.    let filter:=false
12.    for $j := 1$ to $p$
13.        if $u^*_{\mathbf{a_j}} = 0$ then
14.            let Consistent[j]:=$false$
15.            Add to system $\mathcal{C}_{i+1}$ the equality $(u_{\mathbf{a_j}} = 1)$
16.            let filter:=true
17.    let $i := i + 1$

18. for $j := 1$ to $p$
19.    if Consistent$[j]$ = true
20.        return $\mathbf{a_j}$
```

Figure 5.1: Algorithm for computing the consistent query answers to acyclic conjunctive queries by eliminating potential answers.

(x^*, w^*, u^*) to P_{i+1} such that $u^*_{\mathbf{a_j}} = 0$ for at least some $j \in [1..p]$. Notice that it is not possible that at some previous iteration, CONSISTENT$[j]$ has been assigned *false*. If that were the case, then the constraint $u_{\mathbf{a_j}} = 1$ would be in $\mathcal{C}_{i+1}$, and (x^*, w^*, u^*) would not be a solution of $\mathcal{C}_{i+1}$. Therefore, at iteration $i+1$, at least one element in CONSISTENT that has value *true* is changed to *false*.

We will first argue that the algorithm always terminates. The second loop invariant implies in a straightforward manner that the algorithm terminates in at most p iterations.

Next, we show that for any $m \in [1..p]$, a potential answer $\mathbf{a_m}$ is a consistent answer if and only if CONSISTENT$[m] = 1$ at termination. In one direction, if $\mathbf{a_m}$ is a consistent answer, then Theorem 13 implies that for every solution $(\hat{x}, \hat{w}, \hat{u})$ to the constraints $\mathcal{C}$, it always holds that $\hat{u}_{\mathbf{a_m}} = 1$. Since for every $i \in [1..p]$, every optimal solution to P_i is also a solution to $\mathcal{C}$, we have that the algorithm will never execute line 14 for $j = m$. Hence, the value of CONSISTENT$[m]$ will always remain *true*.

In the other direction, if CONSISTENT$[m]$ is *true* after the algorithm has terminated, then assume towards a contradiction that $\mathbf{a_m}$ is not a consistent answer. If the algorithm terminates at the i-th iteration, then every solution to P_i is such that it assigns value 1 to every variable $u_{\mathbf{a_j}}$, for $j \in [1..p]$. So, the minimal value that the objective function of P_i can take is p. If $\mathbf{a_m}$ were not a consistent answer, we know from Theorem 13 that there must be a solution $(\hat{x}, \hat{w}, \hat{u})$ to $\mathcal{C}$ such that $\hat{u}_{\mathbf{a_m}} = 0$. We will reach a contradiction by showing that we can construct from $(\hat{x}, \hat{w}, \hat{u})$ a solution (x^*, w^*, u^*) to P_i such that $\sum_{j\in[1..p]} u^*_{\mathbf{a_j}} < p$. The vectors $\hat{x}, \hat{w}, \hat{u}$ are defined as follows:

$x^* = \hat{x}$; $u^*_{\mathbf{a_m}} = 0$; $u^*_{\mathbf{a_j}} = 1$ for $j \neq m$; for every pair of facts f and g, if they both appear in a minimal witness of $q[\mathbf{a}_m]$ in I, then let $w^*_{f,g} = 0$, otherwise, let $w^*_{f,g} = 1$. Because the equality constraints (a) and (a') in $\mathcal{C}_i$ and in $\mathcal{C}$ are the same, we have that x^* will satisfy the constraints (a) in $\mathcal{C}_i$. For any given pair of facts f and g that together appear in some minimal witness of q on I, there is one constraint (b) and one constraint (c). Two scenarios are possible for f and g: They may or may not appear in some minimal witness of $q[\mathbf{a}_m]$. In case f and g do appear in some minimal witness of $q[\mathbf{a_m}]$, then we know we have assigned $w^*_{f,g} = 0$. In $\mathcal{C}$ there is an inequality $x^{R_k}_f + x^{R_l}_g - u_{f,g} \leq 1$[2] and an inequality of the form $\cdots + u_{\mathbf{a}_m} + \cdots \geq |U| \cdot w_{f,g}$. Both these inequalities are satisfied by $(\hat{x}, \hat{w}, \hat{u})$. Since $\hat{u}_{\mathbf{a}_m} = 0$, then it must be that $\hat{w}_{f,g} = 0$. Subsequently, it must hold that $\hat{x}^{R_k}_f + \hat{x}^{R_l}_g \leq 1$. Since $\hat{x} = x^*$ then $x^{R_k}_f + x^{R_l}_g - w_{f,g} \leq 1$ is satisfied by x^*, w^*. Moreover, since $x^*_{\mathbf{a}_m} = 0$ and $w^*_{f,g} = 0$, it follows that $\cdots + u^*_{\mathbf{a}_m} + \cdots \geq |U| \cdot w^*_{f,g}$ holds true. The other scenario for f and g is that they never participate in a minimal witness to $q[\mathbf{a_m}]$ in I. In this case, $w^*_{f,g}$ is equal to 1. Hence, the inequality $x^{R_k}_f + x^{R_l}_g - w_{f,g} \leq 1$ is trivially satisfied by (x^*, w^*). Let us consider now the other constraint, which is, $\sum_{\mathbf{a} \in U} u_{\mathbf{a}} \geq |U| \cdot w_{f,g}$. In this inequality, the variable $u_{\mathbf{a}_m}$ does not appear. Since we have that $u_{\mathbf{a}_j} = 1$ for $j \neq m$, then the sum $\sum_{\mathbf{a} \in U} u_{\mathbf{a}}$ is equal to $|U|$. Since $w^*_{f,g} = 1$, then the inequality $\sum_{\mathbf{a} \in U} u_{\mathbf{a}} \geq |U| \cdot w_{f,g}$ is satisfied as an equality. We have so far argued that constraints (a), (a'), (b) and (c) in $\mathcal{C}_i$ are satisfied by (x^*, w^*, u^*). Finally, all equality constraint that may have been added to $\mathcal{C}$ during previous iterations are satisfied since $u^*_{\mathbf{a_j}} = 1$ for $j \neq m$, and the equality $u^*_{\mathbf{a_m}} = 1$ cannot be in $\mathcal{C}_i$. Now, it is

[2] R_k, R_l are the names of the relations in which, respectively, g and f exist.

clear that (x^*, w^*, u^*) is a solution to P_i such that $\sum_{j\in[1..p]} u^*_{\mathbf{a_j}} = p-1$. This contradicts the assumption that solutions to P_i yield minimal value p of the objective function. □

5.2.5 Implementation

We use the name EQUIP-AC to refer to the implementation of EQUIP that uses Theorem 13 for acyclic conjunctive queries. In Section 3.4 we described the architecture of EQUIP. In PHASE 1, the database pre-processing module would compute the minimal witnesses to the query and pass them to the module responsible for building the constraints of the BIP program. In Theorem 2, the constraints are generated from the key-equal groups and the minimal witnesses to the query. On the other hand, in Theorem 13, the constraints are generated from the key-equal groups and minimal witnesses to two-atom sub-queries, following an arbitrarily chosen join tree. Due to these differences, in order to use Algorithm ELIMINATEPOTENTIALANSWERS-AC for the acyclic and self-join free queries, we need to adapt PHASE 1 and PHASE 2. Observe that one can avoid modifying PHASE 1, by only modifying PHASE 2 to build the constraints in the fashion of Theorem 13 based on the minimal witnesses of the query. However, the number of all minimal witnesses to a conjunctive query is exponential in the size of the query. The advantage of the reduction for acyclic and self-join free queries lies precisely on the fact that we avoid reasoning based on all minimal witnesses of a given query q, and rather focus on minimal witnesses to two-atom sub-queries of q. Hence, computing all minimal witnesses in PHASE 1 goes against our purpose for developing an approach that is polynomial in the size of the instance and query.

Next, in Figure 5.2 we describe the Phase 1 that EQUIP-AC executes prior to running Algorithm EliminatePotentialAnswers-AC (see Appendix A.2 for a more detailed description of Phase 1). In Phase 1, we still pre-compute a portion of the consistent answers from the consistent part of the database. Then, based on a chosen join-tree τ, we generate a view WITNESSES_$\{R_i, R_j\}$ for every two atoms that are adjacent in τ. The view WITNESSES_$\{R_i, R_j\}$ holds all pairs of joining facts from R_i and R_j, such that the two joining facts are part of a minimal witness to q that gives rise to a potential answer that is not in ANS_FROM_CON. Finally, for every relation R_i, we create a view RELEVANT_R_i that holds (i) all R_i-facts that are part of a fact in at least one of the views WITNESSES_$\{R_i, R_j\}$, for $1 \leq j \leq l$, and (ii) all R_i-facts that are key-equal with some fact from the group of facts added to RELEVANT_R_i in (a).

```
PHASE 1: DATABASE PRE-PROCESSING (FOR ACYCLIC CONJUNCTIVE QUERIES)

Input:
    $\mathbf{R}$ : Schema with relation names $\{R_1, \cdots, R_i, \cdots, R_l\}$
    $q(\mathbf{z}) : -F_1, \cdots, F_j, \cdots, F_k$, where $F_j = R_{p_j}(\underline{\mathbf{x_j}}, \mathbf{y_j})$ for $j \in [1..k]$ and $1 \leq p_j \leq l$
      $I$ : database over $\mathbf{R}$

1. generate a join tree $\tau$ for $q$
2. for all $R_i, 1 \leq i \leq l$
    create view KEYS_R$_i$ that contains all tuples $\mathbf{d}$ s.t. there exists more than one fact of the form R$_\text{i}$($\mathbf{d}$,_) in $I$
3. create view ANS_FROM_CON that contains all tuples $\mathbf{t}$ s.t.
    - $\mathbf{t} \in q(I)$
    - there exists a minimal witness $S$ for $q(\mathbf{t})$, and s.t. no fact $R_i(\mathbf{d}, \_) \in S$ has its key-value $\mathbf{d}$ in KEYS_R$_i$
4. for all edges $(F_i, F_j)$ in $\tau, 1 \leq i \leq l, 1 \leq j \leq l$
    create view WITNESSES_$\{F_i, F_j\}$ with tuples $(\mathbf{t_i}, \mathbf{t_j}, \mathbf{a})$ s.t.
        - $R_{p_i}(\mathbf{t_i}) \in I$, $R_{p_j}(\mathbf{t_j}) \in I$
        - the set $S_{i,j} = \{R_{p_i}(\mathbf{t_{p_i}}), R_{p_j}(\mathbf{t_{p_j}})\}$ forms a minimal witness for $\{F_i, F_j\}$
        - there exists a minimal witness $S$ for $q[\mathbf{a}]$ s.t. $S_{i,j} \in S$ and $\mathbf{a}$ is a potential answer that is not in ANS_FROM_CON
5. for all $R_i, 1 \leq p_i \leq l$
(a) create view RELEVANT_R$_\text{i}$ that contains all tuples $\mathbf{t}$ s.t.
    - $R_i(t) \in I$, and
    - there exists an edge $(F_j, F_m)$ in $\tau$, where $p_j = i$, and
    - there exists a tuple $(t, \_, \_)$ in WITNESSES_$\{F_j, F_m\}$
(b) add to view RELEVANT_R$_\text{i}$ all tuples $\mathbf{t}'$ s.t.
    - $R_i(t') \in I$ is a fact, key-equal with a fact in RELEVANT_R$_\text{i}$
```

Figure 5.2: Description of PHASE 1 for acyclic conjunctive queries.

After PHASE 1 finishes executing, PHASE 2 runs to construct the binary integer program using the relevant facts in views RELEVANT_R_i, and witnesses in views WITNESSES_$\{R_i, R_j\}$. Subsequently, PHASE 3 evaluates the program to iteratively filter potential answers.

5.3 Experimental Evaluation

The main goal of our experiments has been to demonstrate the efficiency and scalability of EQUIP-AC in computing the consistent answers to acyclic conjunctive queries with several atoms, when compared to EQUIP. We evaluate both systems on a variety of acyclic conjunctive queries with varying number of atoms atoms, over synthetic databases generated with different parameters.

5.3.1 Experimental Setting

Our experiments have been carried out on an Amazon EC2 instance with 8GB of RAM, running Ubuntu v12.10, DB2 Express C v10.1, and IBM's ILOG CPLEX v12.6 for solving the binary integer programs. EQUIP-AC is implemented in Java.

In these experiments, we use synthetic inconsistent databases that are generated randomly, by varying several parameters in the generation of the data. We use a different benchmark than the one we used to evaluate EQUIP, mainly because the purpose of our experiments with EQUIP was to demonstrate its scalability w.r.t. the database size, and the benchmark queries of Table 3.3 have at most 3 atoms. We chose a set of conjunctive queries with up to 7 atoms, to compare the performance of EQUIP-AC vs EQUIP with respect to the size of the query. Next, we discuss the experimental queries and the generation of the inconsistent databases into more detail.

Benchmark queries Since the purpose of our experiments with EQUIP-AC is to investigate the performance of computing the consistent answers subject to the size of the queries, our experimental queries are chosen to have a large number of atoms, up to 7 atoms (see Table 5.1). All of the queries in Table 5.1 are such that CERTAINTY(q) and AC_CERTAINTY are coNP-complete. Queries in Table 5.1 are similar, in the sense that they involve joins on attributes that are not part of the key, but they have different number of atoms and free variables. There are a few reasons that have influenced our choice of the benchmark queries. One important reason is that the presence of nonkey-to-nonkey joins, quite often, gives rise to coNP-hardness [42]. Intuitively, in practice, nonkey-to-nonkey joins introduce higher complexity in reasoning about the consistent answers. To understand this, consider a query $q() : R(\underline{x}, y), S(\underline{y}, z)$, which involves a nonkey-to-key join. To reason about the consistent answers of q on any given database I, one need not consider the repairs of S in I, because every key-value of S appears in every repair. Another motivation for using queries that involve the same type of join is that, in order to analyze the behavior of the system with respect to the number of atoms, it is important that we preserve certain "patterns" in the queries as we increase the number of atoms. For instance, if you consider queries Q_1 to Q_5, they all have one free attribute, only ternary atoms, and only nonkey-to-nonkey joins. A similar observation can be made for queries Q_6 to Q_{10}.

Database generation Our synthetic inconsistent databases are generated in two steps: a) generate a consistent database, and b) from the consistent database, generate an inconsistent one by inserting tuples that violate some key constraints.

$Q_1(z) : -R_1(\underline{x_1}, y, z), R_2(\underline{x_2}, y, v), R_3(\underline{x_3}, y)$
$Q_2(z) : -R_1(\underline{x_1}, y, z), R_2(\underline{x_2}, y, v), R_3(\underline{x_3}, y), R_4(\underline{x_4}, y)$
$Q_3(z) : -R_1(\underline{x_1}, y, z), R_2(\underline{x_2}, y, v), R_3(\underline{x_3}, y), R_4(\underline{x_4}, y), R_5(\underline{x_5}, y),$
$Q_4(z) : -R_1(\underline{x_1}, y, z), R_2(\underline{x_2}, y, v), R_3(\underline{x_3}, y), R_4(\underline{x_4}, y), R_5(\underline{x_5}, y), R_6(\underline{x_6}, y)$
$Q_5(z) : -R_1(\underline{x_1}, y, z), R_2(\underline{x_2}, y, v), R_3(\underline{x_3}, y), R_4(\underline{x_4}, y), R_5(\underline{x_5}, y), R_6(\underline{x_6}, y), R_7(\underline{x_6}, y)$
$Q_6(z, v) : -R_1(\underline{x_1}, y, z), R_2(\underline{x_2}, y, v), R_3(\underline{x_3}, y)$
$Q_7(z, v) : -R_1(\underline{x_1}, y, z), R_2(\underline{x_2}, y, v), R_3(\underline{x_3}, y), R_4(\underline{x_4}, y)$
$Q_8(z, v) : -R_1(\underline{x_1}, y, z), R_2(\underline{x_2}, y, v), R_3(\underline{x_3}, y), R_4(\underline{x_4}, y), R_5(\underline{x_5}, y)$
$Q_9(z, v) : -R_1(\underline{x_1}, y, z), R_2(\underline{x_2}, y, v), R_3(\underline{x_3}, y), R_4(\underline{x_4}, y), R_5(\underline{x_5}, y), R_6(\underline{x_6}, y)$
$Q_{10}(z, v) : -R_1(\underline{x_1}, y, z), R_2(\underline{x_2}, y, v), R_3(\underline{x_3}, y), R_4(\underline{x_4}, y), R_5(\underline{x_5}, y), R_6(\underline{x_6}, y), R_7(\underline{x_6}, y)$

Table 5.1: Benchmark queries used to evaluate EQUIP-AC against EQUIP

a) Generation of the consistent databases The consistent databases are generated in a similar way as the ones we used for EQUIP. Apart from the parameter *r_size*, which determines the number of tuples per relation, we introduce two additional parameters. One parameter, α, determines for each relation R_i, the percentage of R_i-facts that participate in a minimal witness of the query. In our experiments, we always fix this parameter to 10%. The other parameter, which we call *join_multiplicity*, determines for each R_i, R_j that join in a query, the number of R_i-facts that join with a single R_j-fact, and vice-versa. For example, if *join_multiplicity=5*, and R_i, R_j are two atoms that share a variable in any of the queries in Table 5.1, then 10% of the R_i-facts join with some R_j-fact, and each such R_i-fact joins with five R_j-facts. Also, 10% of the R_j-facts join with some R_i-fact, and each such R_i-fact joins with five R_j-facts. The role of the *join_multiplicity* parameter is to control the number of minimal witnesses to the queries of Table 5.1, where higher *join_multiplicity* means more minimal witnesses to the query. The third attribute in all of the ternary relations, which is sometimes projected out and never used as a join attribute in the queries of Table 5.1, takes values from a uniform distribution in the range [1, *r_size*/10]. In each relation, there are approximately *r_size*/10 distinct values in the third attribute, each appearing approximately 10 times.

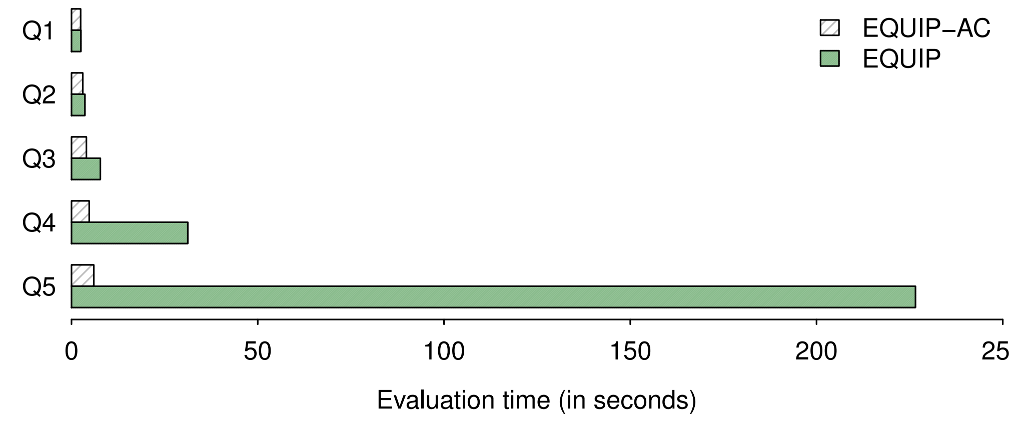
b) Generation of the inconsistent databases The generation of the inconsistent databases is done exactly as we did for EQUIP, in Section 3.5.

Experimental Parameters Unless otherwise stated, our experiments use databases with 10% of the tuples involved in conflicts, and each key-equal group containing two facts. In our evaluations, we measure the time for computing the consistent answers as the time between receiving the query and returning the last consistent answer to the query, i.e., every processing step is accounted for in the total evaluation time. On each database, we run each query five times and take the average. Again, we set the GAP parameter to 0.1, and parameter EpInt to 0.

5.3.2 Experimental Results

Next, we present the results of our experiments with the queries of Table 5.1. Initially, we focus on comparing the performance of EQUIP-AC against EQUIP.

Comparing EQUIP-AC against EQUIP Initially, we compare EQUIP-AC against EQUIP on a database with 100K tuples per relation, and parameter *join_multiplicity* set to 5. Figure 5.3 shows the result of this comparison on queries Q_1 to Q_5. As expected, Figure 5.3 clearly shows that as we increase the number of atoms, EQUIP-AC significantly outperforms EQUIP. In fact, EQUIP fails to evaluate query Q_{10} due to the BIP solver running out of memory. The inconsistent dataset is generated in such a way that the number of potential answers is about the same for all queries Q_1 to Q_5 (also, for Q_6 to Q_{10}). Therefore, the increasing evaluation times from Q_1 to Q_5 (similarly, from Q_6 to Q_{10}) are due to the increasing number of atoms only.



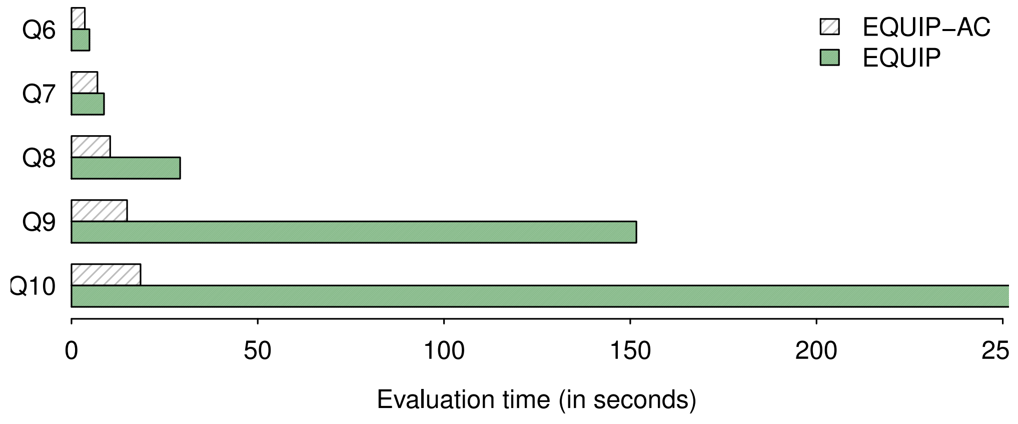


Figure 5.3: Performance of EQUIP-AC vs. EQUIP on a database with 100K tuples/relation, 10% of the tuples involved in violations, and join_multiplicity equal to 5. Top: Performance on queries Q_1 to Q_5; Bottom: Performance on queries Q_6 to Q_{10}.

To better explain Figure 5.3, we break down the total evaluation times into three components, corresponding to each of the three phases of the computation of consistent answers (see Figure 5.4). Because of the high evaluation times of Q_5 and Q_{10} by EQUIP, the large scale does not allow observing the break-down of the running times for EQUIP-AC. Therefore, in Figure 5.4 we show the evaluation up to 30 seconds for Q_1 to Q_5, and up to 70 seconds for Q_6-Q_{10}. This figure reveals that, as we increase the number of atoms, EQUIP-AC becomes faster than EQUIP on each of the three phases. This is explained as follows: In PHASE 1, the pre-computation of the consistent answers from the clean part of the database takes the exact same times in both systems, as they both run the same SQL query to obtain these answers. The rest of PHASE 1 is dominated by the computation of the minimal witnesses. EQUIP materializes all minimal witnesses to the query. On the other hand, EQUIP-AC materializes minimal witnesses to pairs of joining relations. As expected, when the number of atoms is increased, the number of minimal witnesses that EQUIP materializes in PHASE 1 becomes much larger than the minimal witnesses generated by EQUIP-AC. As a consequence, PHASE1 of EQUIP becomes significantly more expensive that PHASE 1 of EQUIP-AC. Obviously, the number of generated constraints depends on the number of minimal witnesses. Thus, EQUIP generates a larger number of constraints than EQUIP-AC, which explains why aslo PHASE 2 & 3 become more expensive compared to EQUIP-AC. We also observe that, in both systems, for the less expensive queries, Q_1 to Q_5, the overall running times are dominated by PHASE 1. On the other hand, for queries Q_6 to Q_{10}, the time that goes to PHASE 2 & 3 is more than 50% of the overall running time. Queries Q_6 to Q_{10} have a much larger number of additional potential answers, and therefore, the BIP programs for these queries are larger.

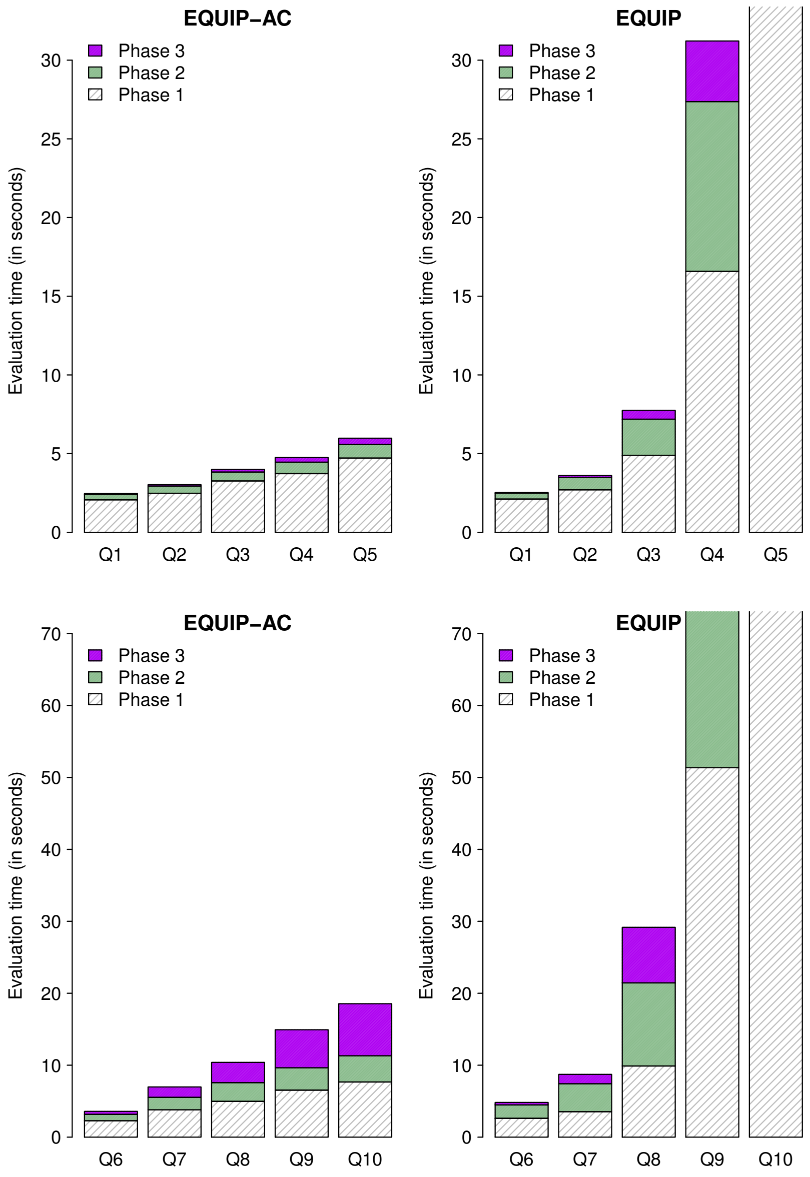


Figure 5.4: Evaluation times split by phase, of queries Q_1 to Q_5 (top) and Q_6 to Q_{10} (bottom), over a database with 100K tuples/relation, and join_multiplicity equal to 5.

Sizes of BIP programs Next, we look closer at the sizes of the BIP programs generated by both systems, which are shown in Table 5.2. The number of constraints generated from the key-equal groups is the same for both systems. Therefore, the relative difference in the number of constraints is due to the constraints generated based on the minimal witnesses. Table 5.2 reveals that the queries on which EQUIP-AC significantly outperforms EQUIP, are the ones for which EQUIP-AC generates a much smaller number of constraints compared to EQUIP (see $Q_4, Q_5, Q_8, Q_9, Q_{10}$). While the number of constraints generated by EQUIP-AC is always between 0.5 and 2.5 times as large as the number of variables, EQUIP generates programs with up to 700 times more constraints than variables.

	EQUIP-AC		EQUIP	
	Constraints	Variables	Constraints	Variables
Q_1	9K	4K	9K	3K
Q_2	16K	7K	37K	4K
Q_3	23K	10K	187K	5K
Q_4	31K	13K	990K	7K
Q_5	39K	17K	5500K	8K
Q_6	35K	24K	40K	16K
Q_7	81K	54K	176K	20K
Q_8	127K	84K	900K	25K
Q_9	172K	113K	4600M	30K
Q_{10}	217K	141K	24000M	34K

Table 5.2: Sizes of BIP programs generated in the experiment of Figure 5.3.

Varying parameter join_multiplicity In Figure 5.5 we report the evaluation times of both systems on queries Q_4, Q_5, Q_9 and Q_{10}, over 5 databases with 100K tuples per relation, varying *join_multiplicity* to take values from 1 to 5. Naturally, as we increase *join_multiplicity*, the number of minimal witnesses grows. As a consequence, the gap in performance between the two systems also increases.

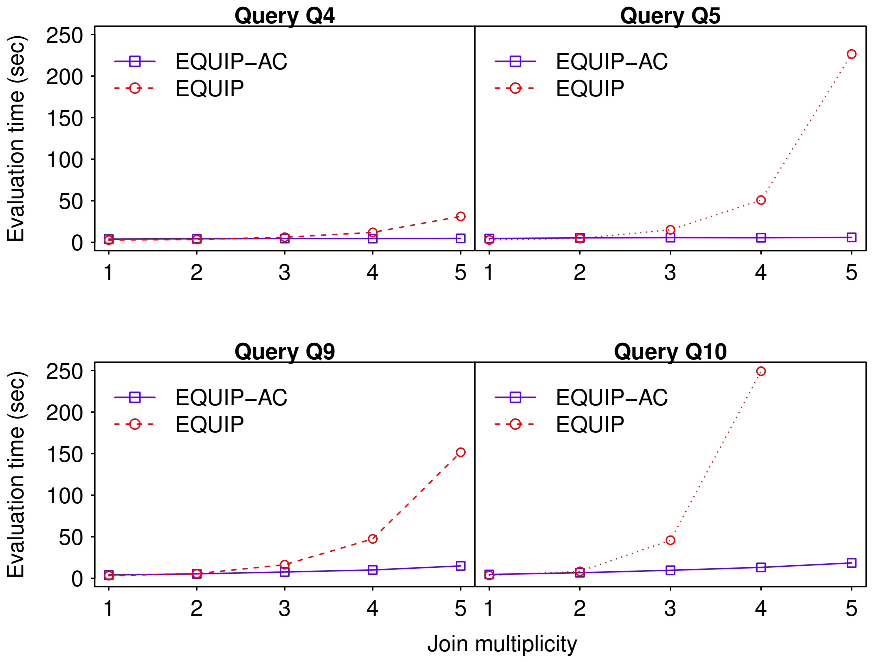


Figure 5.5: Performance of EQUIP-AC and EQUIP over a database with 100K tuples/relation, 10% of the tuples involved in violations, and varying values of join_multiplicity from 1 to 5. Top: Evaluation of query Q_5; Bottom: Evaluation of query Q_{10}

Varying database size We analyze the behavior of EQUIP-AC with respect to the database size. We generate ten inconsistent databases containing 100,000 up to 1 million tuples per relation, each database having 10% of the tuples involved in conflicts, and parameter *join_multiplicity* set to value 5. Figure 5.6 shows the evaluation times of EQUIP-AC on these 10 databases. As the figure shows, the increase of the evaluation times is quite gradual, becoming about 10x higher as we go from 100K to 1000K tuples per relation.

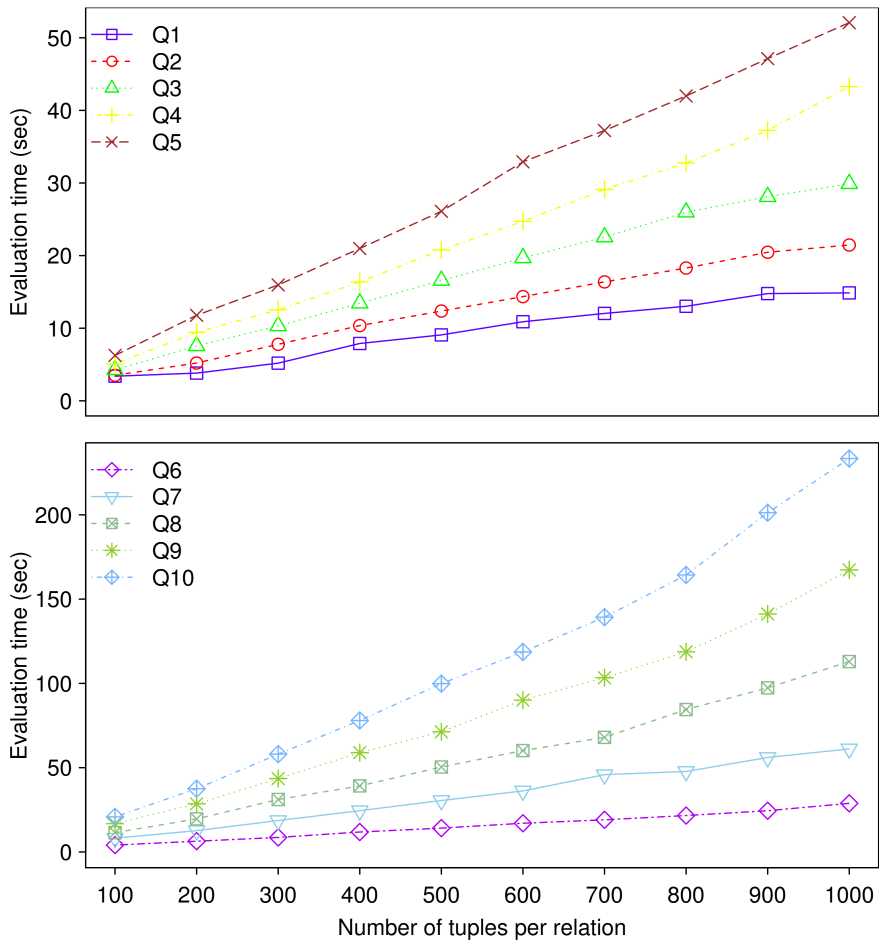


Figure 5.6: Performance of EQUIP-AC on databases with 100K to 1 million tuples per relation, and join_multiplicity equal to 5.

Our experiments in this section have demonstrated that EQUIP-AC significantly outperforms EQUIP on queries with three or more atoms, even over databases containing only 100K tuples per relation. As expected, our experiments show that as we increase the number of atoms, EQUIP generates a significantly larger number of minimal witnesses and BIP constraints compared to EQUIP-AC. In addition, EQUIP-AC scales well over large databases containing up to 1 million tuples per relation (7 million in total).

Chapter 6

Consistent Query Answering with Linear Programming

In EQUIP we follow a generic approach for consistent query answering, that uses the same strategy to model CERTAINTY(q) with binary Integer Programming, despite of the actual complexity of CERTAINTY(q). Later, in Section 4.4, in Lemma 7 we gave a sufficient condition for CERTAINTY(q) to be tractable. We proved this result by giving an algorithm that reduces CERTAINTY(q) to the problem of finding a maximum independent set in a claw-free graph. While a substantial amount of work has been dedicated to building and optimizing systems for Binary Integer Programing, we are not aware of any standard efficient tools for solving the maximum independent set problem. Minty's algorithm [52] is quite involved and an implementation of that algorithm would require complex and expensive graph data structures. In this chapter, we will show that alternatively, we can compute CERTAINTY(q) for the class of queries of Lemma 7 using Linear Optimization. While solving integer programs is well-known to be an NP-complete problem, evaluation of linear programs can be efficiently done in polynomial time. Even though many real optimization problems can be naturally modeled as integer programs, it may be the case that it is enough to solve the LP relaxation of the integer program, i.e., the program obtained by dropping the constraint that the variables take only integer values. There are several well known techniques for proving that the optimal solution of the LP relaxation of an integer program is always integral, i.e., the assignments to the variables are integer values. We explore this direction to provide an alternative proof of tractability for the class of queries in Lemma 7. More specifically, for the same class of queries we give a new reduction from CERTAINTY(q) to Binary Integer Programming. Then we show that this new reduction generates programs whose LP relaxations has always optimal solutions consisting of all integer assignments.

Theorem 15 *Let q be a self-join free, acyclic boolean conjunctive query with atoms $R_1, ..., R_k$. Let τ be a join tree of q such that for every pair R_l, R_m that form an edge in τ, it holds that $key(R_l) \subseteq vars(R_m)$ and $key(R_m) \subseteq vars(R_l)$. Let τ be a join tree of q. Given a database instance I over the same schema as q, we construct in polynomial time the following system of linear equalities and inequalities: System (6):*

Variables:

$x_f^{R_i} \in \{0,1\}$ *for every R_j-fact f, and R_i adjacent with R_j in τ*

$u_{\mathbf{a}} \in \{0,1\}$ *for every $\mathbf{a} \in q(I)$*

<u>Constraints</u>:

(a) $\displaystyle\sum_{f \in K;\ R_i \in N_\tau(R_j)} x_f^{R_i} = 1,$ *for every relation R_j in I and every key-equal group K of R_j.*

(b) $\displaystyle\sum_{f \in X} x_f^{R_i} + \sum_{g \in X} x_g^{R_j} \leq 1,$ *for every edge (R_j, R_i) in τ, and for X a minimal set of facts s.t. there exists $\{f, g\} \subseteq X$, where $\{f, g\}$ is a minimal witness of $q_{\{R_j, R_i\}}$, and every minimal witness of the form $\{f, g'\}$ or $\{f', g\}$ is in X.*

Then the following statements are equivalent:

- *There is a repair r of I such that q is false on r.*
- *System (6) has a solution.*

Proof. To understand the construction of System (6), observe that it uses the same variables as System (3) in Theorem 11, the constraints (a) are the same, but the constraints (b) are generated in a different fashion. The set X, based one which the constraints (b) are generated, is computed as follows: if $\{f, g\}$ is a minimal witness to $q_{\{R_j, R_i\}}$ then we put the facts f and g in X; we add to X every other R_i-fact g' such that $\{f, g'\}$ is also a minimal witness of $q_{\{R_j, R_i\}}$; we add to X every other R_j-fact f' such that $\{f', g\}$ is also a minimal witness of $q_{\{R_j, R_i\}}$. By construction, the set X is such that every R_i-fact in X can be combined with every R_j-fact in X to obtain a minimal witness of $q_{\{R_j, R_i\}}$.

To prove that a repair r of I such that q is false on r exists if and only if System (6) has a solution, we will use Theorem 11. More specifically, we will prove that every solution to System (6) is a solution to System (3).

($\Rightarrow$) Assume that System (6) has a solution x^*. The constraints (a) in System

(3) are trivially satisfied by this solution. Moreover, to see that constraints (b) are also satisfied by x^*, observe that for every minimal witness $\{f, g\}$ to $q_{\{R_j,R_i\}}$, where (R_j, R_i) is an edge in τ, there is a unique set X that contains f and g. Since x^* satisfies the constraints (b) in System (6) ,then $x_f^{R_i} + x_g^{R_j} \leq 1$.

($\Leftarrow$) In this opposite direction, assume that x^* is a solution of System (3). The constraints (a) in System (6) are trivially satisfied by x^*. To argue that constraints (b) in System (6) are also satisfied by x^* we will make use of the syntactic condition of q. Let X be a set of facts computed as described in the formulation of System (6). Let $\{f, g\}$ and $\{f', g\}$ be two minimal witnesses of $q_{\{R_j,R_i\}}$ that exist in X. Because $key(R_j) \subseteq vars(R_i)$, it follows that f and f' are key-equal. Then, all R_j-facts in X are key-equal. Subsequently, the inequality $\sum\limits_{f \in X} x_f^{R_i} \leq 1$ is satisfied by x^*. Similarly, we can derive that because $key(R_i) \subseteq vars(R_j)$, the inequality $\sum\limits_{g \in X} x_g^{R_j} \leq 1$ is also satisfied by x^*. Since x^* is a solution to System (3), then, for every R_j-fact f in X and every R_i-fact g in X, the inequality $x_f^{R_i} + x_g^{R_j} \leq 1$ is satisfied by x^*. Hence, it cannot happen that both inequalities $\sum\limits_{f \in X} x_f^{R_i} \leq 1$ and $\sum\limits_{g \in X} x_g^{R_j} \leq 1$ are satisfied by x^*. It is obvious now that $\sum\limits_{f \in X} x_f^{R_i} + \sum\limits_{g \in X} x_g^{R_j} \leq 1$. $\square$

Next, we will prove an important result about System (6). We will show that solving System (6) without restricting the variables to take binary values, still yields integral optimal solutions. More specifically, we will show that it is enough to solve the LP relaxation of System (6) to obtain an integral solution. Since it is well known that LP is in polynomial time, we will thus be giving an alternative proof of Lemma 7.

First, we will introduce some preliminary notions. Linear Programs (LP) are optimization problems of the form $max\ \{\mathbf{c}^T\mathbf{x} \mid A\mathbf{x} \leq \mathbf{b}; x \in \mathbb{Q}\}$, where $\mathbf{b}$ and $\mathbf{c}$ are vectors of integer coefficients, $\mathbf{b}^T$ and $\mathbf{c}^T$ are the transpose of, respectively, $\mathbf{b}$ and $\mathbf{c}$, A is a matrix of integer coefficients and $\mathbf{x}$ is a vector of variables, ranging over the set Q of rational numbers. We will make use of several important notions and results from polyhedral theory [54]. If $max\ \{c^Tx \mid Ax \leq b; x \in Z^*\}$ is an integer linear program, then the LP relaxation of this program is the linear program taken by dropping the constraints that the variables can take only integer values, i.e., the linear program $max\ \{c^Tx \mid Ax \leq b\}$. Clearly, $max\ \{c^Tx \mid Ax \leq b; x \in Z^*\} \leq max\ \{c^Tx \mid Ax \leq b\}$. The system of linear constraints $Ax \leq b$ defines the *polyhedron* $P = \{x | Ax \leq b\}$. A polyhedron is called *integral* if each of its vertices is an integer vector. Because the optimal value of the objective function of a linear program is attained at some vertex of the polyhedron, it is obvious that if P is integral then $max\ \{c^Tx \mid Ax \leq b\}$ has all integer optimal solutions. Thus, $max\ \{c^Tx \mid Ax = b; x \in Z^*\} = max\ \{c^Tx \mid Ax \leq b\}$. A widely used technique for proving that a polyhedron is integer is by proving that the matrix of constraints is *totally unimodular.*

Definition 11 ([54]) *A matrix A is totally unimodular if each square submatrix of A has determinant equal to 0,+1, or -1.*

Theorem 16 ([54]) *Let A be a totally unimodular $m \times n$ matrix and let $b \in \mathbb{Z}^m$. Then each vertex of the polyhedron $P = \{x|\ Ax \leq b\}$ is an integer vector.*

Our strategy for proving that for the class of queries that satisfy the conditions of Theorem 15 there is a reduction to Linear Programming is to show that the matrix of the coefficients of System (6) is totally unimodular. A few characterizations for totally unimodularity have been given in the literature. One such characterization is given by Heller and Tompkins [39, 40], and Hoffman and Gale [41].

Theorem 17 *Let A be a $\{0, +1, -1\}$-matrix. Matrix A is totally unimodular if:*

- *there are at most two nonzero elements in each column.*
- *the rows of A can be split into two parts B and C such that for every column, if there are two nonzero elements in the column that have the same sign they are in different parts, otherwise, if they have oposite signs they are in the same part.*

Theorem 18 *Let q be a self-join free, acyclic boolean conjunctive query with atoms $R_1, ..., R_k$. Let τ be a join tree of q such that for every pair R_l, R_m that form an edge in τ, it holds that $key(R_l) \subseteq vars(R_m)$ and $key(R_m) \subseteq vars(R_l)$. Let I be a database instance. Then, the matrix of coefficients of System (6) is totally unimodular.*

Proof. Let A be the matrix of coefficients of the constraints in System (6), where each constraint is a linear combination of a row of A with the vector of variables $\mathbf{x}$. We show that A is totally unimodular. Let B be the matrix of coefficients of constraints (a). Let C be the matrix of coefficients of constraints (b). Obviously, $\{B, C\}$ is a partition of the rows of A. Because every database fact appears in exactly one key-equal group, then every variable $x_f^{R_i}$ appears in exactly one constraint in (a). Hence, for every variable

$x_f^{R_i}$, in the corresponding column in A, there is a coefficient 1 which appears in a row that belongs to B, and there is no other coefficient 1 appearing in another row of B. From the construction of the constraints (b) it is easy to see that there is exactly one inequality that mentions $x_f^{R_i}$. Hence, there is a single 1 which appears in a row of C. Now, it is obvious that for every column in A there are no more than two nonzero elements, they are both 1s, each appearing in a different part B or C. □

Finally, we get the following corollary:

Corollary 5 *Let q be an acyclic boolean conjunctive query without self-joins, and with atoms $R_1, R_2, \cdots, R_k$. If q has a join tree τ such that for every pair of atoms R_i, R_j connected with an edge in τ, it holds that $key(R_i) \subseteq vars(R_j)$ and $key(R_j) \subseteq vars(R_i)$, then the LP relaxation of System (6) has $\{0, 1\}$ solutions.*

The LP-based technique we described in this chapter, for computing the consistent answers to a sub-class of acyclic and self-join free conjunctive queries, only applies to the case when the query is boolean. Of course, it is possible to use this technique to check every potential answers of a non-boolean query, if it is a consistent answer or not. However, as we argue in Section 3.2.2, we do not expect this technique to be efficient in practice for the non-boolean queries. In the future, it would be interesting to come up with a strategy for modeling CERTAINTY(q) of non-boolean q (for our class of tractable queries), with a single linear program. However, the technique described in this chapter is interesting as yet another polynomial algorithm for consistent query answering.

Chapter 7

Conclusions and Future Work

We developed EQUIP, a new system for computing the consistent answers of conjunctive queries under primary key constraints. The main technique behind EQUIP is the reduction of the problem of computing the consistent answers of conjunctive queries under primary key constraints to binary integer programming, and the systematic use of efficient integer programming solvers, such as CPLEX. Our extensive experimental evaluation suggests that EQUIP is promising in that it consistently exhibits good performance even on relatively large databases. EQUIP is also, by far, the best available system for evaluating the consistent query answers of queries that are coNP-hard and of queries that are in PTIME but not first-order rewritable. but not in the class C_{forest}. For C_{forest} queries, ConQuer demonstrates superior performance, but is of limited applicability.

We also studied the complexity of consistent query answering for the class of acyclic and self-join free conjunctive queries under primary key constraints. We conjectured that a dichotomy on the complexity of CERTAINTY(q) exists. We proved sufficient conditions for tractability and intractability. If our conjecture is correct, then the sufficient condition for intractability of CERTAINTY(q) provides a proof for the intractability side of the general dichotomy. Since we only have a sufficient condition for tractability of CERTAINTY(q), a gap remains in the dichotomy, which we believe to contain only queries that are in P but not first-order rewritable. As future work, it remains to fill this gap by extending our existing algorithms, or exploring entirely novel algorithms to evaluate the consistent answers in polynomial time. We have already presented two alternative polynomial time algorithms, where one establishes a connection to the Maximum Independent Set problem for claw-free graphs, and the other consists in giving a reduction to Linear Programming. It would be interesting to see if any of the two approaches can be extended to a larger class.

Finally, we investigated the combined complexity of consistent query answering for the class of conjunctive queries and primary key constraints. We showed that in general, for this class, computing the consistent answers can be Π_2^P-complete in combined complexity. However, for the more restricted class of acyclic conjunctive queries, we argue that the problem is coNP-complete. This result motivated us to find an explicit polynomial reduction to BIP, where the BIP programs have size polynomial in the size of the database instance and the query. We implemented this specialized reduction in EQUIP-AC. Our experimental evaluations with EQUIP-AC and EQUIP on queries with up to 7 atoms, demonstrate that, as expected, EQUIP-AC scales significantly better than EQUIP w.r.t. the number of atoms in the query.

Our results in this thesis suggest that an “optimal” system for consistent query answering is likely to rely not on a single technique, but, rather on a portfolio of techniques. Such a system will first attempt to determine the computational complexity of computing the consistent answers of the query at hand and then, based on this information, will invoke the most appropriate technique for computing the consistent answers. Developing such a system, however, will require further exploration and deeper understanding of the boundary between tractability and intractability of consistent query answering. In addition, for future work, we also plan to investigate extensions of our BIP-based technique to unions of conjunctive queries and broader classes of constraints, such as functional dependencies and foreign key constraints.

Bibliography

[1] http://www-01.ibm.com/software/integration/optimization/cplex-optimizer.

[2] Foto N. Afrati and Phokion G. Kolaitis. Repair checking in inconsistent databases: algorithms and complexity. In *ICDT*, pages 31–41, 2009.

[3] M. Arenas, L. Bertossi, J. Chomicki, X. He, V. Raghavan, and J. Spinrad. Scalar aggregation in inconsistent databases. *Theoretical Computer Science. 296(3), 2003.*

[4] Marcelo Arenas, Leopoldo E. Bertossi, and Jan Chomicki. Consistent query answers in inconsistent databases. In *PODS*, pages 68–79, 1999.

[5] Marcelo Arenas, Leopoldo E. Bertossi, and Jan Chomicki. Answer sets for consistent query answering in inconsistent databases. *TPLP*, 3(4-5):393–424, 2003.

[6] Marcelo Arenas, Leopoldo E. Bertossi, Jan Chomicki, Xin He, Vijay Raghavan, and Jeremy Spinrad. Scalar aggregation in inconsistent databases. *TCS*, 296(3):405–434, 2003.

[7] Pablo Barceló and Leopoldo E. Bertossi. Logic programs for querying inconsistent databases. In *PADL*, pages 208–222, 2003.

[8] Catriel Beeri, Ronald Fagin, David Maier, and Mihalis Yannakakis. On the desirability of acyclic database schemes. *J. ACM*, 30(3):479–513, 1983.

[9] Catriel Beeri and Raghu Ramakrishnan. On the power of magic. In *JLP*, pages 269–283, 1987.

[10] TPCH Benchmark. http://www.tpc.org/tpch/.

[11] Leopoldo Bertossi. *Database Repairing and Consistent Query Answering.* Morgan and Claypool Publishers, 2011.

[12] Andrea Calì, Domenico Lembo, and Riccardo Rosati. On the decidability and complexity of query answering over inconsistent and incomplete databases. In *PODS*, pages 260–271, 2003.

[13] Mónica Caniupán and Leopoldo E. Bertossi. The consistency extractor system: Querying inconsistent databases using answer set programs. In *SUM*, pages 74–88, 2007.

[14] Mónica Caniupán and Leopoldo E. Bertossi. The consistency extractor system: Answer set programs for consistent query answering in databases. *DKE*, 69(6):545–572, 2010.

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

[16] Jan Chomicki, Jerzy Marcinkowski, and Slawomir Staworko. Computing consistent query answers using conflict hypergraphs. In *CIKM*, pages 417–426, 2004.

[17] Jan Chomicki, Jerzy Marcinkowski, and Slawomir Staworko. Hippo: A system for computing consistent answers to a class of sql queries. In *EDBT*, pages 841–844, 2004.

[18] Tamraparni Dasu and Theodore Johnson. *Exploratory Data Mining and Data Cleaning*. John Wiley, 2003.

[19] Alexandre Decan, Fabian Pijcke, and Jef Wijsen. Certain conjunctive query answering in sql. In *SUM*, pages 154–167, 2012.

[20] Thomas Eiter, Wolfgang Faber, Christoph Koch, Nicola Leone, and Gerald Pfeifer. Dlv - a system for declarative problem solving. *CoRR*, cs.AI/0003036, 2000.

[21] Thomas Eiter, Michael Fink, Gianluigi Greco, and Domenico Lembo. Efficient evaluation of logic programs for querying data integration systems. In *ICLP*, pages 163–177, 2003.

[22] Ahmed K. Elmagarmid, Panagiotis G. Ipeirotis, and Vassilios S. Verykios. Duplicate record detection: A survey. *IEEE TKDE*, 19(1):1–16, 2007.

[23] Wenfei Fan and Floris Geerts. *Foundations of Data Quality Management*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2012.

[24] Sergio Flesca, Filippo Furfaro, and Francesco Parisi. Consistent query answers on numerical databases under aggregate constraints. In *DBPL*, pages 279–294, 2005.

[25] Sergio Flesca, Filippo Furfaro, and Francesco Parisi. Consistent answers to Boolean aggregate queries under aggregate constraints. In *DEXA (2)*, pages 285–299, 2010.

[26] Sergio Flesca, Filippo Furfaro, and Francesco Parisi. Querying and repairing inconsistent numerical databases. *ACM TODS*, 35(2), 2010.

[27] Sergio Flesca, Filippo Furfaro, and Francesco Parisi. Range-consistent answers of aggregate queries under aggregate constraints. In *SUM*, pages 163–176, 2010.

[28] Gaëlle Fontaine. Why is it hard to obtain a dichotomy for consistent query answering? In *LICS*, pages 550–559, 2013.

[29] Ariel Fuxman, Elham Fazli, and Renée J. Miller. ConQuer: Efficient management of inconsistent databases. In *SIGMOD*, pages 155–166, 2005.

[30] Ariel Fuxman, Diego Fuxman, and Renée J. Miller. Conquer: A system for efficient querying over inconsistent databases. In *VLDB*, pages 1354–1357, 2005.

[31] Ariel Fuxman and Renée J. Miller. First-order query rewriting for inconsistent databases. In *ICDT*, pages 337–351, 2005.

[32] Ariel Fuxman and Renée J. Miller. First-order query rewriting for inconsistent databases. *JCSS*, 73(4):610–635, 2007.

[33] Venkatesh Ganti and Anish Das Sarma. *Data Cleaning: A Practical Perspective.* Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2013.

[34] M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman, 1979.

[35] Georg Gottlob, Nicola Leone, and Francesco Scarcello. Hypertree decompositions and tractable queries. *J. Comput. Syst. Sci.*, 64(3):579–627, 2002.

[36] Gianluigi Greco, Sergio Greco, and Ester Zumpano. A logic programming approach to the integration, repairing and querying of inconsistent databases. In *ICLP*, pages 348–364, 2001.

[37] Gianluigi Greco, Sergio Greco, and Ester Zumpano. A logical framework for querying and repairing inconsistent databases. *IEEE TKDE*, 15(6):1389–1408, 2003.

[38] Sergio Greco, Fabian Pijcke, and Jef Wijsen. Certain query answering in partially consistent databases. *PVLDB*, 7(5):353–364, 2014.

[39] Isidore Heller. *On linear systems with integral valued solutions.* Pacific Journal of Mathematics, 1957.

[40] Isidore Heller and C. B. Tompkins. An extension of a theorem of Dantzig's. *Annals of Mathematical Studies*, (38):247–252, 1956.

[41] Alan J. Hoffman and David Gale. Appendix [to the paper "an extension of a theorem of dantzig's." of heller and tompkins]. *Annals of Mathematical Studies*, (38):252–254, 1956.

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

[43] Phokion G. Kolaitis, Enela Pema, and Wang-Chiew Tan. On the tractability and

intractability of consistent conjunctive query answering. In *Proceedings of the 2011 Joint EDBT/ICDT Ph.D. Workshop*, PhD '11, pages 38–44, 2011.

[44] Phokion G. Kolaitis, Enela Pema, and Wang-Chiew Tan. Efficient querying of inconsistent databases with binary integer programming. *PVLDB*, 6(6):397–408, 2013.

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

[46] Paraschos Koutris and Dan Suciu. A dichotomy on the complexity of consistent query answering for atoms with simple keys. In *ICDT*, pages 165–176, 2014.

[47] Richard E. Ladner. On the structure of polynomial time reducibility. *J. ACM*, 22(1):155–171, 1975.

[48] Nicola Leone, Thomas Eiter, Wolfgang Faber, Michael Fink, Georg Gottlob, and Gianluigi Greco. Boosting information integration: The INFOMIX system. In *SEBD*, pages 55–66, 2005.

[49] Nicola Leone, Marco Manna, Francesco Ricca, and Giorgio Terracina. Private communication.

[50] David Maier. *The Theory of Relational Databases.* Computer Science Press, 1983.

[51] Dany Maslowski and Jef Wijsen. On counting database repairs. In *LID*, pages 15–22, 2011.

[52] George J. Minty. On maximal independent sets of vertices in claw-free graphs. *J. Comb. Theory, Ser. B*, 28(3):284–304, 1980.

[53] Davy Van Nieuwenborgh and Dirk Vermeir. Preferred answer sets for ordered logic programs. *TPLP*, 6(1-2):107–167, 2006.

[54] Alexander Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency.* 2003.

[55] Slawomir Staworko and Jan Chomicki. Consistent query answers in the presence of universal constraints. *Inf. Syst.*, 35(1):1–22, 2010.

[56] Dan Suciu, Dan Olteanu, Christopher Ré, and Christoph Koch. *Probabilistic Databases.* Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2011.

[57] Jef Wijsen. Consistent query answering under primary keys: a characterization of tractable queries. In *ICDT*, pages 42–52, 2009.

[58] Jef Wijsen. On the consistent rewriting of conjunctive queries under primary key constraints. *Inf. Syst.*, 34(7):578–601, 2009.

[59] Jef Wijsen. On the first-order expressibility of computing certain answers to conjunctive queries over uncertain databases. In *PODS*, pages 179–190, 2010.

[60] Jef Wijsen. A remark on the complexity of consistent conjunctive query answering under primary key violations. *IPL*, 110(21):950–955, 2010.

[61] Jef Wijsen. Certain conjunctive query answering in first-order logic. *ACM Trans. Database Syst.*, 37(2):9, 2012.

[62] Jef Wijsen. Charting the tractability frontier of certain conjunctive query answering. In *PODS*, pages 189–200, 2013.

[63] Jef Wijsen. A survey of the data complexity of consistent query answering under key constraints. In Christoph Beierle and Carlo Meghini, editors, *Foundations of Information and Knowledge Systems*, volume 8367 of *Lecture Notes in Computer Science*, pages 62–78. Springer International Publishing, 2014.

Appendix A

Detailed Algorithms for Phase 1 of EQUIP and EQUIP-AC

A.1 Description of Phase 1 of EQUIP

Figure A.1 gives a more detailed description of the implementation of Phase 1 in EQUIP. Steps 1-4 show the actual SQL queries that are generated in each of the steps 1-4 of Figure 3.3. Note that in step 4, each of the views RELEVANT_R$_i$ has an attribute ID_R_i in the view definition. This attribute is a unique identifier for every tuple in RELEVANT_R$_i$. Step 5 is not mentioned in Figure 3.3. In step 5 we create a view WITNESSES* that holds the minimal witnesses and the corresponding potential answers that the minimal witness give rise to, where the database facts that participate in a minimal witness are represented by their unique identifier in RELEVANT_R$_i$. The reason for assigning unique identifiers to tuples in PHASE 1 is to simplify the implementation of PHASE 2. Subsequently, PHASE 2 can directly create an array of binary variables for all facts in RELEVANT_R_i, where the position of each fact in the array corresponds to its unique identifier in RELEVANT_R_i. This is done to avoid building data structures in main memory to store the associations between variables in the programs and facts of relations RELEVANT_R_i. When constraints are generated from the minimal witnesses, it suffices to iterate over the tuples in WITNESSES*.

PHASE 1: DATABASE PRE-PROCESSING

```
Input: $\mathbf{R}$ : Schema with relation names $\{R_1, \cdots, R_i, \cdots, R_l\}$
    $q(\mathbf{z}) : -R_{p_1}(\underline{\mathbf{x_1}}, \mathbf{y_1}), \cdots, R_{p_j}(\underline{\mathbf{x_j}}, \mathbf{y_j}), \cdots, R_{p_k}(\underline{\mathbf{x_k}}, \mathbf{y_k})$ for $j \in [1..k]$ and $1 \leq p_j \leq l$
    $I$ : database over $\mathbf{R}$

Let:
  $<$ select_attributes $>$ be the list of attributes that correspond to free variables in $q$
  $<$ key_attr_$R_i$ $>$ be the list of key attributes of $R_i$
  $<$ relev_attr_$R_i$ $>$ be the relevant attributes of $R_i$

1. for all $R_i, 1 \leq i \leq l$
  create view KEYS_R$_i$ as:
      Select $R_i$.< key_attr_$R_i$ >
      From $R_i$
      Group By $R_i$.< key_attr_$R_i$ >
      Having count(*)>1
2. create view ANS_FROM_CON as:
      Select < select_attributes >
      From $R_1$ join $R_2$ join $\cdots$, join $R_l$
      left outer join KEYS_R$_1$ on $R_1$. < key_attr_$R_1$ >= KEYS_R$_1$. < key_attr_$R_1$ > $\cdots$
      left outer join KEYS_R$_i$ on $R_i$. < key_attr_$R_i$ >= KEYS_R$_i$. < key_attr_$R_i$ > $\cdots$
      left outer join KEYS_R$_l$ on $R_l$. < key_attr_$R_l$ >= KEYS_R$_l$. < key_attr_$R_l$ >
      Where
        KEYS_R$_1$. < key_attr_$R_1$ > is null and $\cdots$
        KEYS_R$_i$. < key_attr_$R_i$ > is null and $\cdots$
        KEYS_R$_l$. < key_attr_$R_l$ > is null
3. create view WITNESSES as:
      Select < relev_attr_$R_1$ >, $\cdots$ < relev_attr_$R_i$ >, $\cdots$ < relev_attr_$R_l$ >
      From $R_1$ join $R_2$ join $\cdots$, join $R_l$
      left outer join ANS_FROM_CON on
        ANS_FROM_CON.< select_attributes >=< select_attributes >
      Where ANS_FROM_CON.< select_attributes > is null
4. for all $R_i, 1 \leq i \leq l$, create view RELEVANT_R$_i$(ID_$R_i$, < relev_attr_$R_i$ >):
      Select distinct < relev_attr_$R_i$ >
      From $R_i$ inner join WITNESSES
        on $R_i$. < key_attr_$R_i$ >=WITNESSES. < key_attr_$R_i$ >
      Order by $R_i$. < key_attr_$R_i$ >
```

```
5. create view WITNESSES* as:
    Select distinct $< select\_attributes >, ID\_R_1, \cdots, ID\_R_i, \cdots, ID\_R_l$
    From WITNESSES, RELEVANT_$R_1$, $\cdots$, RELEVANT_$R_i$, $\cdots$, RELEVANT_$R_l$
    Where RELEVANT_$R_1$. $< relev\_attr\_R_1 >= WITNESSES. < relev\_attr\_R_1 > \cdots$
    Where RELEVANT_$R_i$. $< relev\_attr\_R_i >= WITNESSES. < relev\_attr\_R_i > \cdots$
    Where RELEVANT_$R_l$. $< relev\_attr\_R_l >= WITNESSES. < relev\_attr\_R_l >$
```

Figure A.1: Description of Phase 1 of EQUIP

A.2 Description of Phase 1 of EQUIP-AC

```
PHASE 1: DATABASE PRE-PROCESSING

Input: $\mathbf{R}$ : Schema with relation names $\{R_1, \cdots, R_i, \cdots, R_l\}$
      $q(\mathbf{z}) : -R_{p_1}(\underline{\mathbf{x_1}}, \mathbf{y_1}), \cdots, R_{p_j}(\underline{\mathbf{x_j}}, \mathbf{y_j}), \cdots, R_{p_k}(\underline{\mathbf{x_k}}, \mathbf{y_k})$ for $j \in [1..k]$ and $1 \leq p_j \leq l$
      $I$ : database over $\mathbf{R}$

Let: $< select\_attributes >$ be the list of attributes that correspond to free variables in $q$
  $< key\_attr\_R_i >$ be the list of key attributes of $R_i$
  $< relev\_attr\_R_i >$ be the relevant attributes of $R_i$

2. for all $R_i, 1 \leq i \leq l$
  create view KEYS_R$_i$ as:
        Select $R_i.< key\_attr\_R_i >$
        From $R_i$
        Group By $R_i.< key\_attr\_R_i >$
        Having count(*)>1
3. create view ANS_FROM_CON as:
        Select $< select\_attributes >$
        From $R_1$ join $R_2$ join $\cdots$, join $R_l$
        left outer join KEYS_R$_1$ on $R_1. < key\_attr\_R_1 >=$ KEYS_R$_1. < key\_attr\_R_1 > \cdots$
        left outer join KEYS_R$_i$ on $R_i. < key\_attr\_R_i >=$ KEYS_R$_i. < key\_attr\_R_i > \cdots$
        left outer join KEYS_R$_l$ on $R_l. < key\_attr\_R_l >=$ KEYS_R$_l. < key\_attr\_R_l >$
        Where
          KEYS_R$_1. < key\_attr\_R_1 >$ is null and $\cdots$
          KEYS_R$_i. < key\_attr\_R_i >$ is null and $\cdots$
          KEYS_R$_l. < key\_attr\_R_l >$ is null
```

```
4. create view POTENTIAL($< select\_attributes >, POT\_ID$) as:
        Select distinct $< select\_attributes >$
        From $R_1$ join $R_2$ join $\cdots$, join $R_l$
        left outer join KEYS_R$_1$ on $R_1. < key\_attr\_R_1 >=$ KEYS_R$_1. < key\_attr\_R_1 > \cdots$
        left outer join KEYS_R$_i$ on $R_i. < key\_attr\_R_i >=$ KEYS_R$_i. < key\_attr\_R_i > \cdots$
        left outer join KEYS_R$_l$ on $R_l. < key\_attr\_R_l >=$ KEYS_R$_l. < key\_attr\_R_l >$
        Where
          KEYS_R$_1. < key\_attr\_R_1 >$ is not null or $\cdots$
          KEYS_R$_i. < key\_attr\_R_i >$ is not null or $\cdots$
          KEYS_R$_l. < key\_attr\_R_l >$ is not null
   for all edges $(R_i, R_j)$ in $\tau$
       create view WITNESSES_$\{R_i, R_j\}$ as:
          Select $< relev\_attr\_R_1 >, \cdots < relev\_attr\_R_i >, \cdots < relev\_attr\_R_l >$, $POT\_ID$
          From $R_1$ join $R_2$ join $\cdots$ join $R_l$
            join POTENTIAL on POTENTIAL.$< select\_attributes >=< select\_attributes >$
5. for all $R_i, 1 \le i \le l$, create view RELEVANT_R$_i$($ID\_R_i, < relev\_attr\_R_i >$) as:
        Select distinct $< relev\_attr\_R_i >$
        From $R_i$ inner join WITNESSES_$R_j$
          on $R_i. < key\_attr\_R_i >=$WITNESSES_$\{R_i, R_j\}. < key\_attr\_R_i >$
        Order by $R_i. < key\_attr\_R_i >$
```

Figure A.2: Description of Phase 1 of EQUIP-AC

Appendix B

List of TPCH Queries Used to Evaluate EQUIP

Here we provide the list of queries which we ran over the TPCH database with EQUIP:

Q2:

select s_acctbal,s_name,n_name,p_partkey, p_mfgr,s_address,s_phone,s_comment,r_name from part,supplier,partsupp,nation,region
where p_partkey=ps_partkey and s_suppkey=ps_suppkey and s_nationkey=n_nationkey and n_regionkey=r_regionkey and p_size=15 and r_name='EUROPE'

Q3:

select l_orderkey,o_orderdate,o_shippriority from customer,orders,lineitem
where l_orderkey=o_orderkey and o_custkey=c_custkey and c_mktsegment='BUILDING' and o_orderdate<'1995-03-15' and l_shipdate≥'1995-03-15'

Q4:

select o_orderpriority from orders where o_orderdate≥'1993-07-01' and o_orderdate < '1993-10-01'

Q10:

select c_custkey,c_name,c_acctbal,n_name, c_address,c_phone,c_comment

from customer,orders,lineitem,nation

where c_custkey=o_custkey and l_orderkey=o_orderkey and c_nationkey=n_nationkey and l_returnflag='R' and o_orderdate≥'1993-10-01' and o_orderdate <'1994-01-01'

Q11:

select ps_partkey from supplier,partsupp,nation

where ps_suppkey=s_suppkey and s_nationkey=n_nationkey and n_name='GERMANY'

Q20:

select s_name,s_address from supplier,nation

where s_nationkey=n_nationkey and n_name='CANADA'

Q21:

select s_name from orders,lineitem,supplier,nation

where l_orderkey=o_orderkey and l_suppkey=s_suppkey and s_nationkey=n_nationkey and o_orderstatus='F' and l_receiptdate>l_commitdate and n_name='SAUDI ARABIA'

Appendix C

Complexity of certainty(q) for Specific Queries

C.1 PTIME algorithm for $q() : -R_1(\underline{x}, y), R_2(\underline{z}, x, y), R_3(\underline{y}, z)$

The query $q() : -R_1(\underline{x}, y), R_2(\underline{z}, x, y), R_3(\underline{y}, z)$ is such that CERTAINTY(q) is not first-order expressible. It does not satisfy the sufficient condition of Theorem 6 for intractability of CERTAINTY(q). It also does not satisfy the sufficient condition of Theorem 7 for tractability of CERTAINTY(q) because $key(R_2) \not\subseteq vars(R_1)$. However, according to our Conjecture 1, CERTAINTY(q) should be in P. And indeed, we will establish here that CERTAINTY(q) can be computed in polynomial time.

Lemma 6 *Let q be the query $q() : -R_1(\underline{x}, y), R_2(\underline{z}, x, y), R_3(\underline{y}, z)$. Then* CERTAINTY($q$) *is in P*

PROOF. We will prove that CERTAINTY(q) is in P by using a slightly different version of the conflict-join graph. We will define the construction of a graph $H_{I,q,\tau}*$ for this query, where τ is the join tree with edges (R_1, R_2) and (R_2, R_3).

Let I be a database instance over the same schema as q. The conflict-join graph $H_{I,q,\tau}*$ is constructed as follows:

1. For every R_i-fact f, for every $R_j \in N_\tau(R_i)$ in τ, add a vertex f^{R_j}.

2. Add an edge between every two vertices f^{R_i} and g^{R_j} where f and g are key-equal facts in I.

3. For every two atoms R_i and R_j that are neighbors in τ, for every $\{f, g\} \in R_i \times R_j$ such that $\{f, g\}$ is a minimal witness of $q_{\{R_i,R_j\}}$ in I, add an edge between vertices f^{R_j} and g^{R_i}.

4. For every set X of facts from I that is minimal with the property that, (a) there exists in X a minimal witness $\{f, g\}$ of $q_{\{R_i,R_j\}}$ and (b) every other minimal witness of the form $\{f, g'\}$ or $\{f', g\}$ is also in X, ad an edge between every two vertices f^{R_j} and f'^{R_j}, and between every two vertices g^{R_i} and g'^{R_i}. Let this set of edges be named $E*$.

Note that $H_{I,q,\tau}*$ has more edges than $H_{I,q,\tau}$. Let s be the number of key-equal groups in I. We will prove that there is an independent set M of $H_{I,q,\tau}$ such that $|M| = s$ if and only if there is an independent set $M*$ of $H_{I,q,\tau}*$ such that $|M*| = s$.

In one direction, it is obvious that if $M*$ is an independent set of $H_{I,q,\tau}*$ such that $|M*| = s$, then $M*$ itself is an independent set of $H_{I,q,\tau}$.

In the opposite direction, assume that M is an independent set of $H_{I,q,\tau}$ that has size s. We will show how to construct $M*$ from M:

- For every vertex f^{R_2} in M, add f^{R_2} to $M*$.
- For every vertex f^{R_3} in M, add f^{R_3} to $M*$.
- For every vertex f^{R_1} in M, if there exists a vertex g^{R_2} in M, where g is an R_3-fact and $\{f, g\}$ is a minimal witness of $q_{\{R_2,R_3\}}$, then add f^{R_1} to $M*$; otherwise, add f^{R_3} to $M*$.

We will argue that $M*$ is an independent set of $H_{I,q,\tau}*$ of size s. First, from the construction of $M*$ it is obvious that it has size equal to $|M|$. Assume towards a contradiction that $M*$ is not independent. Let e be an edge induced by $M*$. Then e must be an edge in $E*$. Because $keys(R_1) \subseteq vars(R_2)$, then, if $\{f, g\}$ and $\{f',\}$ are minimal witnesses of $q_{\{R_1,R_2\}}$, the facts f and f' must be key-equal. Therefore, the edge (f^{R_2}, f'^{R_2}) is in $H_{I,q,\tau}$. From the construction of $M*$ and the fact that M is an independent set, it follows that the edge e cannot be of the form (f^{R_2}, f'^{R_2}), where f and f' are R_1-facts. Similarly, because $keys(R_3) \subseteq vars(R_2)$, the edge e cannot be of the form (f^{R_2}, f'^{R_2}) where f and f' are R_3-facts; and because $keys(R_2) \subseteq vars(R_3)$, the edge e cannot be of the form (f^{R_3}, f'^{R_3}) where f and f' are R_2-facts. Then, it could only happen that e is of the form (f^{R_1}, f'^{R_1}). By construction of $M*$, it must be that there exists a vertex g^{R_2} in M such that g is an $R_3 - fact$ and $\{f, g\}$ is a minimal witness of $q_{\{R_2,R_3\}}$; and there exists a vertex g'^{R_2} in M, such that g is an $R_3 - fact$ and $\{f', g'\}$ is a minimal witness of $q_{\{R_2,R_3\}}$. Because e is in $E*$, then both facts f, f' form a minimal witness with some R_1-fact h. Let f be a fact of the form $R_2(t)$ and let f' be a fact of the form $R_2(t')$. Then, $t[x, y] = t'[x, y]$ and $t[z] \neq t'[z]$. Let g be the fact $R_3(d)$ and let g' be the fact $R_3(d')$. Then $d[y] = t[y]$ and $d'[y] = t'[y]$. It follows that d and d' are such that $d[y] = d'[y]$ and $d[z] \neq d'[z]$. Then g and g' must be key-equal, and consequently, the vertices g^{R_2} and g'^{R_2} must induce an edge in M. This contradicts the assumption that M is independent. □

C.2 coNP-hardness proof for $Q_{22}(v) : -R_5(\underline{x}, y, z), R_6(\underline{x'}, y, z), R_{11}(\underline{x, x'}, v)$

We prove here that for the query $Q_{22}(v) : -R_5(\underline{x}, y, z), R_6(\underline{x'}, y, z), R_{11}(\underline{x, x'}, v)$, CERTAINTY($Q_{22}$) is coNP-hard.

Lemma 7 *Let $Q_{22}(v) : -R_5(\underline{x}, y, z), R_6(\underline{x'}, y, z), R_{11}(\underline{x, x'}, v)$. Then* CERTAINTY($Q_{22}$) *is coNP-hard*

PROOF. We prove that CERTAINTY(Q_{22}) is coNP-hard by giving a polynomial reduction from CERTAINTY(q) where q is the query $q() : -S_5(\underline{x}, y, z), S_6(\underline{x'}, y, z)$. It can be established that CERTAINTY(q) is coNP-hard from the dichotomy for two-atom queries presented in Section 4.2. Let I be a database over the schema of q. We construct a database I' over the schema of Q_{22} by copying S_5 to R_5, copying S_6 to R_6, and generating a fact $R_{11}(a, a', N_a)$ from every minimal witness $\{f, g\}$ of q, where f is a fact of the form $S_5(a, \_, \_)$, g is a fact of the form $S_6(a', \_, \_)$, and N_a is a new generated value (a labeled null). It is easy to see that R_{11} is consistent; hence, it appears in every repair of I'. Then, given any repair r of I, once can create a repair r' of I' by copying R_5^r to $S_5^{r'}$ and R_6^r to $S_6^{r'}$. Similarly, given any repair r' of I', once can create a repair r of I by copying $S_5^{r'}$ to R_5^r and $S_6^{r'}$ to R_6^r, and keeping the entire relation R_{11} in r. Obviously, if r does not satisfy Q_{22}, then r' does not satisfy q, and vice-versa. $\square$