November 2017

# The Complexity of Resilience

Cibele Matos Freire
*University of Massachusetts Amherst*

# THE COMPLEXITY OF RESILIENCE

A Thesis Presented

by

CIBELE FREIRE

Submitted to the Graduate School of the
University of Massachusetts Amherst in partial fulfillment
of the requirements for the degree of

DOCTOR OF PHILOSOPHY

September 2017

College of Information and Computer Sciences

# THE COMPLEXITY OF RESILIENCE

A Thesis Presented

by

CIBELE FREIRE

Approved as to style and content by:

Neil Immerman, Chair

Alexandra Meliou, Member

Andrew McGregor, Member

Wolfgang Gatterbauer, Member

Barbara Partee, Member

James Allan, Chair of the Faculty
College of Information and Computer Sciences

# ABSTRACT

## THE COMPLEXITY OF RESILIENCE

SEPTEMBER 2017

CIBELE FREIRE

B.Sc., UNIVERSIDADE FEDERAL DO CEARÁ

M.Sc., UNIVERSIDADE FEDERAL DO CEARÁ

M.Sc., UNIVERSITY OF MASSACHUSETTS AMHERST

Ph.D., UNIVERSITY OF MASSACHUSETTS AMHERST

Directed by: Professor Neil Immerman

One focus area in data management research is to understand how changes in the data can affect the output of a view or standing query. Example applications are explaining query results and propagating updates through views. In this thesis we study the complexity of the Resilience problem, which is the problem of finding the minimum number of tuples that need to be deleted from the database in order to change the result of a query. We will see that resilience is closely related to the well-studied problems of deletion propagation and causal responsibility, and that analyzing its complexity offers important insight for solving those problems as well.

Our contributions include the definition of the concept of triads for conjunctive queries, which is a crucial tool on our analysis, and the characterization of a NP versus P dichotomy for the resilience problem considering the class of conjunctive queries without self-joins. Moreover, this result allowed us to show dichotomies for

the same class of queries for both deletion propagation with source side-effects and causal responsibility problems. We also completely characterize how the presence of functional dependencies can change the complexity of such problems.

The class of conjunctive queries with self-joins is far richer and more complicated than the self-join-free ones. Therefore we focus on *binary queries* without variable repetition, which are queries formed by unary or binary relations only and each atom has only one occurrence of any variable. For this restricted case, we identify three main query structures that help us identify complexity: chains, permutations and confluences. Using those we are able to characterize classes of queries for which resilience is NP-complete and some for which it is P.

# TABLE OF CONTENTS

# LIST OF FIGURES

**Figure** **Page**

# CHAPTER 1

# INTRODUCTION

As data continues to grow in volume, the results of relational queries become harder to understand, interpret, and debug through manual inspection. Data management research has recognized this fundamental need to derive *explanations for query results* and explanations for surprising observations. Existing work has defined explanations as predicates in a query [41, 37, 11], or as modifications to the input data [35, 28, 27]. In the latter category, the metric of *causal responsibility*, first introduced by [13], quantifies the contribution of an input tuple to a particular output. One can then derive explanations by *ranking input tuples* using their responsibilities: tuples with high degree of responsibility are better explanations for a particular query result than tuples with low responsibility [35].

A seemingly unrelated notion, the concept of *deletion propagation with source side-effects* [9], seeks a minimum set of tuples in the input tables that should be deleted from the database in order to delete a particular tuple from a query. Query results that have a larger set of tuples that need to be deleted are more reliable or more "robust" to changes in the input database than others.

In this thesis, we take a step back and re-examine how particular interventions (*tuple deletions in the input of a query*) impact its output. Specifically, we study how "resilient" a Boolean query is with respect to such interventions. *Resilience* identifies the smallest number of tuples to delete from the input to make the query false. We will show that characterizing the complexity of this problem also allows us

to study the complexities of *both* deletion propagation with source side-effects and causal responsibility.

The main contributions of this work are:

1. Complete characterization of the complexity of resilience for conjunctive queries without self-joins, resulting in a NP versus PTIME dichotomy;
2. Dichotomy theorem for the self-join-free case in the presence functional dependencies;
3. Complete characterization of the complexity of responsibility for conjunctive queries without self-joins with functional dependencies, also resulting in a NP versus PTIME dichotomy;
4. Characterization of the complexity of resilience for certain classes of conjunctive queries with self-joins.

The thesis proceeds as follows. Chapter 2 covers the background and notation and discusses related work comprehensively, showing the close connections between deletion propagation, causal responsibility and resilience. Chapter 3 presents our dichotomy results for the resilience of conjunctive queries without self-joins and with functional dependencies. Chapter 4 discusses resilience for queries with self-joins. We explain how different from the self-join-free case this one is, in particular, that we need a more diverse set of structures than just triads to characterize hardness, and that we need to consider other query elements such as order and repetition of variables. In Chapter 5, we characterize the complexity of causal responsibility considering again conjunctive queries without self-joins and we obtain similar results of that for resilience. Finally, Chapter 6 presents our conclusions and future work.

# CHAPTER 2

# BACKGROUND AND RELATED WORK

In this chapter we present the background and formal setup, and give an overview of problems related to resilience, giving particular attention to the deletion propagation and causal responsibility problems.

## 2.1 General notation

We use boldface (e.g., $\mathbf{x} = (x_1, \ldots, x_k)$) to denote tuples or ordered sets. A *conjunctive query* (CQ) is a first-order formula $q(\mathbf{y}) = \exists \mathbf{x} \, (A_1 \wedge \ldots \wedge A_m)$ where the variables $\mathbf{x} = (x_1, \ldots, x_k)$ are called *existential variables*, $\mathbf{y} = (y_1, \ldots, y_c)$ are called the *head variables* (or free variables). Each atom $A_i$ represents a relation $R_i(\mathbf{z}_i)$ where $\mathbf{z}_i \subseteq \mathbf{x} \cup \mathbf{y}$ and we say that a query is *self-join free* (sj-free) if no relation symbol occurs more than once, otherwise we have a query with self-join (sj). We often refer to self-join free conjunctive queries as sj-free CQ, and to conjunctive queries with self-joins as sj-CQ. If a query has no head variables, i.e. $\mathbf{y} = \emptyset$, we say it is a Boolean query and its output is either `true` or `false`, otherwise it is a non-Boolean query.

We write $\texttt{var}(A_j)$ for the set of variables occurring in atom $A_j$. The database instance is then the union of all tuples in the relations $D = \bigcup_i R_i$. As usual, we abbreviate the query in Datalog notation by $q(\mathbf{y}) :- A_1, \ldots, A_m$. For tuple $\mathbf{t}$, we write $D \models q[\mathbf{t}/\mathbf{y}]$ to denote that $\mathbf{t}$ is in the query result of the non-Boolean query $q(\mathbf{y})$ over database $D$. The set of query results over database $D$ is denoted by $q(\mathbf{y})^D$.

We write $D \models q$ to denote that the query $q$ evaluates to `true` over the database instance $D$, and $D \not\models q$ to denote that $q$ evaluates to `false`.

**Definition 2.1** (Witness)**.** *We call a valuation of all existential variables that is permitted by $D$ and that makes $q :- A_1, \ldots, A_m$* `true`*, a* witness $\mathbf{w}$*. The set of witnesses of $D \models q$ is the set* $\{\mathbf{w} \mid D \models (A_1 \wedge \ldots \wedge A_m)[\mathbf{w}/\mathbf{x}]\}$.

Notice that our notion of witness slightly differs from the one commonly seen in provenance literature where a "witness" refers to a subset of the input database records that is sufficient to ensure that a given output tuple appears in the result of a query [12].

A database instance may contain some "forbidden" tuples that may not be deleted. Since we are interested in the data complexity of resilience, we specify *at the query level* which tables contain tuples that may or may not be deleted. Those atoms from which tuples may not be deleted are called *exogenous*[1] and we write these atoms or relations with a superscript "x". The other atoms, whose tuples may be deleted, are called *endogenous*. We may occasionally attach the superscript "n" to an atom to emphasize that it is endogenous. Moreover, we can refer to a database as a partition of its tables into its exogenous and endogenous parts, $D = D^{\mathrm{x}} \cup D^{\mathrm{n}}$.

## 2.2 Query resilience

Given $D \models q$, our motivating question is: what is the minimum number of tuples to remove in order to make the query false? The formal definition of the resilience problem:

**Definition 2.2** (Resilience)**.** *Given a query $q$ and database $D$, we say that $(D, k) \in$* `RES`$(q)$ *if and only if $D \models q$ and there exists some $\Gamma \subseteq D^{\mathrm{n}}$ such that $D - \Gamma \not\models q$ and $|\Gamma| \leq k$.*

[1] In other words, tuples in these atoms provide context and are outside the scope of possible "interventions" in the spirit of causality [25].

In other words, $(D, k) \in \texttt{RES}(q)$ means that there is a set of $k$ or fewer tuples in the endogenous tables of $D$, the removal of which makes the query false. Observe that since $q$ is computable in PTIME, $\texttt{RES}(q)$ is in NP. We will see that there is a dichotomy for all sj-free conjunctive queries: for all such queries $q$, either $\texttt{RES}(q)$ is in PTIME or $\texttt{RES}(q)$ is NP-complete (Theorem 3.24). We are naturally interested in the optimization version of this decision problem: given $q$ and $D$, find the *minimum* $k$ so that $(D, k) \in \texttt{RES}(q)$. A larger $k$ implies that the query is more "*resilient*" with respect to a given database and requires the deletion of more tuples to change the query output.

We focus on Boolean queries, however we can also define the resilience problem for non-Boolean queries as follows:

**Definition 2.3** (Resilience for non-Boolean queries)**.** *Given non-Boolean query $q(\mathbf{y})$ and database $D$, we say that $(D, k) \in \texttt{RES}(q(\mathbf{y}))$ if and only if $q(\mathbf{y})^D \neq \emptyset$ and there exists some $\Gamma \subseteq D^{\text{n}}$ such that $q(\mathbf{y})^{D-\Gamma} = \emptyset$ and $|\Gamma| \leq k$.*

It is clear from the definition that we are interested in eliminating all the output tuples from the query result, and it is easy to see that $\texttt{RES}(q(\mathbf{y})) \equiv \texttt{RES}(q')$, where $q'$ is obtained by removing all variables $\mathbf{y}$ from the head of $q$, turning them into existential variables.

## 2.3 Related work

In this section we make a direct connection between the problems of resilience, deletion propagation and causal responsibility. After reading this section, the reader should be able to understand how our results on the complexity of resilience contribute to the understanding of those problems.

### 2.3.1 Deletion propagation

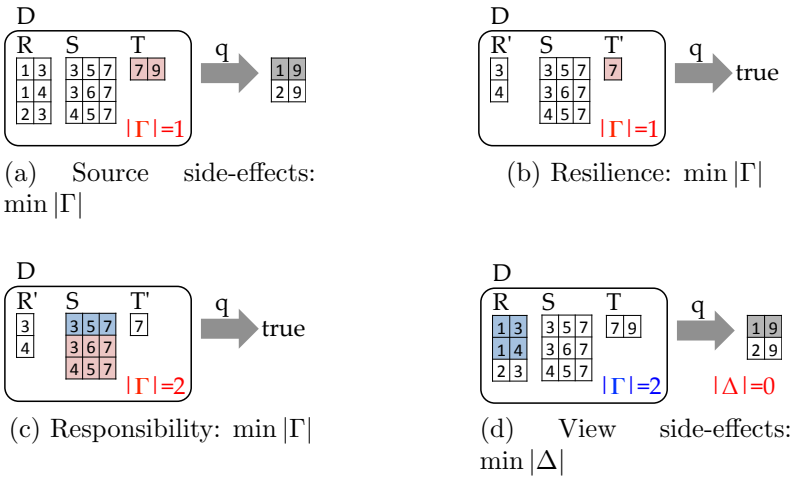
Databases allow users to interact with data through views, which are often conjunctive queries. Views can be used to simplify complex queries, enforce access control policies, and preserve data independence for external applications. Of particular interest is how deletions in the input data affect the view (which is a trivial problem), but also how deletions in the view could be achieved by appropriately chosen deletions in the input data (which is far less trivial). Concretely, the problem of *deletion propagation* [9, 17] seeks a set $\Gamma$ of tuples in the input tables that should be deleted from the database in order to delete a particular tuple from the view. Intuitively, this deletion should be achieved with *minimal side-effects*, where side-effects are defined with either of two objectives: (a) deletion propagation with *source side-effects* ($\texttt{DP}_{\texttt{source}}$) seeks a minimum set of input tuples $\Gamma$ in order to delete a given output tuple; whereas (b) deletion propagation with *view side-effects* ($\texttt{DP}_{\texttt{view}}$) seeks a set of input tuples $\Gamma$ whose removal results in a minimum number of output tuple deletions in the view, other than the tuple of interest [9].

**Example 2.4** (Source & View side-effects)**.** *Consider the query*

$$q(x, u) :- R(x, y), S(y, z, w), T(w, u)$$

*defining a view over the database $R, S, T$ shown below. To delete tuple $v_1$ from the resulting view with minimum source side-effects, one only needs to remove tuple $t_1$ from the database. Therefore, the optimal solution to $\texttt{DP}_{\texttt{source}}$ is $\Gamma = \{t_1\}$ with $|\Gamma| = 1$ (see Fig. 2.1a).*

*However, the deletion of $t_1$ also removes $v_2$, which is a view side-effect: $\Delta = \{v_2\}$ with $|\Delta| = 1$. The optimal solution to $\texttt{DP}_{\texttt{view}}$, which minimizes the side-effects on the view (set $\Delta$) is the set of input tuples $\Gamma = \{r_1, r_2\}$: deleting these two tuples removes only $v_1$ from the view but not $v_2$, and thus has no view-side effects, i.e., $\Delta = \emptyset$ with $|\Delta| = 0$ (see Fig. 2.1d).*




(a) Source side-effects: min $|\Gamma|$

(b) Resilience: min $|\Gamma|$

(c) Responsibility: min $|\Gamma|$

(d) View side-effects: min $|\Delta|$

Figure 2.1: Example illustrating similarities and differences between (a) deletion propagation with source side-effects, (b) resilience, (c) responsibility for causality, and (d) deletion propagation with view side-effects

| | $R$ | |
|---|---|---|
| | $X$ | $Y$ |
| $r_1$ | 1 | 3 |
| $r_2$ | 1 | 4 |
| $r_3$ | 2 | 3 |

| | $S$ | | |
|---|---|---|---|
| | $Y$ | $Z$ | $W$ |
| $s_1$ | 3 | 5 | 7 |
| $s_2$ | 3 | 6 | 7 |
| $s_3$ | 4 | 5 | 7 |

| | $T$ | |
|---|---|---|
| | $W$ | $U$ |
| $t_1$ | 7 | 9 |

| | $q$ | |
|---|---|---|
| | $X$ | $U$ |
| $v_1$ | 1 | 9 |
| $v_2$ | 2 | 9 |

**Deletion propagation: source side-effects**

Deletion propagation in view updates generally refers to non-Boolean queries $q(\mathbf{y}) :- A_1, \ldots, A_m$. We next define the problem [9, 17] formally in our notation:

**Definition 2.5** (Source side-effects)**.** *Given a query $q(\mathbf{y})$, database $D$, and an output tuple $t$, we say that $(D, t, k) \in \mathtt{DP}_{\mathtt{source}}(q(\mathbf{y}))$ if and only if $t \in q(\mathbf{y})^D$ and there exists some $\Gamma \subseteq D$ such that $t \notin q(\mathbf{y})^{D-\Gamma}$ and $|\Gamma| \leq k$.*

We will see that there is a homomorphism between resilience and the source side-effect variant of deletion propagation. We illustrate this correspondence in the example below and next describe this transformation more formally.

**Example 2.6** (Resilience & Source side-effect)**.** *Consider again the query from Example 2.4 and the output tuple $v_1 = (1, 9)$. Applying the substitution $[(x, u)/(1, 9)]$,*

*i.e., substituting $x$ and $u$ with 1 and 9, respectively, we get a query $q(1,9) :- R(1,y), S(y,z,w), T(w,9)$. The solution to $\texttt{DP}_{\texttt{source}}$ for $q$ and tuple $v_1$ is then equivalent to the solution of the resilience problem over the Boolean query $q' :- R'(y), S(y,z,w), T'(w)$ over the database $R', S, T'$ with $R'(y) :- R(1,y)$ and $T'(w) :- T(w,9)$ shown below. The answer to the resilience problem for $q'$ is $\Gamma = \{t'_1\}$ with $|\Gamma| = 1$: deleting tuple $t'_1$ makes the query false (also see Fig. 2.1b).*

| | $R'$ |
|---|---|
| | $Y$ |
| $r'_1$ | 3 |
| $r'_2$ | 4 |

| | $S$ | | |
|---|---|---|---|
| | $Y$ | $Z$ | $W$ |
| $s_1$ | 3 | 5 | 7 |
| $s_2$ | 3 | 6 | 7 |
| $s_3$ | 4 | 5 | 7 |

| | $T'$ |
|---|---|
| | $W$ |
| $t'_1$ | 7 |

Given a conjunctive query $q(\mathbf{y}) :- A_1, \ldots, A_m$ and a tuple $t = \mathbf{c}$ in the output $q(\mathbf{y})^D$. We first obtain a Boolean query $q'$ by deleting the head variables in $q(\mathbf{y})$. Then we modify the database by applying a filter (selection): for each relation $R_i(\mathbf{z}_i)$ we define a new relation $R'_i(\mathbf{x}_i) :- R_i(\theta_t(\mathbf{z}_i))$ with $\mathbf{x}_i$ being the existential variables that occur in $R_i$, and where the substitution $\theta_t : \mathbf{y} \to \mathbf{c}$ replaces the former head variables with the corresponding constants from $t$ and keep the existential variables as they are. For example, $R'(y) :- R(1,y)$ in Example 2.6 (see Fig. 2.1a and Fig. 2.1b). This will lead to a new database $D' = \bigcup_i R'_i$ and a new Boolean query $q' :- A'_1, \ldots, A'_m$, where $A'_i = R'_i(\mathbf{x}_i)$ if $A_i = R_i(\mathbf{z}_i)$, for which the following holds:

**Corollary 2.7** (Resilience & Source side-effects)**.** *Given a query $q(\mathbf{y})$, database $D$, and output tuple $t \in q(\mathbf{y})^D$, let $q'$ and $D'$ be the new Boolean query and new database instance obtained by the above transformation. Then: $(D,t,k) \in \texttt{DP}_{\texttt{source}}(q(\mathbf{y})) \Leftrightarrow (D',k) \in \texttt{RES}(q')$.*

Notice that by solving the complexity of resilience, we immediately also solve the problem of deletion propagation with source side-effects. We proceed to discuss existing results on the complexity of deletion propagation with source side-effects, and explain how our results on the complexity of resilience extend this prior work.

[9] define a dichotomy for the hardness of $\texttt{DP}_{\texttt{source}}(q)$ based only on the operations that occur in $q$, namely, selection, projection, join, union. Specifically, they show that $\texttt{DP}_{\texttt{source}}(q(\mathbf{y}))$ is NP-complete for PJ and JU queries (i.e., queries involving projections and joins, or queries involving joins and unions), while it is PTIME for SJ and SPU queries (i.e., queries involving selections and joins, or queries involving selections, projections, and unions only). Later, [14] showed that $\texttt{DP}_{\texttt{source}}(q(\mathbf{y}))$ is in PTIME for a SPJ query if all primary keys of the involved relations appear in the head variables $\mathbf{y}$ (a condition called "key preservation"). Notice that the concept of key preservation does not apply to the problem of resilience, as keys are never preserved in Boolean queries.

Our results on resilience imply a refinement for the complexity of minimum source side-effects by defining a novel, yet simple and intuitive property of the query structure called "*triads*." For the class of self-join-free conjunctive queries, we show that resilience is NP-complete if the query contains this structure, and PTIME otherwise (Chapter 3). Determining whether a query contains a triad can be done very efficiently, in polynomial time with respect to query complexity. These results are analogous to the results of [33] for the deletion propagation with view-side effect problem. In addition, our dichotomy criterion also allows the specification of "forbidden" tables (called *exogenous* tables) that do not allow deletions. This is an extension to the traditional definition of the deletion propagation problem and affects the complexity of queries in non-obvious ways (defining a table as exogenous can make both easy queries hard, and hard queries easy).

Our work also provides a complete dichotomy result for the class of sj-free conjunctive queries with Functional Dependencies (Section 3.2). At a high-level, we define rewrite steps that are induced by the functional dependencies, and check the resulting query for the presence of triads.

**Deletion propagation: view side-effects**

The problem of deletion propagation with view side-effects has a different objective than resilience: it attempts to minimize the changes in the view rather than the source.

**Definition 2.8** (View side-effects)**.** *Given a query $q(\mathbf{y})$, a database $D$, and a tuple $t$ in the view, we say that $(D, t, k) \in \mathtt{DP}_{\mathtt{view}}(q(\mathbf{y}))$ if and only if $t \in q(\mathbf{y})^D$ and there exists some $\Gamma \subseteq D$ such that $t \notin q(\mathbf{y})^{D-\Gamma}$, and $|\Delta| \leq k$, where $\Delta = (q(\mathbf{y})^D - (q(\mathbf{y})^{D-\Gamma} \cup \{t\}))$. In other words, $\Delta$ is the set of tuples other than $t$ that were eliminated from the view.*

The dichotomy results from [9] extend to the case of $\mathtt{DP}_{\mathtt{view}}(q)$, and the same is true for key preservation [14]. Later, [33] refined the dichotomy for the view side-effect problem by providing a characterization that uses the query structure: $\mathtt{DP}_{\mathtt{view}}(q(\mathbf{y}))$ is PTIME for queries that are *head dominated*, and NP-complete otherwise. Head domination checks for the components of the query that are connected by the existential variables, where all head variables contained in the atoms of that component appear in a single atom in the query.

Kimelfeld augmented the dichotomy on $\mathtt{DP}_{\mathtt{view}}(q)$ for cases where functional dependencies (FDs) hold over the data instance $D$ [32]. The tractability condition for this case checks whether the query has *functional head domination*, which is an extension of the notion of head domination.

A variant of deletion propagation that aims to remove a group of tuples from the view instead of a single target was studied in [14]. Their results classify all conjunctive queries as NP-complete, but recently, [34] provided a trichotomy for the class of sj-free CQs that extends the notion of head domination, classifying queries into PTIME, $k$-approximable in PTIME, and NP-complete.

### 2.3.2 Causal responsibility

The problem of causal responsibility [35] seeks, for a given query and a specified input tuple, a minimum set of other input tuples $\Gamma$ that, if deleted would make the tuple of interest "counterfactual," i.e., the query would be true with that tuple present, or false if the tuple was also deleted. Both problems of resilience and of causal responsibility rely on the notion of minimal interventions in the input database and are thus closely related. However, we show that resilience is easier (has lower complexity) than responsibility.

A tuple $t$ is a *counterfactual cause* for a query if by removing it the query changes from `true` to `false`. A tuple $t$ is an *actual cause* if there exists a set $\Gamma$, called the *contingency set*, removing of which makes $t$ a counterfactual cause. Determining actual causality is NP-complete for general formulas [18], but there are families of tractable cases [19]. Specifically, causality is PTIME for all conjunctive queries [35]. Responsibility measures the *degree* of causal contribution of a particular tuple $t$ to the output of a query as a function of the size of a minimum contingency set: $\rho = \frac{1}{1+\min|\Gamma|}$. These definitions stem from the work of [25] and [13],and were adapted to queries in [35]. Even though responsibility ($\rho$) was originally defined as inversely proportional to the size of the contingency set $\Gamma$, here we alter this definition slightly to draw parallels to the problem of resilience.

**Definition 2.9** (Responsibility)**.** *Given query $q$, we say that $(D, t, k) \in \mathtt{RSP}(q)$ if and only if $D \models q$ and there is $\Gamma \subseteq D^{\mathrm{n}}$ such that $D - \Gamma \models q$ and $|\Gamma| \leq k$ but $D - (\Gamma \cup \{t\}) \not\models q$.*

**Example 2.10** (Resilience & Causal responsibility)**.** *The causal responsibility problem requires a tuple in the lineage of the query as additional input. Consider query $q'$ and database $R', S, T'$ from Example 2.6. The responsibility of tuple $s_1$ in query $q'$ corresponds to the contingency set $\Gamma = \{s_2, s_3\}$ with $|\Gamma| = 2$. Deleting these two*

*tuples makes $s_1$ a counterfactual cause for $q'$, i.e., the query is true if $s_1$ is present or false, otherwise (also see Fig. 2.1c).*

In contrast to resilience, the problem of responsibility is defined for *a particular tuple $t$* in $D$, and instead of finding a $\Gamma$ that will leave no witnesses for $D - \Gamma \models q$, we want to preserve only witnesses that involve $t$, so that there is no witness left for $D - (\Gamma \cup \{t\}) \models q$. This difference, while subtle, is significant, and can lead to different results. In Example 2.6, the resilience of query $q'$ has size 1 and contains tuple $t_1$. However, the solution to the responsibility problem in Example 2.10 depends on the chosen tuple: the contingency set of $s_1$ has size 2, and this size can be made arbitrarily bigger by adding more tuples in $S$ with attribute $W = 7$. Furthermore, we show that the problems differ in terms of their complexity.

For completeness, we briefly recall the notions of *reduction* and *equivalence* in complexity theory:

**Definition 2.11** (Reduction ($\leq$) and Equivalence ($\equiv$))**.** *For two decision problems, $S, T \subseteq \{0,1\}^*$, we say that $S$ is* reducible *to $T$ ($S \leq T$) if there is an* easy to compute reduction $f : \{0,1\}^* \to \{0,1\}^*$ *such that*

$$\forall w \in \{0,1\}^* \big( w \in S \Leftrightarrow f(w) \in T \big) \ .$$

*The idea is that the complexity of $S$ is less than or equal to the complexity of $T$ because any membership question for $S$ (i.e., whether $w \in S$) can be easily translated into an equivalent question for $T$, (i.e., whether $f(w) \in T$). "Easy to compute" can be taken as expressible in first-order logic*[2]*. We say that two problems have* equivalent *complexity ($S \equiv T$) iff they are inter-reducible, i.e., $S \leq T$ and $T \leq S$.*

[2]All reductions presented in this work are first-order, i.e., when we write $S \leq T$ we mean $S \leq_{\text{fo}} T$. First-order reductions are natural for the relational database setting and they are more restrictive than logspace reductions, which in turn are more restrictive than polynomial-time reductions ($S \leq_{\text{fo}} T \Rightarrow S \leq_{\log} T \Rightarrow S \leq_{\text{p}} T$) [29].

The problem of calculating resilience can always be reduced to the problem of calculating responsibility.

**Lemma 2.12** ($\texttt{RES} \leq \texttt{RSP}$)**.** *For any query $q$, $\texttt{RES}(q) \leq \texttt{RSP}(q)$, i.e., there is a reduction from $\texttt{RES}(q)$ to $\texttt{RSP}(q)$. Thus, if $\texttt{RES}(q)$ is hard (i.e., NP-complete) then so is $\texttt{RSP}(q)$. Equivalently, if $\texttt{RSP}(q)$ is easy (i.e., PTIME) then so is $\texttt{RES}(q)$.*

*Proof.* Let $q :- \exists x_1, \ldots, x_s\, A_1(\mathbf{z_1}) \wedge \cdots \wedge A_r(\mathbf{z_r})$. The reduction from $\texttt{RES}(q)$ to $\texttt{RSP}(q)$ is as follows: given $(D, k)$, we map it to $(D', \mathbf{t_0}, k)$ where $D'$ consists of the database $D$ together with unique new values $a_1, \ldots a_s$ and the new tuples $A_1(\mathbf{z_1}[\mathbf{a}/\mathbf{x}]), \ldots, A_r(\mathbf{z_r}[\mathbf{a}/\mathbf{x}])$. In other words, we enter a completely new witness $\mathbf{a}$ for $q$ that has no values in common with the domain of $D$. Let $\mathbf{t_0} = A_1(\mathbf{z_1}[\mathbf{a}/\mathbf{x}])$, i.e., the tuple of these new values from atom $A_1$. It follows that the size of the minimal contingency set for $q$ in $D$ is the same as the size of the minimal contingency set for $q$ and $\mathbf{t_0}$ in $D'$. Thus, as desired, $(D, k) \in \texttt{RES}(q) \Leftrightarrow (D', \mathbf{t_0}, k) \in \texttt{RSP}(q)$. □

### 2.3.3 Additional related problems

Sections 2.3.1 and 2.3.2 have extensively discussed prior work and the connections between resilience, deletion propagation and causal responsibility [9, 14, 32, 33]. In this section, we discuss additional related work.

**Data provenance.** Data provenance studies formalisms that can characterize the relation between the input and the output of a given query [8, 12, 16, 24]. Among the kinds of provenance, "Why-provenance" is the most closely related to resilience in databases. The motivation behind Why-provenance is to find the "witnesses" for the query answer, i.e., the tuples or group of tuples in the input that can produce the answer. Resilience, searches to find a *minimum* set of input tuples that can make a query false.

**View updates.** The view update problem is a classical problem studied in the database literature [5, 14, 15, 17, 21, 30]. In its general form, the problem consists of

finding the set of operations that should be applied to the database in order to obtain a certain modification in the view. Resilience and deletion propagation are special cases of view updates.

**Causality.** The study of causality is important in many areas other than databases, for example in Artificial Intelligence and Philosophy. Although an intuitive concept, it is difficult to formally define causality and many authors have presented possible definitions of causality. In [35], the notions of causality and responsibility were strongly inspired by the work of Halpern and Pearl [13, 25]. Causal reasoning is based on the idea of *interventions*: understand how changes of input variables affect an outcome, and thus relates in spirit to resilience. In the case of resilience, the intervention is the deletion of input tuples.

**Explanations in Databases.** Providing explanations to query answers is important because it can help identify inconsistencies and errors in the data, as well as understand the data and queries that operate on it. Causality can provide a framework for explanations of query results [35, 36], but it relies on the computation of responsibility, which is a harder problem than resilience. Other work on explanations also applies interventions, but on the queries instead of the data [37, 41]. These approaches, try to understand how the deletion, addition, or modification of predicates may affect the result of a query. There are also other approaches on deriving explanations that focus on specific database applications [2, 6, 7, 20, 31, 39]. Finally, the problem of explaining *missing* query results [11, 26, 28, 27, 40] is a problem analogous to deletion propagation, but in this case, we want to add, rather than remove tuples from the view.

# CHAPTER 3

# COMPLEXITY OF RESILIENCE FOR SJ-FREE QUERIES

In this chapter we present the complexity results for the resilience problem when considering sj-free conjunctive queries (sj-free CQ). We show how we extend these results to consider the presence of functional dependencies (FDs).

The results in this chapter are published in [23], and a full version with all the proofs can be found in [22]

## 3.1 Complexity of resilience: sj-free case

In this section we present the data complexity of resilience. We prove that the complexity of resilience of a query $q$ can be exactly characterized via a natural property of its *dual hypergraph* $\mathcal{H}(q)$ (Definition 3.1).

In Section 3.1.1, we begin by showing that the resilience problem for two basic queries, the triangle query ($q_{\triangle}$) and the tripod query ($q_{\mathrm{T}}$) are both NP-complete. We then generalize these queries to a feature of hypergraphs that we call a *triad* (Definition 3.7), which is a set of 3 atoms that are connected in a special way in $\mathcal{H}(q)$. We then prove that if $\mathcal{H}(q)$ contains a triad, then $\texttt{RES}(q)$ is NP-complete. Conversely, we show in Section 3.1.2 that if $\mathcal{H}(q)$ does not contain any triad, then $\texttt{RES}(q)$ is in PTIME. We prove this by showing how to transform a triad-free sj-free CQ into a linear query $q'$ of equivalent complexity. The resilience of linear queries can be computed efficiently in polynomial time using a reduction to network flow that was proposed by previous work [35]. Our dichotomy theorem for the resilience of sj-free CQ then follows (Theorem 3.24).

### 3.1.1 Triads make resilience hard

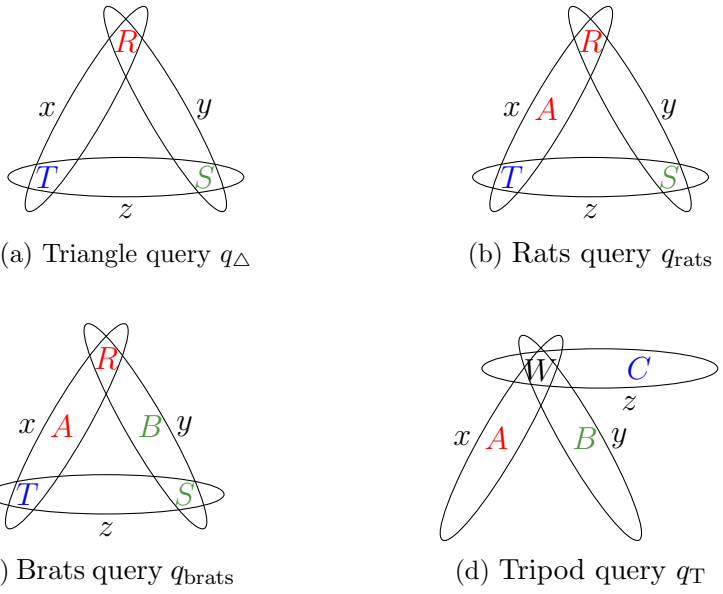
In this section, we present our first main contribution which is the novel concept of triads (Definition 3.7): we prove that if the dual hypergraph of a query $q$ contains a triad, then the resilience problem $\texttt{RES}(q)$ is NP-complete (Lemma 3.10).

Triads were inspired by a set of queries previously studied in causal responsibility [35], but the particular structure was not discovered, nor characterized in prior work. In order to lead to the concept of triads, we have to review some basic results and queries that were initially introduced in causal responsibility [35]. While these intermediate results seem on the surface similar to those that appeared in prior work, their proofs follow different reductions that are important in understanding the proof of our main result in Lemma 3.10.

We first define the (dual) hypergraph $\mathcal{H}(q)$ of query $q$. The hypergraph of a query $q$ is usually defined with its vertices being the variables of $q$ and the hyperedges being the atoms [1]. In this work we use only the dual hypergraph:

**Definition 3.1** (Dual Hypergraph $\mathcal{H}(q)$)**.** *Let $q :- A_1, \ldots, A_m$ be a sj-free CQ. Its* dual hypergraph $\mathcal{H}(q)$ *has vertex set $V = \{A_1, \ldots, A_m\}$. Each variable $x_i \in \texttt{var}(q)$ determines the hyperedge consisting of all those atoms in which $x_i$ occurs: $e_i = \{A_j \mid x_i \in \texttt{var}(A_j)\}$.*

For example, Fig. 3.1 shows the dual hypergraphs of four important queries defined in Example 3.2. We only consider dual hypergraphs, so we use the shorter term "hypergraph" from now on. In fact we will think of a query and its hypergraph as one and the same thing. Furthermore, when we discuss *vertices*, *edges* and *paths*, we are referring to those objects in the hypergraph of the query under consideration. Thus, a *vertex* is an *atom*, an *edge* is a *variable*, and a *path* is an alternating sequence of vertices and edges, $A_1, x_1, A_2, x_2, \ldots, A_{n-1}, x_{n-1}, A_n$, such that for all $i$, $x_i \in \texttt{var}(A_i) \cap \texttt{var}(A_{i+1})$, i.e., the hyperedge $x_i$ joins vertices $A_i$ and $A_{i+1}$. We explicitly list the




(a) Triangle query $q_{\triangle}$

(b) Rats query $q_{\text{rats}}$

(c) Brats query $q_{\text{brats}}$

(d) Tripod query $q_{\text{T}}$

Figure 3.1: Example 3.2: The hypergraphs of queries $q_{\triangle}$, $q_{\text{rats}}$, $q_{\text{brats}}$, $q_{\text{T}}$. $\{R, S, T\}$ is a triad of $q_{\triangle}$; {A,B,C} is a triad of $q_{\text{T}}$.

hyperedges in the path, because more than one hyperedge may join the same pair of vertices.

Furthermore, since disconnected components of a query have no effect on each other, each of several disconnected components can be considered independently. We will thus assume throughout that *all queries are connected.* Similarly, WLOG we assume no query contains two atoms with exactly the same set of variables, otherwise, if two atoms $A, B$ appear in $q$ with the identical set of variables, we can replace $A$ by $A \cap B$ and delete $B$.

**Example 3.2** (Important queries)**.** *Before we precisely define what a triad is, we identify two hard queries, $q_{\triangle}, q_{\text{T}}$ and two related queries, $q_{rats}, q_{brats}$ (see Fig. 3.1 for drawings of their hypergraphs).*

$$q_{\triangle} :- R(x,y), S(y,z), T(z,x) \qquad \textit{(Triangle)}$$

$$q_{rats} :- A(x), R(x,y), S(y,z), T(z,x) \qquad \textit{(Rats)}$$

$$q_{brats} :- A(x), R(x,y), B(y), S(y,z), T(z,x) \qquad \textit{(Brats)}$$

$$q_{\mathrm{T}} :- A(x), B(y), C(z), W(x,y,z) \qquad \textit{(Tripod)}$$

We now prove that $q_{\triangle}$ and $q_{\mathrm{T}}$ are both hard, i.e., their resilience problems are NP-complete. This will lead us to the definition of triads, the hypergraph property that implies hardness. Later, we will see that $q_{\mathrm{brats}}$ is easy for both resilience and responsibility. *However, counter to our initial intuition, $q_{rats}$ is easy for resilience but hard for responsibility.*

**Proposition 3.3** (Triangle $q_{\triangle}$ is hard)**.** $\texttt{RES}(q_{\triangle})$ *and* $\texttt{RSP}(q_{\triangle})$ *are* NP*-complete.*

*Proof.* We reduce 3SAT to $\texttt{RES}(q_{\triangle})$. It will then follow that $\texttt{RES}(q_{\triangle})$ is NP-complete, and thus so is $\texttt{RSP}(q_{\triangle})$ by Lemma 2.12. Let $\psi$ be a 3CNF formula with $n$ variables $v_1, \ldots, v_n$ and $m$ clauses $C_0, \ldots, C_{m-1}$. Our reduction will map any such $\psi$ to a pair $(D_\psi, k_\psi)$ where $D_\psi$ is a database satisfying $q_{\triangle}$, and

$$\psi \in \mathrm{3SAT} \quad \Leftrightarrow \quad (D_\psi, k_\psi) \in \texttt{RES}(q_{\triangle}) \tag{3.4}$$

In our construction, if $\psi \in \mathrm{3SAT}$, then the size of each minimum contingency set for $q_{\triangle}$ in $D_\psi$ will be $k_\psi = 6mn$, whereas if $\psi \notin \mathrm{3SAT}$, then the size of all contingency sets for $q_{\triangle}$ in $D_\psi$ will be greater than $k_\psi$.

Notice that $D_\psi \models q_{\triangle}$ iff it contains three tuples $R(a,b)$, $S(b,c)$, $T(c,a)$ that together form a witness. We visualize $R(a,b)$ as a red edge, $S(b,c)$ as a green edge and $T(c,a)$ as a blue edge. In other words, each witness $(a,b,c)$ for $D_\psi \models q_{\triangle}$ forms an RGB triangle. (Notice that the edge direction $a \to b$ drawn in Figures 3.2, 3.3 and 3.4 corresponds to the variable order in $R$, and analogously for $S$ and $T$.) The job of a contingency set for $q_{\triangle}$ is to remove all RGB triangles.

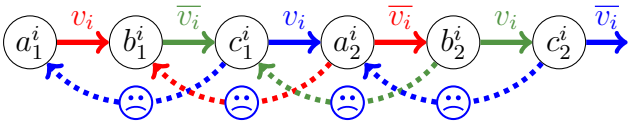



Figure 3.2: A six-node segment of the gadget $G_i$ in the hardness proof for $q_\triangle$: A minimum contingency set chooses either all the solid lines marked $v_i$, or all the solid lines marked $\overline{v_i}$. The dotted lines are sad because each of them is only part of one single RGB triangle, thus they are never chosen.

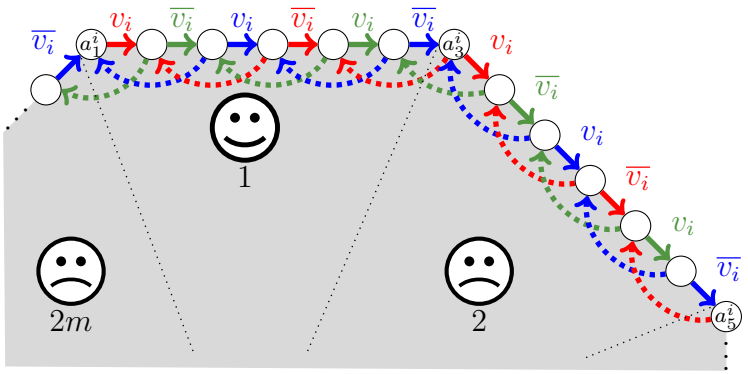



Figure 3.3: Each gadget $G_i$ in the hardness proof for $q_\triangle$ is a cycle containing $2m$ six-node segments and a total of $12m$ RGB triangles. They can all be eliminated by removing the $6m$ edges marked $v_i$ or the $6m$ edges marked $\overline{v_i}$. The even numbered segments are sad because they are never used for connecting different gadgets (corresponding to clauses that use several variables); they only separate the odd ones, thus preventing spurious triangles.

$D_\psi$ contains one circular gadget $G_i$ for each variable $v_i$. The circle consists of $12m$ solid edges, half of them marked $v_i$ and the other half marked $\mathbf{v_i}$ (see Figures 3.2, 3.3). Note that there are $12m$ RGB triangles and they can be minimally broken by choosing the $6m$ $v_i$ edges or the $6m$ $\mathbf{v_i}$ edges. Any other way would require more edges removed. Thus, each minimum contingency set for $D_\psi$ corresponds to a truth assignment to the variables of $\psi$. And there will be a minimum contingency set of size $k_\psi = 6mn$ iff $\psi \in$ 3SAT.

We complete the construction of $D_\psi$ by adding one RGB triangle for each clause $C_j$. For example, suppose $C_j = v_1 \vee \overline{v_2} \vee v_3$. The RGB triangle we add consists of a

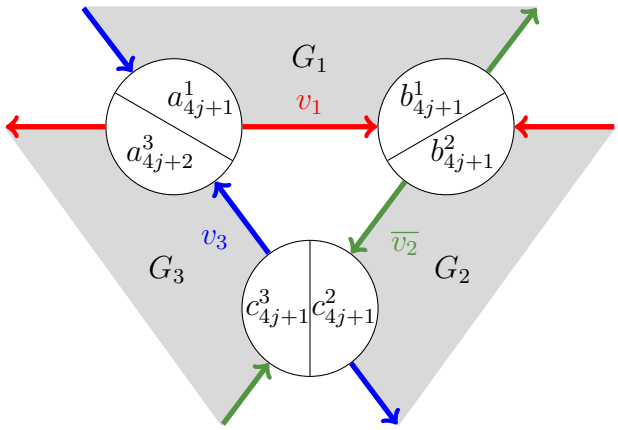



Figure 3.4: For clause $C_j = (v_1 \vee \overline{v_2} \vee v_3)$ in the hardness proof for $q_\triangle$, we identify vertices $b^1_{4j+1} \in G_1$ with $b^2_{4j+1} \in G_2$; $c^2_{4j+1} \in G_2$ with $c^3_{4j+1} \in G_3$ and $a^3_{4j+2} \in G_3$ with $a^1_{4j+1} \in G_1$. This RGB triangle will be deleted iff the chosen variable assignment satisfies $C_j$.

red edge marked $v_1$, a green edge marked $\overline{v_2}$ and a blue edge marked $v_3$ (see Fig. 3.4). Note that if the chosen assignment satisfies $C_j$, then all $v_1$ edges are removed, or all $\overline{v_2}$ edges are removed, or all $v_3$ edges are removed. Thus the $C_j$ triangle is automatically removed.

How do we create $C_j$'s RGB triangle? Remember that we have chosen $G_i$ to contain 2 segments for each clause. We use segment $2j+1$ of $G_i$ to produce the $v_i$ or $\overline{v_i}$ used in $C_j$'s triangle. The even numbered segments are not used: they serve as buffers to prevent spurious RGB triangles from being created. In Fig. 3.3, we mark these even segments with frowns: they are sad because they are never used.

More precisely, the red $v_1$-edge from $G_1$ is $(a^1_{4j+1}, b^1_{4j+1})$, the green $\overline{v_2}$-edge from $G_2$ is $(b^2_{4j+1}, c^2_{4j+1})$, and the blue $v_3$-edge from $G_3$ is $(c^3_{4j+1}, a^3_{4j+2})$ (see Fig. 3.4).

Now to make this an RGB triangle in $D_\psi$, we identify the two $a$-vertices, the two $b$ vertices and the two $c$ vertices. In other words, $G_1$'s $a$-vertex $a^1_{4j+1}$ is equal to $G_3$'s $a$-vertex $a^3_{4j+2}$, i.e., they are the same element of the domain of $D_\psi$. We have thus constructed $C_j$'s RGB triangle (see Fig. 3.4).

The key idea is that these identifications can only create this single new RGB triangle because there is no other way to get back to $G_1$ from $G_2$ in two steps. All other identifications involve different segments and so are at least six steps away. Recall that this is the reason why the even-numbered segments in the $G_i$'s are not used: this ensures that no spurious RGB triangles are created. Thus, as desired, Eq. 3.4 holds and we have reduced 3SAT to $\texttt{RES}(q_\triangle)$. □

We next show that the tripod query $q_{\mathrm{T}}$ is also hard. We do this by reducing the triangle to the tripod.

**Proposition 3.5** (Tripod $q_{\mathrm{T}}$ is hard)**.** $\texttt{RES}(q_{\mathrm{T}})$ *and* $\texttt{RSP}(q_{\mathrm{T}})$ *are* NP*-complete.*

*Proof.* First observe that in $q_{\mathrm{T}}$, $\texttt{var}(A)$ is a subset of $\texttt{var}(W)$. We say that $A$ *dominates* $W$ (Definition 3.8). It thus follows that when computing the resilience of $q_{\mathrm{T}}$, a tuple $W(a,b,c)$ is never needed in a minimum contingency set because it could always be replaced at least as efficiently by the tuple $A(a)$. It follows that we may assume that $W$ is exogenous, i.e., $\texttt{RES}(q_{\mathrm{T}}) \equiv \texttt{RES}(q'_{\mathrm{T}})$ where $q'_{\mathrm{T}} :- A(x), B(y), C(z), W^{\mathrm{x}}(x,y,z)$ (Prop. 3.9).

We now reduce $\texttt{RES}(q_\triangle)$ to $\texttt{RES}(q'_{\mathrm{T}})$. It will then follow that $\texttt{RES}(q_{\mathrm{T}})$ is NP-complete, and thus so is $\texttt{RSP}(q_{\mathrm{T}})$ by Lemma 2.12. Let $(D,k)$ be an instance of $\texttt{RES}(q_\triangle)$. We construct an instance $(D',k)$ of $\texttt{RES}(q'_{\mathrm{T}})$ by constructing relations $A, B, C$ as copies of $R, S, T$ from $D$. Define $D' = (A, B, C, W^{\mathrm{x}})$ as follows:

$$
\begin{aligned}
A &= \big\{\langle ab\rangle \;\big|\; R(a,b) \in D\big\} \\
B &= \big\{\langle bc\rangle \;\big|\; S(b,c) \in D\big\} \\
C &= \big\{\langle ca\rangle \;\big|\; T(c,a) \in D\big\} \\
W^{\mathrm{x}} &= \big\{(\langle ab\rangle, \langle bc\rangle, \langle ca\rangle) \;\big|\; a,b,c \in \mathrm{dom}(D)\big\}
\end{aligned}
$$

Here, $\texttt{dom}(D)$ is the set of domain elements of $D$ and $\langle ab\rangle$ stands for a new unique domain value resulting from the concatenation of domain values $a$ and $b$.

Observe that there is a 1:1 correspondence between the witnesses of $D \models q_\triangle$ and the witnesses of $D' \models q'_{\mathrm{T}}$. For example, $(a, b, c)$ is a witness that $D \models q_\triangle$ iff tuples $R(a, b), S(b, c), T(c, a)$ occur in $D$. This holds iff $(\langle ab\rangle, \langle bc\rangle, \langle ca\rangle)$ is a witness that $D' \models q'_{\mathrm{T}}$, i.e., the tuples $A(\langle ab\rangle), B(\langle bc\rangle), C(\langle ca\rangle), W(\langle ab\rangle, \langle bc\rangle, \langle ca\rangle)$ occur in $D'$. Thus, every contingency set for $q_\triangle$ in $D$ corresponds to a contingency set of the same size for $q'_{\mathrm{T}}$ in $D'$. It follows that $(D, k) \in \texttt{RES}(q_\triangle) \Leftrightarrow (D', k) \in \texttt{RES}(q'_{\mathrm{T}})$. □

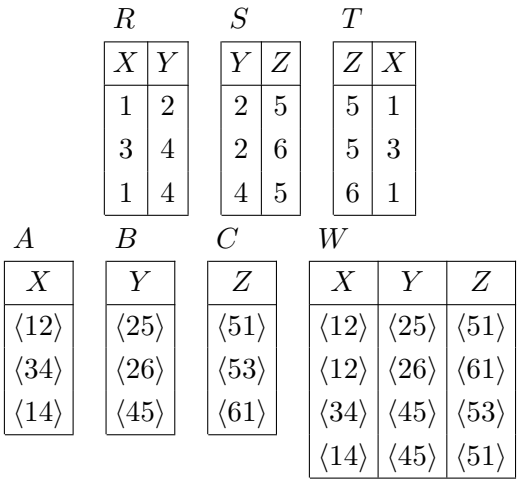
Understanding the reduction $q_\triangle \leq q_{\mathrm{T}}$ is useful for understanding the proof of our main result. Here is an example of the reduction to help understand what is behind the general case.

**Example 3.6.** *The reduction $q_\triangle \leq q_{\mathrm{T}}$ maps any pair $(D, k)$ to a pair $(D', k')$ such that $(D, k) \in \texttt{RES}(q_\triangle)$ iff $(D', k') \in \texttt{RES}(q_{\mathrm{T}})$. We illustrate this mapping with an example: Fig. 3.5.*

*The mapping produces the tables $A, B, C, W$ from the tables $R, S, T$. For each tuple $R(a, b) \in D$, we create a new value $\langle ab\rangle$ and put $A(\langle ab\rangle)$ into $D'$. Similarly, $S(b, c) \in D$ generates $B(\langle bc\rangle) \in D'$ and $T(c, a) \in D$ generates $C(\langle ca\rangle \in D'$. Finally each witness $(a, b, c)$ that $D \models q_\triangle$ is mapped to the tuple $W(\langle ab\rangle, \langle bc\rangle, \langle ca\rangle) \in D'$.*

*By the way, a minimum contingency set, $\Gamma$, for $q_{\mathrm{T}}$ never needs to have a tuple from $W$ because the effect of any tuple $W(i, j, k) \in \Gamma$ would be just to remove that witness $(i, j, k)$ of $D' \models q_{\mathrm{T}}$. This can be more efficiently done instead by putting any one of $A(i)$, $B(j)$, or $C(k)$ into $\Gamma$. We will see that $W$ is (dominated) by $A, B, C$ (Definition 3.8).*

*It is easy to see that this reduction gives a 1:1 correspondence between minimum contingency sets for $D$ and those for $D'$. For example, the minimum contingency set $\{R(1, 2), S(4, 5)\}$ for $D$ corresponds to the minimum contingency set $\{A(\langle 12\rangle), B(\{45\})\}$ for $D'$.*



| $R$ | |
|---|---|
| $X$ | $Y$ |
| 1 | 2 |
| 3 | 4 |
| 1 | 4 |

| $S$ | |
|---|---|
| $Y$ | $Z$ |
| 2 | 5 |
| 2 | 6 |
| 4 | 5 |

| $T$ | |
|---|---|
| $Z$ | $X$ |
| 5 | 1 |
| 5 | 3 |
| 6 | 1 |

| $A$ |
|---|
| $X$ |
| $\langle 12 \rangle$ |
| $\langle 34 \rangle$ |
| $\langle 14 \rangle$ |

| $B$ |
|---|
| $Y$ |
| $\langle 25 \rangle$ |
| $\langle 26 \rangle$ |
| $\langle 45 \rangle$ |

| $C$ |
|---|
| $Z$ |
| $\langle 51 \rangle$ |
| $\langle 53 \rangle$ |
| $\langle 61 \rangle$ |

| $W$ | | |
|---|---|---|
| $X$ | $Y$ | $Z$ |
| $\langle 12 \rangle$ | $\langle 25 \rangle$ | $\langle 51 \rangle$ |
| $\langle 12 \rangle$ | $\langle 26 \rangle$ | $\langle 61 \rangle$ |
| $\langle 34 \rangle$ | $\langle 45 \rangle$ | $\langle 53 \rangle$ |
| $\langle 14 \rangle$ | $\langle 45 \rangle$ | $\langle 51 \rangle$ |

Figure 3.5: Database $D$ and database $D'$ defined by the reduction.

While $q_{\triangle}$ and $q_{\mathrm{T}}$ appear to be very different, they share a key common structural property, which we define next.

**Definition 3.7** (triad)**.** *A* triad *is a set of three endogenous atoms, $\mathcal{T} = \{S_0, S_1, S_2\}$ such that for every pair $i, j$, there is a path from $S_i$ to $S_j$ that uses no variable occurring in the other atom of $\mathcal{T}$.*

Intuitively, a triad is a triple of points with robust connectivity. Observe that atoms $R, S, T$ form a triad in $q_{\triangle}$ and atoms $A, B, C$ form a triad in $q_{\mathrm{T}}$ (see Fig. 3.1). For example, there is a path from $R$ to $S$ in $q_{\triangle}$ (across hyperedge $y$) that uses only variables (here $y$) that are not contained in the other atom (here $y \notin \mathtt{var}(T)$).

A triad is composed of endogenous atoms. Some atoms such as $W$ in $q_T$ are given as endogenous, but are not needed in contingency sets. We will simplify the query by making all such atoms exogenous.

**Definition 3.8** (Domination)**.** *If a query $q$ has endogenous atoms $A, B$ such that $\mathtt{var}(A) \subset \mathtt{var}(B)$, then we say that $A$* dominates *$B$.*[1]

[1]Recall that for $A \neq B$, we never have that $\mathtt{var}(A) = \mathtt{var}(B)$.

We already saw an example in Example 3.6: in $q_{\mathrm{T}}$, each of the atoms $A, B, C$ dominates $W$. The following proposition was proved in [35].

**Proposition 3.9** (Domination for resilience)**.** *Let $q$ be an sj-free CQ and $q'$ the query resulting from labeling some dominated atoms as exogenous. Then $\mathtt{RES}(q) \equiv \mathtt{RES}(q')$.*

*Proof.* Let $\Gamma$ be a minimum contingency set of $q$ in $D$. Suppose that atom $A$ dominates atom $B$ but there is some tuple $B(\mathbf{t}) \in \Gamma$. Let $\mathbf{p}$ be the projection of $\mathbf{t}$ onto $\mathtt{var}(A)$. Then we can replace $B(\mathbf{t})$ by $A(\mathbf{p})$ and we remove at least as many witnesses that $D \models q$. It follows, as desired, that the complexity of $\mathtt{RES}(q)$ is unchanged if $B$ is exogenous, i.e., $\mathtt{RES}(q) \equiv \mathtt{RES}(q')$. □

When studying resilience, we follow the convention that *all dominated atoms should become exogenous*, and we consider that the normal form of a query. For example, $A$ dominates $R$ and $T$ in the query $q_{\mathrm{rats}}$, and $B$ dominates $R$ and $S$ in the query $q_{\mathrm{brats}}$. We thus transform the queries so that the dominated atoms are exogenous. Exogenous atoms have the superscript "x".

$$q'_{\mathrm{rats}} :- A(x), R^{\mathrm{x}}(x, y), S(y, z), T^{\mathrm{x}}(z, x)$$
$$q'_{\mathrm{brats}} :- A(x), R^{\mathrm{x}}(x, y), B(y), S^{\mathrm{x}}(y, z), T^{\mathrm{x}}(z, x)$$

By Proposition 3.9, $\mathtt{RES}(q_{\mathrm{rats}}) \equiv \mathtt{RES}(q'_{\mathrm{rats}})$ and $\mathtt{RES}(q_{\mathrm{brats}}) \equiv \mathtt{RES}(q'_{\mathrm{brats}})$. We now state our first main result.

**Lemma 3.10** (Triads make $\mathtt{RES}(q)$ hard)**.** *Let $q$ be an sj-free CQ where all dominated atoms are exogenous. If $q$ has a triad, then $\mathtt{RES}(q)$ is* NP*-complete.*

*Proof.* Let $q$ be a query with triad $\mathcal{T} = \{S_0, S_1, S_2\}$. We build a reduction from $\mathtt{RES}(q_{\triangle})$ to $\mathtt{RES}(q)$. Given any $D$ that satisfies $q_{\triangle}$ we will produce a database $D'$ that satisfies $q$ such that for all $k$:

$$(D,k) \in \texttt{RES}(q_\triangle) \quad \Leftrightarrow \quad (D',k) \in \texttt{RES}(q) \tag{3.11}$$

We will assume that no variable is shared by all three elements of $\mathcal{T}$ (we can ignore any such variable by setting it to a constant). Our proof splits into two cases:

*Case 1*: $\texttt{var}(S_0), \texttt{var}(S_1), \texttt{var}(S_2)$ are pairwise disjoint: Our reduction is similar to the reduction from $q_\triangle$ to $q_{\mathrm{T}}$ (Prop. 3.5).

We first define the triad relations in $D'$:

$$\begin{aligned} S_0 &= \big\{(\langle ab\rangle, \ldots, \langle ab\rangle) \;\big|\; R(a,b) \in D\big\} \\ S_1 &= \big\{(\langle bc\rangle, \ldots, \langle bc\rangle) \;\big|\; S(b,c) \in D\big\} \\ S_2 &= \big\{(\langle ca\rangle, \ldots, \langle ca\rangle) \;\big|\; T(c,a) \in D\big\}. \end{aligned} \tag{3.12}$$

Thus, each tuple of, for example, $S_0$ consists of identical entries with value $\langle ab\rangle$ for each pair $R(a,b) \in D$. Thus, $S_0, S_1, S_2$ mirror $R, S, T$, respectively.

To define all the relations corresponding to the other atoms $A_i$ of $D'$, we first partition the variables of $q$ into 4 disjoint sets: $\texttt{var}(q) = \texttt{var}(S_0) \cup \texttt{var}(S_1) \cup \texttt{var}(S_2) \cup V_3$. Now for each atom $A_i$, arrange its variables in these four groups. Then define the relation $R'_i$ of $D'$ corresponding to atom $A_i$ as follows

$$R'_i = \big\{(\langle ab\rangle; \langle bc\rangle; \langle ca\rangle; \langle abc\rangle) \;\big|\; D \models q_\triangle(a,b,c)\big\} \tag{3.13}$$

For example, all the variables $v \in \texttt{var}(S_0)$ are assigned the value $\langle ab\rangle$ and all the variables $v \in V_3$ are assigned $\langle abc\rangle$.

By the definition of triad, there is a path from $S_0$ to $S_1$ not using any edges (variables) from $\texttt{var}(S_2)$. Thus, any witness of $D' \models q$ that includes occurrences of $\langle ab\rangle$ and $\langle b'c'\rangle$ must have $b = b'$.

Similarly, a path from $S_1$ to $S_2$ guarantees that $c$ is preserved and a path from $S_2$ to $S_0$ guarantees that $a$ is preserved. It follows that the witnesses that $D' \models q$ are essentially identical to the witnesses that $D \models q_\triangle(x, y, z)$ (see Fig. 3.6).[2]

Furthermore, any minimum contingency set only needs tuples from $S_0, S_1$ or $S_2$. Thus the sizes of minimum contingency sets are preserved, i.e., Eq. 3.11 holds, as desired. Thus $\texttt{RES}(q)$ is NP-complete.

*Case 2*: $\texttt{var}(S_i) \cap \texttt{var}(S_j) \neq \emptyset$ for some $i \neq j$: We generalize the construction from Case 1 as follows. Partition $\texttt{var}(S_i)$ into those unshared, those shared with $S_{i-1}$, and those shared with $S_{i+1}$ (addition here is mod 3).

We then assign the relations of the triad as follows:

$$S_0 = \left\{ (\langle ab \rangle; a; b) \;\middle|\; R(a, b) \in D \right\}$$
$$S_1 = \left\{ (\langle bc \rangle; b; c) \;\middle|\; S(b, c) \in D \right\}$$
$$S_2 = \left\{ (\langle ca \rangle; c; a) \;\middle|\; T(c, a) \in D \right\}$$

Since none of the $S_i$'s is dominated, both $a$ and $b$ occur in each tuple of $S_0$, both of $b$ and $c$ in each tuple of $S_1$ and both of $c$ and $a$ in each tuple of $S_2$. Thus, as in Case 1, $S_0, S_1, S_2$ capture $R, S, T$, respectively. The key ideas is now that we partition all the variables $\texttt{var}(q)$ into 7 sets according to their respective appearance in each of the 3 tables. For each assignment of $x, y, z$ to values $a, b, c$ in $D$, we will then make assignments to the variables according to their partition:

---

[2]More precisely, if $(a, b, c)$ is a witness that $D \models q_\triangle$, then $(\langle ab \rangle, \langle bc \rangle, \langle ca \rangle, \langle abc \rangle, a, b, c)$ is a witness that $D' \models q$, with the variables partitioned according to Eq. 3.14, and these are the only possible such witnesses.

| set name | variable partition | assignment |
|---|---|---|
| $V_0$ | $\texttt{var}(S_0) - (\texttt{var}(S_1) \cup \texttt{var}(S_2))$ | $\langle ab \rangle$ |
| $V_1$ | $\texttt{var}(S_1) - (\texttt{var}(S_0) \cup \texttt{var}(S_2))$ | $\langle bc \rangle$ |
| $V_2$ | $\texttt{var}(S_2) - (\texttt{var}(S_0) \cup \texttt{var}(S_1))$ | $\langle ca \rangle$ |
| $V_3$ | $\texttt{var}(q) - (\texttt{var}(S_0) \cup \texttt{var}(S_1) \cup \texttt{var}(S_2))$ | $\langle abc \rangle$ |
| $V_4$ | $\texttt{var}(S_2) \cap \texttt{var}(S_0)$ | $a$ |
| $V_5$ | $\texttt{var}(S_0) \cap \texttt{var}(S_1)$ | $b$ |
| $V_6$ | $\texttt{var}(S_1) \cap \texttt{var}(S_2)$ | $c$ |

(3.14)

We then define the relations in $D'$ corresponding to each of the other atoms $A$ of $q$ to be the following set of tuples, where the only difference is which of the 7 members of the partition of variables occurs in $\texttt{var}(A)$.

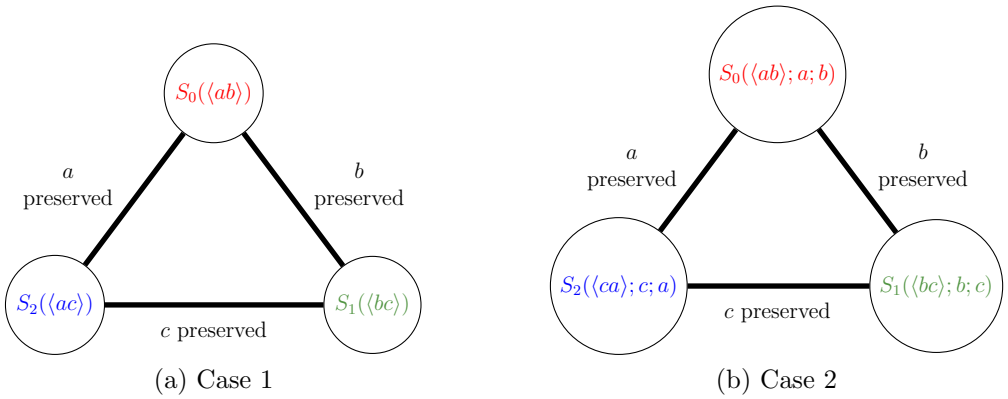
$$\left\{(\langle ab \rangle; \langle bc \rangle; \langle ca \rangle; \langle abc \rangle; a; b; c) \;\middle|\; D \models q_{\triangle}(a, b, c)\right\} \tag{3.15}$$

By the definition of a triad, there is a path from $S_0$ to $S_1$ not using any edges (variables) from $S_2$. Thus, "$b$" is always present (see Eq. 3.14). Thus, any witness including occurrences of some of $\langle ab \rangle, b', \langle b''c \rangle$ must have $b = b' = b''$. Thus, as in Case 1, the witnesses of $D' \models q$ are essentially identical to the witnesses of $D \models q_{\triangle}$ and we have reduced $\texttt{RES}(q_{\triangle})$ to $\texttt{RES}(q)$ (see Fig. 3.6). □

### 3.1.2 Polynomial algorithm for linear queries

We just showed that resilience for queries with triads is NP-complete. Next we will prove a strong converse: resilience for triad-free queries is in PTIME. We start by defining a class of queries for which resilience is known to be in PTIME.

**Definition 3.16** (Linear Query)**.** *A query $q$ is* linear *if its atoms may be arranged in a linear order such that each variable occurs in a contiguous sequence of atoms.*




(a) Case 1 (b) Case 2

Figure 3.6: Reduction from $\texttt{RES}(q_\triangle)$ to $\texttt{RES}(q)$ when $q$ contains a triad $\{S_0, S_1, S_2\}$ in the proof of Lemma 3.10.

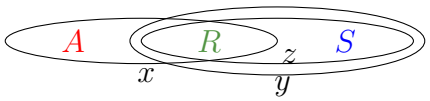



Figure 3.7: Definition 3.16: Linear query $q :- A(x), R(x, y, z), S(y, z)$

**Example 3.17** (Linear Query)**.** *Geometrically, a query is linear if all of the vertices of its hypergraph can be drawn along a straight line and all of its hyperedges can be drawn as convex regions. For example, the following query is linear,* $q :- A(x), R(x, y, z), S(y, z)$ *(see Fig. 3.7).*

The responsibility of linear queries is known to be in PTIME [35] and thus by Lemma 2.12, resilience of linear queries is in PTIME as well.

**Fact 3.18** (Linear queries in PTIME [35])**.** *For any linear sj-free CQ* $q$, $\texttt{RSP}(q)$ *(and thus also* $\texttt{RES}(q)$*) are in* PTIME.

The proof of Fact 3.18 is that $\texttt{RES}(q)$ may be computed in a natural way using network flow. The same is true for computing the responsibility of tuple $\mathbf{t}$ for $D \models q$. In the latter case, we consider each possible extension, $\mathbf{e}$ of $\mathbf{t}$ that is a witness of $D \models q$, and use network flow to compute the minimum size contingency set $\Gamma$ for $\mathbf{t}$ such that $\mathbf{e}$ remains a witness of $D - \Gamma \models q$. The responsibility of $\mathbf{t}$ for $D \models q$ is the

minimum over all such extensions $\mathbf{e}$ of the size of the minimum contingency set that preserves $\mathbf{e}$.

If all queries without a triad were linear, then this would complete the dichotomy theorem for resilience. While this is not the case, we will show that *any triad-free query can be transformed into a query of equivalent complexity that is linear.*

Recall that when studying resilience, we make atoms which are dominated, exogenous (Proposition 3.9). This was done, for example, to the rats and brats queries to transform them into the $q'_{\text{rats}}$ and $q'_{\text{brats}}$ queries. Neither of $q'_{\text{rats}}$ or $q'_{\text{brats}}$ is linear. However they can be transformed to linear queries without changing their complexity via the following transformation from [35]:

**Definition 3.19** (Dissociation)**.** *Let $A^{\text{x}}$ be an exogenous atom in a query $q$, and $v \in \texttt{var}(q)$ a variable that does not occur in $A^{\text{x}}$. Let $q'$ be the same as $q$ except that we add $v$ to the arguments $A^{\text{x}}$. This transformation is called* dissociation*.*

**Example 3.20** (Dissociation)**.** *The above queries $q'_{rats}$ and $q'_{brats}$ have no triads but they are not linear. However, by applying certain dissociations, we obtain the following linear queries:*

$$q''_{rats} :- A(x), R^{\text{x}}(x,y,z), S(y,z), T^{\text{x}}(x,y,z)$$
$$q''_{brats} :- A(x), R^{\text{x}}(x,y,z), B(y), S^{\text{x}}(x,y,z), T^{\text{x}}(x,y,z)$$

*Note also that $q''_{rats}$ and $q''_{brats}$ have duplicate atoms which we finally delete, without affecting their complexity:*

$$q'''_{rats} :- A(x), R^{\text{x}}(x,y,z), S(y,z)$$
$$q'''_{brats} :- A(x), R^{\text{x}}(x,y,z), B(y)$$

The key fact is that *dissociation can increase, but never decrease the complexity of resilience or responsibility.* For example, query $q :- A(x), W_1^{\mathrm{x}}(x,y), B(y), W_2^{\mathrm{x}}(y,z), C(z)$ is linear, but by dissociating $W_1$ and $W_2$, we can transform it into $q_{\mathrm{T}}$.

**Lemma 3.21** (Dissociation increases complexity [35])**.** *If $q'$ can be obtained from $q$ through dissociation, then* $\texttt{RES}(q) \leq \texttt{RES}(q')$.

It follows from Lemma 3.21 that if $q$ can be dissociated into a linear query, then $\texttt{RES}(q)$ is in PTIME. In particular, the above dissociations of $q'_{\mathrm{rats}}$ and $q'_{\mathrm{brats}}$ prove that $\texttt{RES}(q'_{\mathrm{rats}})$ and $\texttt{RES}(q'_{\mathrm{brats}})$ are in PTIME. Thus, since the transformations from $q_{\mathrm{rats}}$ to $q'_{\mathrm{rats}}$ and $q_{\mathrm{brats}}$ to $q'_{\mathrm{brats}}$ preserve the complexity of resilience, we thus conclude that $\texttt{RES}(q_{\mathrm{rats}})$ and $\texttt{RES}(q_{\mathrm{brats}})$ are easy.

**Corollary 3.22.** $\texttt{RES}(q_{rats})$ *and* $\texttt{RES}(q_{brats})$ *are in* PTIME.

Now we are ready to show that $\texttt{RES}(q)$ is easy if $q$ is triad-free. We will show that for every triad-free query, we can linearize the endogenous atoms and use some dissociations to make the exogenous atoms fit into the same order.

**Lemma 3.23** (Queries without triads are easy)**.** *Let $q$ be an sj-free CQ that has no triad. Then* $\texttt{RES}(q)$ *is in* PTIME.

*Proof.* Let $q$ be a triad-free query. We prove by induction on the number of *endogenous* atoms in $q$ that we can transform it into a linear query by using dissociations. Since dissociations cannot decrease complexity (Lemma 3.21) and resilience is easy for linear queries (Fact 3.18), it follows that $\texttt{RES}(q)$ is in PTIME.

*Base case*: $q$ has fewer than three endogenous atoms. Consider $S_1, S_2$ the endogenous atoms of $q$. Using dissociation, we add all the variables to all the exogenous atoms. Thus all the exogenous atoms are identical and we can remove all but one, call it $E_1^{\mathrm{x}}$. The resulting query, $q'$, is linear with ordering $S_1, E_1^{\mathrm{x}}, S_2$. Thus $\texttt{RES}(q) \in$ PTIME.

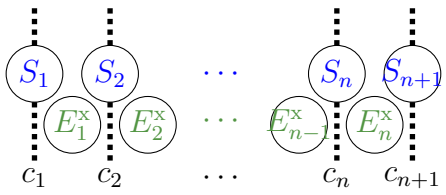



Figure 3.8: A walk along the endogenous atoms in the proof of Lemma 3.23. The cut $c_i$ results from removing all the variables (edges) from atom $S_i$.

*Inductive case*: assume true for triad-free queries with $n$ endogenous atoms. Let $q_{n+1}$ be triad-free and have $n+1$ endogenous atoms. We now describe a way to linearize these atoms. For each endogenous atom $S_i$, let $c_i$ be the cut of the hypergraph resulting from removing all the variables of $S_i$, i.e., all the hyperedges that touch $S_i$. These cuts are drawn as dotted vertical lines in Fig. 3.8.

Let $S_1$ and $S_2$ be two endogenous atoms and draw $S_2$ to the right of $S_1$. Now consider a third endogenous atom $S_3$. Since $q_{n+1}$ is connected and has no triads, there is a unique $i \in \{1, 2, 3\}$ such that the cut $c_i$ disconnects the two atoms in $\{S_1, S_2, S_3\} - \{S_i\}$.

Thus we must place $S_i$ between the other two. In other words, there is exactly one place that $S_3$ can be added to the figure: to the left of $S_1$ if $c_1$ separates $S_3$ from $S_2$; in between $S_1$ and $S_2$ if $c_3$ separates $S_1$ from $S_2$; or to the right of $S_2$ if $c_2$ separates $S_1$ from $S_3$.

For example, let $S_1(x, y)$ and $S_2(y, z)$ be the first two endogenous atoms. Let the third be $S_3(z, w)$ which shares a variable with $S_2$. Note that $c_3$ does not separate $S_1$ from $S_2$ and $c_1$ does not separate $S_2$ from $S_3$. Since $q_{n+1}$ has no triad, it must be the case that $c_2$ separates $S_1$ from $S_3$. Thus, the order in this case must be $S_1, S_2, S_3$.

Now add the remaining endogenous atoms one at a time. Since $q_{n+1}$ has no triad, by the above observation, there is exactly one place that each next endogenous atom may be placed. Finally once all the endogenous atoms have been placed, renumber them so left to right they are $S_1$, $S_2$, ..., $S_{n+1}$.

Define the query $q_n$ to be the result of removing all the variables in $\texttt{var}(S_{n+1}) - \texttt{var}(S_n)$ and removing all the atoms in which any of those removed variables occurred. In Fig. 3.8, this corresponds to removing everything to the right of $c_n$.

By our inductive hypothesis, there is a query $q'_n$ that is the result of doing some dissociations to $q_n$, and $q'_n$ is linear. Furthermore by our observation above, the ordering of the endogenous atoms remains $S_1, S_2, \ldots, S_n$.

Now, we form $q'_{n+1}$ by first adding back to $q_n$ all the variables and atoms that we removed. Note that we are thus adding back just one endogenous atom, $S_{n+1}$, together with zero or more exogenous atoms, all of which contain some variables in $\texttt{var}(S_{n+1}) - \texttt{var}(S_n)$. Finally, to all these exogenous atoms that we have just added back (if any), add all the variables in $\texttt{var}(S_n) \cup \texttt{var}(S_{n+1})$, together with any other variables occurring in any of these exogenous atoms. Thus all the newly re-added exogenous atoms are identical and we can combine them into one, call it, $E_n^{\text{x}}$. Note that $c_n$ still separates $E_n^{\text{x}}$ and $S_{n+1}$ from the rest of the hypergraph.

Thus, we have transformed $q_{n+1}$ to a linear query $q'_{n+1}$ such that $\texttt{RES}(q_{n+1}) \leq \texttt{RES}(q'_{n+1})$. Thus $\texttt{RES}(q_{n+1}) \in \mathsf{PTIME}$ as desired. □

### 3.1.3 Dichotomy for sj-free conjunctive queries

Combining Lemma 3.10 and Lemma 3.23 leads to our first dichotomy result on the complexity of resilience:

**Theorem 3.24** (Dichotomy of resilience)**.** *Let $q$ be an sj-free CQ and let $q'$ be the result of making all dominated atoms exogenous. If $q'$ has a triad, then* $\texttt{RES}(q)$ *is* NP*-complete, otherwise it is in* PTIME*.*

Note that it is easy to tell whether $q$ has a triad. Checking whether a given triple of atoms is a triad consists of three reachability problems – is there a path from $S_i$ to $S_j$ not using any of the edges in $\texttt{var}(S_k)$ – and is thus doable in linear time. An exhaustive search of all endogenous triples thus provides a PTIME algorithm:

**Corollary 3.25.** *We can check in polynomial time in the size of the query $q$ whether* $\texttt{RES}(q)$ *is* NP*-complete or* PTIME*.*

## 3.2 Functional dependencies

Functional dependencies (FDs), such as key constraints, restrict the set of allowable data instances. In this section, we characterize how these restrictions affect the complexity of resilience. We first show that FDs cannot increase the complexity of the resilience of a query (Proposition 3.26). Next we introduce a transformation of queries suggested by a given set of FDs call *induced rewrites* (Definition 3.29). We show that induced rewrites preserve the complexity of resilience (Lemma 3.30).

We call a query *closed* if all possible induced rewrites have been applied (Definition 3.29). We conjectured that induced rewrites capture the full power of FDs with respect to the complexity of resilience, in other words, the complexity of the resilience of a closed query is unchanged if we remove its FDs (Conjecture 3.32).

We prove that the complexity of resilience for closed queries that have triads is NP-complete (Lemma 3.33). On the other hand, even without its FDs, we know that a closed query that has no triads has an easy resilience problem (Lemma 3.23). We thus conclude that in the presence of FDs, the dichotomy – still determined by the presence or absence of triads, but now in the closure of the query – remains in force (Lemma 3.23). It follows as a corollary that Conjecture 3.32 holds.

### 3.2.1 FDs can only simplify resilience

We write $\texttt{RES}(q; \Phi)$ to refer to the resilience problem for query $q$, restricted to databases satisfying the set of FDs $\Phi$. Note that since we are always considering conjunctive queries, any particular FD either holds or does not hold on the whole query, so it is not necessary to mention which atom the FD is applied to.

First we observe that FDs cannot make the resilience problem harder:

**Proposition 3.26** (FDs do not increase complexity)**.** *Let $q$ be an sj-free CQ and $\Phi$ a set of functional dependencies. Then* $\texttt{RES}(q;\Phi) \leq \texttt{RES}(q)$.

*Proof.* The reduction is the identity function. Note that $\texttt{RES}(q;\Phi)$ is just the restriction of $\texttt{RES}(q)$ to databases satisfying $\Phi$. Thus, for all databases $D$ that satisfy $(q;\Phi)$,

$$(D,k) \in \texttt{RES}(q;\Phi) \quad \Leftrightarrow \quad (D,k) \in \texttt{RES}(q)$$

□

**Corollary 3.27** (Triad-free queries are still easy)**.** *If $q$ is an sj-free CQ that has no triad, and therefore* $\texttt{RES}(q)$ *is in* PTIME*, then* $\texttt{RES}(q;\Phi)$ *is also in* PTIME.

We next show that for some queries, FDs do in fact reduce the complexity of resilience. Recall that the tripod query, $q_{\text{T}}$ is hard (Proposition 3.5). However, $q_{\text{T}}$ becomes polynomial when we add the FD $\varphi = x \to y$.

**Proposition 3.28.** $\texttt{RES}(q_{\text{T}};\{x \to y\})$ *is in* PTIME.

We will prove Proposition 3.28 along the way, as we learn about the effect of FDs. Recall that the tripod query $q_{\text{T}}$ has the triad $\{A, B, C\}$. Notice that the FD $x \to y$ "disarms" this triad because $A$ and $B$ are no longer independent. More explicitly, once we know $x$, we also know $y$. Thus $\texttt{RES}(q_{\text{T}};\{x \to y\}) \equiv \texttt{RES}(r)$ where $r :- A'(x,y), B(y), C(z), W^{\text{x}}(x,y,z)$ (Lemma 3.30). Furthermore, since $B$ dominates $A'$ in $r$, $A'$ becomes exogenous: $r' :- A'^{\text{x}}(x,y), B(y), C(z), W^{\text{x}}(x,y,z)$. Query $r'$ has no triad and thus is easy.

### 3.2.2 Induced rewrites preserve complexity

We call the transformation $(q_{\mathrm{T}}; \{x \to y\}) \rightsquigarrow (r; \{x \to y\})$ an *induced rewrite*[3]. Induced rewrites are key to understanding the effect of FDs on the complexity of resilience.

**Definition 3.29** (induced rewrite: $\rightsquigarrow$, closed query)**.** *Given a set of functional dependencies $\Phi$ and a query $q$, we write $(q;\Phi) \rightsquigarrow (q';\Phi)$ to mean that $q'$ is the result of adding the dependent variable $u$ to some relation that contains all the determinant variables $\mathbf{v}$ for some $\mathbf{v} \to u \in \Phi$. We use $\overset{\star}{\rightsquigarrow}$ to indicate zero or more applications of $\rightsquigarrow$. If $(q;\Phi) \overset{\star}{\rightsquigarrow} (q^*;\Phi)$ and no more induced rewrites can be applied to $(q^*;\Phi)$, then we call $(q^*;\Phi)$ a* closed query *and we say that $(q^*;\Phi)$ is the* closure *of $(q;\Phi)$.*

This work began as an attempt to determine whether the dichotomy for responsibility of sj-free CQs [35] continues to hold in the presence of FDs. In studying the effect of FDs, we defined induced rewrites and proved that induced rewrites preserve the complexity of responsibility. We conjectured that once we have reached a closed query, all the effect of the FDs on the complexity of responsibility has been exhausted and thus there is no further change if we delete all the FDs. We were able to prove this conjecture for unary FDs, i.e., those of the form $v \to u$ where $v$ is a single variable.

However we had great difficulty proving this conjecture for all FDs. We studied the responsibility problem more carefully and found that responsibility is quite delicate. In particular, we discovered an error in Lemma 4.10 of [35], namely that Proposition 3.9 (in the present work) does not hold for responsibility.

We identified resilience as a better-behaved notion than responsibility and we characterized the complexity of resilience via triads. Once we had done that, we were able to use the notion of triads to prove our conjecture about closed queries and thus prove the dichotomy theorem for resilience in the presence of arbitrary FDs.

---

[3]Transformations of queries called *rewrites* were defined in [35]. An induced rewrite is a rewrite that is induced by an FD.

We first show that induced rewrites preserve the complexity of resilience.

**Lemma 3.30** (Induced rewrites preserve complexity)**.** *Let $q$ be a query, $\Phi$ a set of functional dependencies, and $q'$ the result of an induced rewrite, i.e., $(q;\Phi) \rightsquigarrow (q';\Phi)$. Then $\texttt{RES}(q';\Phi) \equiv \texttt{RES}(q;\Phi)$.*

*Proof.* Let the change from $q$ to $q'$ be the transformation of the atom $B$ to the new atom $B'$ caused by adding variable $u$ to $B$ where $(\mathbf{v} \to u) \in \Phi$ and $\mathbf{v} \subseteq \texttt{var}(B)$.

(a) $\texttt{RES}(q';\Phi) \leq \texttt{RES}(q;\Phi)$: Suppose we are given $(D', k)$ where $D'$ satisfies $\Phi$. Let $D$ be the result of projecting out the $u$ entry from $B'$. Note that $D$ still satisfies $\Phi$. Furthermore, the set of witnesses that $D \models q$ is identical to the set of witnesses that $D' \models q'$ and the sizes of all minimum contingency sets are unchanged. This is because the effect of the tuple $B(\mathbf{t})$ in a contingency set in $D$ is identical to the effect of the tuple $B'(\mathbf{t}')$ in the corresponding contingency set in $D'$, where $\mathbf{t}'$ is the result of adding to $\mathbf{t}$ the unique $u$-attribute which is determined by the $\mathbf{v}$-attributes of $\mathbf{t}$. Thus the map $(D', k) \mapsto (D, k)$ is a reduction of $\texttt{RES}(q';\Phi)$ to $\texttt{RES}(q;\Phi)$.

(b) $\texttt{RES}(q;\Phi) \leq \texttt{RES}(q';\Phi)$. We are given $(D, k)$ where $D$ satisfies $\Phi$. Let $B'$ be the set of tuples resulting from adding to each tuple $\mathbf{t}$ from $B$, the uniquely determined $u$-attribute, $c$. In symbols, $B' =$

$$\left\{(\mathbf{t}, c) \;\middle|\; B(\mathbf{t}) \in D \;\wedge\; \exists \mathbf{s} \in D\,(\pi_{\mathbf{v}}(\mathbf{s}) = \pi_{\mathbf{v}}(\mathbf{t}) \wedge c = \pi_u(\mathbf{s}))\right\}$$

For the same reason as above, the witnesses of $q'$ in $D'$ are the same as the witnesses of $q$ in $D$ and the sizes of all minimum contingency sets are unchanged. Thus the map $(D, k) \mapsto (D', k)$ is a reduction of $\texttt{RES}(q;\Phi)$ to $\texttt{RES}(q';\Phi)$.

□

It follows immediately that applying any set of induced rewrites preserves the complexity of resilience:

**Corollary 3.31.** *If* $(q;\Phi) \overset{\star}{\rightsquigarrow} (q';\Phi)$*, then* $\texttt{RES}(q';\Phi) \equiv \texttt{RES}(q;\Phi)$.

### 3.2.3 For closed queries, FDs are superfluous

Recall that our current goal is to determine whether the dichotomy of the complexity of resilience remains true in the presence of FDs. The following is a natural conjecture which would given an affirmative answer to this question.

**Conjecture 3.32** (Induced rewrites suffice)**.** *Let* $(q^*;\Phi)$ *be a closed query, i.e., it is closed under induced rewrites. Then* $\texttt{RES}(q^*;\Phi) \equiv \texttt{RES}(q^*)$.

It is fairly easy to see that Conjecture 3.32 holds when all the FDs in $\Phi$ are unary, i.e., of the form $v \to u$, with $u$ a single variable. However we were stumped about how to prove this for general FDs. This lead to our more careful analysis of the complexity of responsibility, our definition of resilience, and our characterization of the complexity of resilience via triads (Theorem 3.24). Now we will use that analysis to prove that the complexity of a closed query is NP-complete if it contains a triad, and in PTIME otherwise. Thus Conjecture 3.32 is true and the dichotomy for the complexity of resilience remains true in the presence of FDs.

**Lemma 3.33** (Closed queries with triads are hard)**.** *Let* $(q^*;\Phi)$ *be a closed sj-free CQ all of whose dominated atoms are exogenous. If* $q^*$ *has a triad, then* $\texttt{RES}(q^*;\Phi)$ *is* NP*-complete.*

*Proof.* Let $(q^*;\Phi)$ be as in the statement of the lemma. Recall that we proved in Lemma 3.10 that $\texttt{RES}(q_\triangle) \leq \texttt{RES}(q^*)$ and thus $\texttt{RES}(q^*)$ is NP-complete. Let $f$ be the reduction we produced from $\texttt{RES}(q_\triangle)$ to $\texttt{RES}(q^*)$. We will now show that if $f(D,k) = (D',k')$ then $D' \models \Phi$. It will then follow that $f$ is a reduction from $\texttt{RES}(q_\triangle)$ to $\texttt{RES}(q^*;\Phi)$. Thus $\texttt{RES}(q^*;\Phi)$ is NP-complete as claimed.

To see why $D' \models \Phi$, we will recall the definition of the reduction in the proof of Lemma 3.10. But first, we will examine how $q_\triangle$ (Example 3.2) itself is affected by FDs.

In particular, let $\Phi_0$ be any set of FDs for which $(q_\triangle, \Phi_0)$ is closed under induced rewrites. Notice that since $q_\triangle$ is closed, there can be no nontrivial unary FDs such as $x \to y$, (otherwise, $T(z, x)$ would have been replaced by $T'(z, x, y)$) nor any nontrivial binary FDs such as $xy \to z$ (otherwise $R(x, y)$ would have been replaced by $R'(x, y, z)$). In fact, $\Phi_0$ has no nontrivial FDs, i.e., $\Phi_0 = \emptyset$.

Now recall the reduction from $\texttt{RES}(q_\triangle)$ to $\texttt{RES}(q^*)$ in the proof of Lemma 3.10. What that proof did was to embed $q_\triangle$ into $q^*$. Using the triad of $q^*$, $\mathcal{T} = \{S_0, S_1, S_2\}$, we partitioned the variables of $q^*$ into 7 sets, and for each assignment of $x, y, z$ to values $a, b, c \in \texttt{dom}(D)$, we made assignments according to that partition (see Equation 3.14).

The net effect, is that just as for $q_\triangle$, since $(q; \Phi)$ is closed, it must be the case that $D' \models \Phi$. In particular, suppose that $\Phi$ contains the FD, $\mathbf{u} \to v$. First suppose that $\mathbf{u}$ is contained in one of the 7 sets of the partition (see Equation 3.14). Then, since $(q^*; \Phi)$ is closed, $v$ must be in the same set and thus it has exactly the same value as each of the variables in $\mathbf{u}$. If $\mathbf{u}$ has a variable from $V_3$ $(\texttt{var}(q) - (\texttt{var}(S_0) \cup \texttt{var}(S_1) \cup \texttt{var}(S_2)))$ then its value is $\langle abc \rangle$ so it determines all other variables. Similarly, if $\mathbf{u}$ has variables from two of $V_0, V_1, V_2$ then it again determines all three values. Suppose $\mathbf{u}$ does not determine all three values, e.g., say it does not determine $c$. Then, looking at Equation 3.14, we see that all the variables of $\mathbf{u}$ are from $V_0, V_4$ or $V_5$, i.e., they are all from $\texttt{var}(S_0)$. But then since $(q^*; \Phi)$ is closed, $v$ must be in $\texttt{var}(S_0)$ as well, and thus it is determined by $a$ and $b$.

Thus, we have shown that the reduction $f$ is also a reduction from $\texttt{RES}(q_\triangle)$ to $\texttt{RES}(q^*, \Phi)$ and thus the latter problem is NP-complete. □

### 3.2.4 Dichotomy of resilience with FDs

Recall that FDs cannot increase the complexity of resilience and thus if $q$ has no triad, then $\texttt{RES}(q;\Phi) \in \mathsf{PTIME}$ (Corollary 3.27). Thus, we have succeeded in proving the dichotomy for resilience in the presence of FDs:

**Theorem 3.34** (FD Dichotomy)**.** *Let $(q;\Phi)$ be a sf-free CQ with functional dependencies. Let $(q^*,\Phi)$ be its closure under induced rewrites, and such that all dominated atoms of $q^*$ are exogenous. If $q^*$ has a triad then $\texttt{RES}(q;\Phi)$ is $\mathsf{NP}$-complete. Otherwise, $\texttt{RES}(q;\Phi) \in \mathsf{PTIME}$.*

Note that we have also proved Conjecture 3.32:

**Corollary 3.35** (Induced rewrites suffice)**.** *Let $(q;\Phi)$ be an sj-free CQ with functional dependencies, and let $q^*$ be the closure of $q$ under induced rewrites. Then, $\texttt{RES}(q;\Phi) \equiv \texttt{RES}(q^*;\Phi) \equiv \texttt{RES}(q^*)$.*

# CHAPTER 4

# COMPLEXITY OF RESILIENCE FOR QUERIES WITH SELF-JOINS

Self-joins have long plagued the complexity study of many problems in database theory research and that also applies to resilience. Their presence makes the problem of categorizing the complexity of a query much richer and more complicated. Queries with triads remain hard, but self-joins can make queries without triads and even linear queries hard. There is no longer a polynomial time algorithm (in the size of the query) to tell whether a query is hard or easy. Furthermore, the order and repetition of variables can effect the complexity of resilience. These are irrelevant in the self-join free case. We essentially characterize the complexity of resilience when all relations have arity at most two and we do the same for queries of unbounded arity with just two atoms.

To the best of our knowledge, there are no current results for the self-join case for either view-side effects or source-side effects despite quite some work on the self-join free case [9], [14], [33], [32], [34], [4].

In this chapter, we first show that very simple queries can already be hard (Section 4.1), indicating that we will need more than the notion of triads to characterize hardness. However, triads can still help in certain cases (Section 4.3). We then focus on the case of linear queries and give various criterioa of hardness and show some cases we identified as in PTIME Section 4.4.

## 4.1 Basic hard queries

We next show that self-joins change everything that we have known about resilience so far. In the sj-free case, a query needed a triad to be hard. In particular, a query needed at least 3 variables and 3 atoms to be hard. We next prove hardness for two queries that will play an important role in our later results. The first $q_{\text{vc}}$ (for "vertex cover") has only 2 variables and 3 atoms. The second $q_{\text{chain}}$ (since it "chains" two binary relations together) has only 2 atoms and 3 variables:

$$q_{\text{vc}} :- A(x), R(x, y), A(y) \qquad \text{(Vertex cover)}$$

$$q_{\text{chain}} :- R(x, y), R(y, z) \qquad \text{(Chain query)}$$

**Proposition 4.1.** $\texttt{RES}(q_{\text{vc}})$ *is* NP*-complete.*

*Proof.* First note that any database with vocabulary $A, R$, with unary $A$ and binary $R$ can be seen as a directed graph, where $A$-tuples are the nodes and $R$-tuples represent directed edges. That said, $D \models q_{\text{vc}}$ if and only if there is an edge on the graph. To make the query false we need to delete all the edges, which is known as the NP-complete vertex cover problem. Therefore, $\texttt{RES}(q_{\text{vc}})$ is NP-complete. □

**Proposition 4.2.** $\texttt{RES}(q_{\text{chain}})$ *is* NP*-complete.*

*Proof.* We define a reduction from 3SAT to $\texttt{RES}(q_{\text{chain}})$. It will then follow that $\texttt{RES}(q_{\text{chain}})$ is NP-complete. While the reduction is from 3SAT, the intuition behind the reduction can be connected to the vertex cover problem (Fig. 4.5). The variable gadget is such that a minimum cover will choose either blue nodes (variable is set to `true`), or red nodes (variable is set to `false`). The clause gadget (black nodes) is chosen to enforce a clause: if one or more of the outermost black nodes are chosen, then the minimum cover is 2, otherwise 3.

Let $\psi$ be a 3CNF formula with $n$ variables $v_1, \ldots, v_n$ and $m$ clauses $C_1, \ldots, C_m$. Our reduction will map any such $\psi$ to a pair $(D_\psi, k_\psi)$ where $D_\psi$ is a database satisfying $q_{\text{chain}}$, and

$$\psi \in \text{3SAT} \quad \Leftrightarrow \quad (D_\psi, k_\psi) \in \texttt{RES}(q_{\text{chain}})$$

In our construction, if $\psi \in \text{3SAT}$, then the size of each minimum contingency set for $q_{\text{chain}}$ in $D_\psi$ will be $k_\psi = (n+5)m$, whereas if $\psi \notin \text{3SAT}$, then the size of all contingency sets for $q_{\text{chain}}$ in $D_\psi$ will be greater than $k_\psi$.

1. Variable gadget: For each variable $v_i$ and each $j \in [m]$ insert the following two tuples into the database: $R(v_i^j, \overline{v_i^j})$ and $R(\overline{v_i^j}, v_i^{j+1})$. If $j+1 > m$, make the superscript 1. The resulting joins between the tuples form a cycle of length $2m$. The minimum contingency sets are to either choose all tuples $R(v_i^j, \overline{v_i^j})$ representing a variable to have assignment `true`, or all tuples $R(\overline{v_i^j}, v_i^{j+1})$ representing a variable to have assignment `false`.
2. Clause gadget: For each clause $j \in [m]$ insert the following 6 tuples into the database: $R(a_j, b_j)$, $R(b_j, c_j)$, $R(c_j, a_j)$ $R(a'_j, a_j)$, $R(b'_j, b_j)$, $R(c'_j, c_j)$. The intuition is that the resulting joins form a triangle. If either of the $R(*', *)$ is removed, then the remaining joins can be destroyed by choosing only 2 or more tuples, otherwise we need 3.
3. Connecting the gadgets: For each variable $i$ that appears as positive in clause $j$ at position 1, add the following tuple: $R(\overline{v_i^j}, a'_j)$. For each variable $i$ that appears as negated in clause $j$ at position 1, add the following tuple: $R(v_i^{j+1}, a'_j)$. Analogously use $b'_j$ or $c'_j$ instead of $a'_j$ for positions 2 and 3 instead of position 1.

Fig. 4.1 illustrates an excerpt from the gadget by focusing on clause $C_1 = (v_1 \vee \bar{v}_2 \vee v_3)$. Notice that since there is a valuation that makes the clause true, only 5 tuples need to be removed to break all joins. □

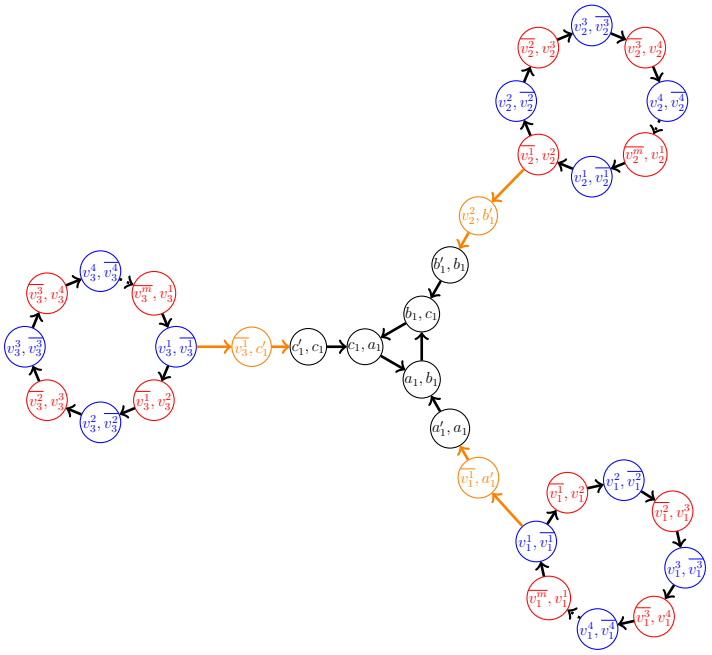


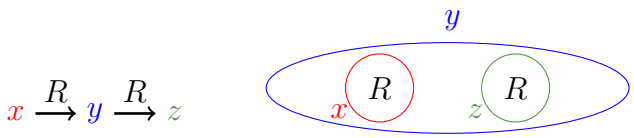
Figure 4.1: Excerpt from the construct showing the gadget for clause $C_1 = (v_1 \vee \bar{v}_2 \vee v_3)$ in the hardness proof for $q_{\text{chain}}$. Note that blue nodes represent a `true` value and red nodes a `false` value.

## 4.2 Notation and setup

In this section we discuss certain properties of conjunctive queries that were not relevant for the self-join free case. We explain why they matter now and what assumptions we make before we present our results.

In Chapter 3 we used the query dual hypergraph (Definition 3.1) in order to help identify triads. We can still use that representation for queries with self-joins. However, certain features of the query can be hidden in this representation, for example, the order in which variables appears in the atoms. Since we are only focusing on binary queries with self-joins we can represent those queries as a graph.

**Definition 4.3** (Binary graph)**.** *Let* $q :- A_1, \ldots, A_m$ *be a binary query with self-joins. Its* binary graph *has vertex set* $V = \texttt{var}(q)$ *and labeled edge sets defined by atoms* $A_1, \ldots, A_m$*. For unary atoms, the edge will be a loop.*




(a) Binary graph for $q_{\text{chain}}$

(b) Hypergraph for $q_{\text{chain}}$

Figure 4.2: Hypergraphs only represent which variables occur in a given atom, whereas binary graphs represent containment and position within each atom, as we can see for query $q_{\text{chain}}$

See Fig. 4.2 to illustrate the differences between the hypergraph and binary graph of a query.

### 4.2.1 Query minimization

Given queries $q_1$ and $q_2$, we say that $q_1$ is *contained* in $q_2$ ($q_1 \subseteq q_2$) if answers to $q_1$ over any database instance $D$ are always a subset of the answers to $q_2$ over $D$. We say $q_1$ is *equivalent* to $q_2$ ($q_1 \equiv q_2$) if $q_1 \subseteq q_2$ and $q_2 \subseteq q_1$ [1]. We say a conjunctive query $q$ is *minimal* if for every other conjunctive query $q'$ such that $q \equiv q'$, $q'$ has at least as many atoms as $q$. For every query $q$, there exists a minimal equivalent CQ $q'$ that can be obtained from $q$ by removing zero or more atoms [10].

We next establish a rule that we will always have to minimize a query before evaluating the complexity of resilience. The reason is that our hardness evaluation will rely on identifying certain subqueries (or patterns) in a query that make this query hard by enabling a reduction from 3SAT or another already established hard query. However, if a subquery is removed during minimization, then, intuitively, this pattern does not allow this reduction, and does not render the original query hard. While this observation seems obvious, the assumption of minimality is a key aspect of our proofs.

We illustrate this next with an example of a query which seems to contain the subquery $q_{\text{chain}}$ which we know to be NP-complete (Proposition 4.2). However, it con-

tains an additional atom which will "disarm" this hard pattern during minimization. This results in our original reduction not going through anymore.

**Example 4.4** (Query minimization)**.** *Consider queries $q$ and its minimized version $q_{min}$*

$$q :- R(x,y), R(y,x), R(y,z)$$
$$q_{min} :- R(x,y), R(y,x)$$

*over a reduced database $D$ with tuples $R(1,2), R(2,1), R(1,3)$. Then $q_{min}$ has 3 joins:*

$$\langle 1,2,3\rangle = R(1,2), R(2,1), R(1,3)$$
$$\langle 1,2,1\rangle = R(1,2), R(2,1), R(1,2)$$
$$\langle 2,1,2\rangle = R(2,1), R(1,2), R(2,1)$$

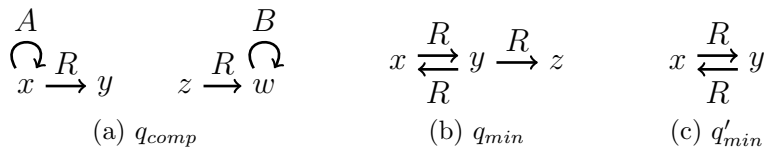
*$q_{min}$ has only 2 joins which are obtained with the same tuples:*

$$\langle 1,2\rangle = R(1,2), R(2,1)$$
$$\langle 2,1\rangle = R(2,1), R(1,2)$$

*Both queries have as minimum contingency set, given $D$, either $\{R(1,2)\}$ or $\{R(2,1)\}$. Observe that a tuple like $R(1,3)$ that participates only in joins for $q$ but not in $q_{min}$ need never be chosen to be part of a contingency set.*

**Lemma 4.5** (Minimization for resilience)**.** *Let $q$ be a CQ and $q'$ the query resulting from minimizing $q$. Then $\texttt{RES}(q) \equiv \texttt{RES}(q')$.*

*Proof.* Because $q$ and $q'$ are equivalent we can say that $D \models q \leftrightarrow D \models q'$. Therefore, for any $\Gamma$, we have $D - \Gamma \not\models q \leftrightarrow D - \Gamma \not\models q'$. This proves that a contingency set for $D, q$ is a contingency set for $D, q'$ and vice-versa. □




(a) $q_{comp}$ (b) $q_{min}$ (c) $q'_{min}$

Figure 4.3: (a) illustrates a query with two components. (b) and (c) show a query and its minimized version, respectively.

Thus when studying resilience, we follow the convention that all our queries are minimized.

### 4.2.2 Query components

A *connected component* of $q$ (or "component" in short) is a maximal subset of atoms that are connected via existential[1] variables. A query $q$ is *disconnected* if its atoms can be partitioned into two or more components that do not share any existential variables. For example,

$$q_{comp} :- A(x), R(x, y), R(z, w), B(w)$$

is disconnected and has two components:

$$q^1_{comp} :- A(x), R(x, y)$$
$$q^2_{comp} :- R(z, w), B(w)$$

The resilience of a query is determined by taking the minimum of the resiliences of each of its components. In the following, let $\rho(q, D)$ stand for the resilience of query $q$ over database $D$, which is the size of the minimum contingency set for $(q, D)$.

**Lemma 4.6** (Query components)**.** *Let $q :- q_1, \ldots, q_k$ be a query that consists of $k$ components $q_i$, $i \in [k]$. Then $\rho(q, D) = \min_i \rho(q_i, D)$.*

[1]Because we only consider Boolean queries, all variables are existential.

*Proof.* First observe that disconnected components join as a cross-product, so for a query to be made false it is necessary and sufficient that at least one of its query components is made false. Hence, for each query component $q_i$, if $D - \Gamma_i \not\models q_i$, then $D - \Gamma_i \not\models q$, which then implies $\rho(q, D) = \min_i \rho(q_i, D)$. □

We can now show that the complexity of a query is determined by the hardest of its components, if the query is minimal:

**Lemma 4.7** (Query components complexity)**.** *Let $q$ be a minimal query that consists of $k$ query components.* $\texttt{RES}(q)$ *is* NP*-complete if and only if $q$ has some component, $q_i$, for which* $\texttt{RES}(q_i)$ *is* NP*-complete.*

In the remainder of the paper we assume queries are connected.

### 4.2.3 Isolated variables

The definition and lemma below help proving hardness in certain cases.

**Definition 4.8** (Isolated variables)**.** *Consider $\{w_1, \ldots, w_j\}$ to be the set of variables appearing as the first attribute of all occurrences of relation $R$ in a query $q$. We say that $\{w_1, \ldots, w_j\}$ are* isolated *for $R$ in $q$, if those variables only occur in $R$ and only as a first attribute.*

**Lemma 4.9.** *Consider $q :- R(w_i, x_1), \ldots, R(w_j, x_l), \ldots$ and $\{w_i, \ldots, w_j\}$ a set of isolated variable. Then $\texttt{RES}(q') \leq \texttt{RES}(q)$, with*

$$q' :- R'(x_1), \ldots, R'(x_l), \ldots.$$

*Proof.* Let $D'$ be a database and $D' \models q'$. We can create $D$ by augmenting $R$-tuples in $D$ with a new constant value $\langle c \rangle$ and keeping other tuples the same. With that modification, we have a 1:1 correspondence of the tuples and joins, and the contingency sets are preserved. □

**Example 4.10.** *Consider query*

$$q :- A(y), R(y, x), B(y, w), R(w, z).$$

*We can see that* $\{x, z\}$ *are variables isolated to* $R$*, and therefore we can obtain* $q'$*, such that* $\texttt{RES}(q') \leq \texttt{RES}(q)$*,*

$$q' :- A(y), R'(y), B(y, w), R'(w).$$

*Since* $\texttt{RES}(q')$ *is hard, it follows from Lemma 4.9 that* $\texttt{RES}(q)$ *is hard as well.*

*Now consider* $q_{\text{conf}}$*, that we saw earlier is an easy query:*

$$q_{\text{conf}} :- A(x), R(x, y), R(z, y), C(z).$$

*According to the definition of isolated variables, Definition 4.8,* $y$ *is isolated in* $q_{\text{conf}}$*, therefore we can obtain*

$$q'_{\text{conf}} :- A(x), R'(x), R'(z), C(z),$$

*which is now a disconnected query. Note that* $\texttt{RES}(q'_{\text{conf}})$ *is in* P*.*

We do not assume that all isolated variables are deleted from our queries. Since we can decrease complexity by doing so, we can only use this as a tool to show hardness results.

### 4.2.4 Domination for the self-join case

We defined domination for the sj-free case as the notion that a relation $A$ is more relevant for resilience than another relation $B$ if $\texttt{var}(A) \subseteq \texttt{var}(B)$. In the next example, we see that is not true anymore when considering relations involved in a self-join:

**Example 4.11.** *Consider a query similar to $q_{rats}$ but having self-joins:*

$$q^{sj}_{rats} :- A(x), R(x,y), R(y,z), R(z,x)$$

*Remember that $q_{rats}$ has no triads because $R, T$ are dominated by $A$. Following the definition of domination, $R$ should be dominated by $A$ in $q^{sj}_{rats}$, but that is not the case. Let $D$ be a database with tuples $\{A(a), A(b), R(a,b), R(b,c), R(c,a), R(c,d), R(d,b)\}$. We have 3 joins, namely $\langle abc\rangle$, $\langle bca\rangle$ and $\langle bcd\rangle$:*

$$\langle abc\rangle = A(a), R(a,b), R(b,c), R(c,a)$$
$$\langle bca\rangle = A(b), R(b,c), R(c,a), R(a,b)$$
$$\langle bcd\rangle = A(b), R(b,c), R(c,d), R(d,a)$$

*The minimum contingency set is $\Gamma = \{R(b,c)\}$ and, since there is no other minimum contingency set, we can conclude that $R$ is not exogenous.*

The old definition of domination does not work here but we have defined a generalization of Definition 3.8 that works.

We write $\texttt{pos}_g(i) = x$ to express that the $i$-th attribute of atom $g$ is variable $x$. We write $[k]$ as short notation for the set $\{1, \ldots, k\}$.

**Definition 4.12** (Domination)**.** *Let relations $A$ and $B$ be endogenous relations for a* ***connected*** *query $q$. We say that $A$* dominates *$B$ if there exists a function*

$$f : [\texttt{arity}(A)] \to [\texttt{arity}(B)]$$

*such that for each occurrence $h$ of $B$, there exists an occurrence $g$ of $A$ such that $pos_h(i) = pos_g(f(i))$ for $\forall i \in [\texttt{arity}(A)]$. Notice that for the case of $B$ appearing only once, we get back our original definition of domination: $\texttt{var}(A) \subseteq \texttt{var}(B)$.*


(a) $q_1$ (b) $q_1$

Figure 4.4: $A$ dominates $R$ in $q_1$ but not in $q_2$.

**Example 4.13.** *Consider queries $q_1$ and $q_2$:*

$$q_1 :- A(x), R(x, y), A(y), R(y, z)$$

$$q_2 :- A(x), R(x, y), A(z), R(y, z)$$

*$A$ dominates $R$ in $q_1$ but not in $q_2$.*

Notice that domination as defined in Definition 4.12 is a generalization of the previous notion of domination (Definition 3.8): If a relation $R$ appears only in one atom $g_i$ in a query with variables $\texttt{var}(g_i) = \{x, y\}$, then it is enough to have a unary relation $A$ that appears in a subgoal $g_j$ with $\texttt{var}(g_j) = \{x\}$ or $\texttt{var}(g_j) = \{y\}$.

**Proposition 4.14** (Domination with self-join)**.** *Let $q$ be a conjunctive query and $q'$ the query resulting from labeling some dominated relations as exogenous. Then $\texttt{RES}(q) \equiv \texttt{RES}(q')$.*

*Proof.* We show that tuples from dominated relations do not need to be used in minimum contingency sets. Assume $q$ is a connected query and let $\Gamma$ be a minimum contingency set of $q$ in $D$.

Suppose that relation $A$ dominates relation $B$ and there is some tuple $B(\mathbf{t})$ that is in $\Gamma$. Tuple $B(\mathbf{t})$ can participate in joins as one or more of the $B$-atoms in $q$. Let's call those atoms $B_i$, for $i \in [k]$. Our definition of domination guarantees that there exists an atom $A_j$ for each atom $B_i$ such that the projection of $\mathbf{t}$ onto $\texttt{var}(A_j)$ always produces the same tuple $\mathbf{p}$. Then we can replace $B(\mathbf{t})$ by $A(\mathbf{p})$ and we remove at least as many witnesses if $D \models q$.

As a result we show the complexity of $\texttt{RES}(q)$ is the same if $B$ is made exogenous and therefore $\texttt{RES}(q) \equiv \texttt{RES}(q')$. □

When studying resilience for queries with self-joins, we follow the convention that *all dominated atoms are exogenous*, as we did for the self-join free case. In other words, after we minimize the query, we transform it so that the dominated atoms are exogenous. We will indicate such exogenous atoms with the superscript "x". In the remainder, we will refer to a query that is minimized, connected and that has all dominated atoms made exogenous as a "*minimal query.*"

## 4.3 Non-linear queries

In this section we will show when non-linear queries will be hard. The intuition is that we can use the notion of triads from the sj-free case.

We next show that minimal queries that contain triads are still NP-hard even if they contain a self-join (whether or not the triad consists of repeated relations).

**Definition 4.15** (Subgoal renaming)**.** *Consider queries $q$ and $q'$. We say that $\Phi(q, q')$ holds if and only if we can transform $q'$ into $q$ by* renaming *one or more atoms of $q'$ with new relational symbols. For each atom $g$ in $q$ we define function $f$ to return the corresponding atom in $q'$, i.e. $f(g) = g'$.*

**Example 4.16.** *Consider queries below*

$$q :- R(x, y), S(y, z), A(x), T(x, y), W(y, z)$$
$$q' :- R(x, y), R(y, z), A(x), T(x, y), T(y, z)$$

*Given our definition, $\Phi(q, q')$ is true, and*

$$f = \{(R(x,y), R(x,y)), (S(y,z), R(y,z)),$$
$$(A(x), A(x)), (T(x,y), T(x,y)), (W(y,z), T(y,z))\}$$

**Lemma 4.17** (Self-join elimination)**.** *If $\Phi(q, q')$ where $q$ is a sj-free query and $q'$ is a minimal self-join query, then* $\texttt{RES}(q) \leq \texttt{RES}(q')$.

*Proof.* Consider $q :- \exists x_1, \ldots, x_k \, (A_1 \wedge \cdots \wedge A_t)$ and let $D$ be a database satisfying $q$. Since $q$ is sj-free, we can replace $D$ by an equivalent version, $D_{\text{col}}$, in which each element is colored according to which variable it represents. For example, suppose that the atom $S(x, y, z)$ occurs in $q$. We change $S$ to its colored version as follows,

$$S_{\text{col}} = \left\{ (a_{1,x}, a_{2,y}, a_{3,z}) \;\middle|\; (a_1, a_2, a_3) \in S \right\}$$

Observe that $D_{\text{col}}$ satisfies $q$ and there is a 1:1 correspondence between the joins of $q$ in $D$ and those in $D_{\text{col}}$.

Now we define the reduction $\texttt{RES}(q) \leq \texttt{RES}(q')$. Given a colored database $D_{\text{col}}$, we create $D'$ as follows:

$$R' = \{(a, b) \mid g(a, b) \in D_{\text{col}} \wedge f(g) = R'\}$$

It is easy to see that a join in $D_{\text{col}}, q$ will also be a join in $D', q'$. Now suppose that $D', q'$ has a new join. That only can happen if there is a tuple joining with an atom with a different combination of colors. Suppose tuple $R(a_x, b_y)$ participates in new a join through atom $R(w, z)$. We know there exists an atom $R(x, y)$ in $q'$, o.w. $R(a_x, b_y)$ would not be a tuple in $D'$, and for any occurrence of $w, z$ there must be a corresponding occurrence of $x, y$. This would imply that $q'$ is not minimal, contradicting our assumption.

Since there is a 1:1 correspondence between joins and tuples, the contingency sets are preserved. □

**Corollary 4.18** (Self-join queries with triads)**.** *If $q$ has a triad, $q'$ is minimal and $\Phi(q, q')$, then* $\texttt{RES}(q')$ *is* NP*-complete.*

Lemma 4.17 is very useful but it does not allow us to classify all non-linear queries. Remember that in the sj-free case, some non-linear queries are in P. Because of domination they do not have a triad and there exists an equivalent linear query, with respect to resilience. For example $q_{\text{rats}}$ and $q_{\text{brats}}$ are both easy.

Consider the following queries as a possible self-join variation of $q_{\text{rats}}$ and $q_{\text{brats}}$:

$$q^{sj}_{\text{rats}} :- A(x), R(x, y), R(y, z), R(z, x)$$
$$q^{sj}_{\text{brats}} :- A(x), R(x, y), B(y), R(y, z), R(z, x)$$

We can obtain $q_{\text{rats}}$ and $q_{\text{brats}}$ from $q^{sj}_{\text{rats}}$ and $q^{sj}_{\text{brats}}$ through subgoal renaming (Definition 4.15) but it does not help elucidate what their complexity is. In fact, their resilience is hard.

**Proposition 4.19.** $\texttt{RES}(q^{sj}_{rats})$ *and* $\texttt{RES}(q^{sj}_{brats})$ *are* NP*-complete.*

*Proof.* We show that $\texttt{RES}(q^{sj}_{\text{rats}})$ is NP-complete by a reduction from 3SAT, similar to the one used to prove $\texttt{RES}(q_{\triangle})$ is NP-complete (Proposition 3.3).

Let $\psi$ be a 3CNF formula with $n$ variables $v_1, \ldots, v_n$ and $m$ clauses $C_0, \ldots, C_{m-1}$. Our reduction will map any such $\psi$ to a pair $(D^s_\psi, k_\psi)$ where $D^s_\psi$ is a database satisfying $q^{sj}_{\text{rats}}$, and

$$\psi \in \text{3SAT} \quad \Leftrightarrow \quad (D^s_\psi, k_\psi) \in \texttt{RES}(q^{sj}_{\text{rats}})$$

In our construction, if $\psi \in$ 3SAT, then the size of each minimum contingency set for $q^{sj}_{\text{rats}}$ in $D^s_\psi$ will be $k_\psi = 6mn$, whereas if $\psi \notin$ 3SAT, then the size of all contingency sets for $q^{sj}_{\text{rats}}$ in $D^s_\psi$ will be greater than $k_\psi$.

We construct $D^s_\psi$ by taking $D_\psi$ from Proposition 3.3, and adding the following tuples for each join $\langle a, b, c\rangle$ in $D_\psi, q_\triangle$:

$$R = \{(a, b), (b, c), (c, a)\}$$
$$A = \{(a), (b), (c)\}$$

Notice that for each join $\langle a, b, c\rangle$ we create 3 joins, $\langle a, b, c\rangle, \langle b, c, a\rangle, \langle c, a, b\rangle$ but they all use the same $R$-triangle.

We know from Proposition 3.3 that some $R$-tuples participate in 2 joins (triangles) and some only in 1 within a variable gadget. Observe that $A$-tuples only participate in 2 joins each, so it is never better to choose $A$-tuples. Therefore it follows that the same choice of tuples for the minimum contingency set for $D_\psi, q_\triangle$ will also work for $D^s_\psi, q^{sj}_{\text{rats}}$ by choosing the corresponding $R$-tuples in $D^s_\psi$ based on the $R, S, T$-tuples chosen from $D_\psi$.

The same idea works for query $q^{sj}_{\text{brats}}$. When defining $D^s_\psi$ for this case, we just need to add the appropriate $B$-tuples:

$$R = \{(a, b), (b, c), (c, a)\}$$
$$A = \{(a), (b), (c)\}$$
$$B = \{(a), (b), (c)\}$$

Since $B$-tuples have the same properties of the $A$-tuples, they are never better choices than $R$-tuples and we can obtain a minimum contingency set with only $R$-tuples, as we saw above. □

As we saw with $q^{sj}_{\text{rats}}$ and $q^{sj}_{\text{brats}}$, subgoal renaming is not always helpful and we might need an alternative way of showing the complexity of a non-linear query. In the following, we look into some similar cases that require different proofs.

$$q^{sj'}_{\text{rats}} :- A(x), R(x,y), S(y,z), R(z,x)$$
$$q^{sj''}_{\text{rats}} :- A(x), R(x,y), R(y,z), T(z,x)$$

**Proposition 4.20.** $\texttt{RES}(q^{sj'}_{rats})$ *and* $\texttt{RES}(q^{sj''}_{rats})$ *are* NP*-complete.*

*Proof.* First notice that $T$ is dominated by $A$ in $q^{sj''}_{\text{rats}}$. Therefore a reduction from $\texttt{RES}(q^{\text{a}}_{\text{chain}})$ is straightforward.

For $q^{sj'}_{\text{rats}}$ we use the same reduction we used to show $\texttt{RES}(q_{\text{chain}})$ is NP-complete (Proposition 4.2), but with the appropriate $A$- and $S$-tuples. By doing that, note that $A$-tuples and $S$-tuples in the variable gadgets are only involved in one join each, whereas $R$-tuples are involved in two joins each, which makes $R$-tuples the best choice for contingency sets. In the clause gadget, the same is true for $S$-tuples but not $A$-tuples. However, for an $A$-tuple in two joins there is always an equivalent choice of $R$-tuple, so the minimum contingency sets are the same as in Proposition 4.2. □

$$q^{sj'}_{\text{brats}} :- A(x), R(x,y), A(y), R(y,z), R(z,x)$$
$$q^{sj''}_{\text{brats}} :- A(x), R(x,y), B(y), S(y,z), R(z,x)$$
$$q^{sj'''}_{\text{brats}} :- A(x), R(x,y), B(y), R(y,z), T(z,x)$$

**Proposition 4.21.** $\texttt{RES}(q^{sj'}_{brats})$, $\texttt{RES}(q^{sj''}_{brats})$ *and* $\texttt{RES}(q^{sj'''}_{brats})$ *are* NP*-complete.*

*Proof.* $\texttt{RES}(q^{sj'}_{\text{brats}})$ is NP-complete through a reduction from $\texttt{RES}(q_{\text{vc}})$. Given a database $D$ we want to define a database $D'$ such that

$$(D,k) \in \texttt{RES}(q_{\text{vc}}) \Leftrightarrow (D',k) \in \texttt{RES}(q^{sj'}_{\text{brats}})$$

We define $R$ as follows:

$$R = \{(a,b), (b, \langle ab \rangle_z), (\langle ab \rangle_z, a) | D \models q_{\text{vc}}(a,b)\}$$

($\Rightarrow$) With this construction we guarantee that any join in $D', q^{sj'}_{\text{brats}}$ has only one correspondent in $D, q_{\text{vc}}$. Therefore, any contingency set $\Gamma$ for $D, q_{\text{vc}}$ is also a contingency set for $D', q^{sj'}_{\text{brats}}$.

($\Leftarrow$) Let $\Gamma'$ be a minimum contingency set for $D', q^{sj'}_{\text{brats}}$. If $\Gamma'$ contains a tuple with a new domain value, for example $R(b, \langle ab \rangle_z)$ or $R(\langle ab \rangle_z, a)$, then we can exchange it by $A(b)$ or $A(a)$, respectively. Therefore, we can obtain another contingency set $\Gamma''$ of same size as $\Gamma'$ but that is a contingency set for $D, q_{\text{vc}}$, since it only contains tuples with domain values.

Note that in this case, it does not matter if the binary atoms are all $R$'s or not, just the presence of $A(x), A(y)$ is enough to show hardness.

In $q^{sj''}_{\text{brats}}$, $S$ is exogenous,

$$q^{sj''}_{\text{brats}} :- R(z, x), A(x), R(x, y), B(y), S^{\text{x}}(y, z),$$

so we can show hardness with a straightforward reduction from $q^{\text{bc}}_{\text{chain}}$.

In $q^{sj'''}_{\text{brats}}$, $T$ is exogenous,

$$q^{sj'''}_{\text{brats}} :- T^{\text{x}}(z, x), A(x), R(x, y), B(y), R(y, z),$$

and, similar to the case above, we can show hardness with a straightforward reduction from $q^{\text{ab}}_{\text{chain}}$. □

## 4.4 Linear queries

Unlike the sj-free case, linear queries need no longer be in PTIME, as made evident by Proposition 4.1 and Proposition 4.2. In this section we present the 3 main structures we can find in linear queries with self-joins and how they help us understand the complexity of resilience for those queries.

### 4.4.1 Chains

We next show some variations of query $q_{\text{chain}}$ and explain how the reduction to show hardness can vary with small variations in the query. Nevertheless, we can show that any linear query with self-joins that only forms chains will be NP-complete. We first show that chains of length longer than 2 are all NP-complete.

**Lemma 4.22.** *Consider queries*

$$q^l_{\text{chain}} :- R(x,y), R(y,z), R(z,w_3), \ldots, R(w_{l-1}, w_l),$$

*for* $l \geq 3$. *Then* $\texttt{RES}(q^l_{\text{chain}})$ *is* NP*-complete.*

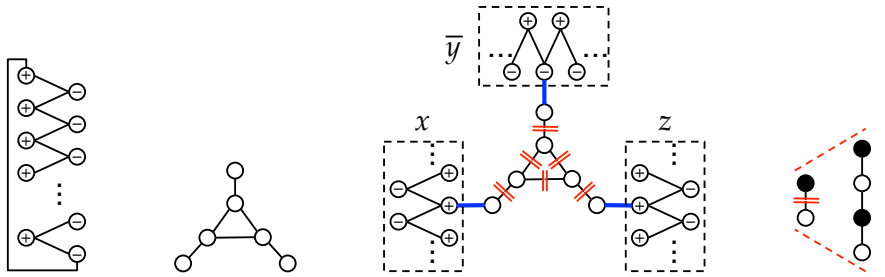
*Proof.* We define a reduction from $\texttt{RES}(q_{\text{chain}})$ to $\texttt{RES}(q^l_{\text{chain}})$. We may assume that there are no loops $R(a,a) \in D$. Given a database $D$, we define $D'$ as follows:

$$\begin{aligned} R' =& \{(a,b),(b,c) | R(a,b), R(b,c) \in D\} \\ & \cup \{(c, \langle abc \rangle_3), \ldots, \langle abc \rangle_{l-1}, \langle abc \rangle_l) | R(a,b), R(b,c) \in D\} \end{aligned}$$

Now we want to show

$$(D,k) \in \texttt{RES}(q_{\text{chain}}) \Leftrightarrow (D',k) \in \texttt{RES}(q^l_{\text{chain}})$$

Let $\Gamma$ be a minimum contingency set for $D, q_{\text{chain}}$. Suppose that $D' - \Gamma \models q^l_{\text{chain}}$ and let $\langle a_0, a_1, a_2, \ldots, a_l \rangle$ be a witness. First observe that, by construction, we cannot have more than $l-2$ new values in any given witness for $D', q^l_{\text{chain}}$, and that if $a_i \notin \texttt{dom}(D)$, then $a_j \notin \texttt{dom}(D)$ for $i \leq j$. Therefore, it must be that $a_0, a_1, a_2 \in \texttt{dom}(D)$, which contradicts the fact that $\Gamma$ was a contingency set for $D, q_{\text{chain}}$, proving that $D' - \Gamma \not\models q^l_{\text{chain}}$.




(a) Variable gadget (b) Clause gadget (c) Clause $(x \vee \bar{y} \vee z)$ (d) Extender

Figure 4.5: Intuition behind the proof of Proposition 4.2 in terms of vertex cover: the double red lines show "extenders" allowing to extend the distance between corners.

Now let $\Gamma'$ be a minimum contingency set for $D', q^l_{\text{chain}}$. First let's argue that there is a minimum $\Gamma''$ that only contains $R$-tuples that are also in $D$. That's true because any witness for $D', q^l_{\text{chain}}$ can only contain at most $l-2$ new values, so there are at least two $R$-tuples from $D$ participating in any given join, so it's better to chose $R$-tuples that came from $D$ instead of the ones with new values and therefore $D - \Gamma' \not\models q_{\text{chain}}$. □

Now we look into the following variations of query $q_{\text{chain}}$. Note that by including certain unary atoms we need to slightly change the clause gadget in our reductions. However, the intuition behind how we construct such reductions still remains. All the reductions follow the pattern of creating a variable gadget and a clause gadget which will be connected by an extender, accordingly to the 3SAT formula, as illustrated in Fig. 4.5.

Now consider the following variations of $q_{\text{chain}}$:

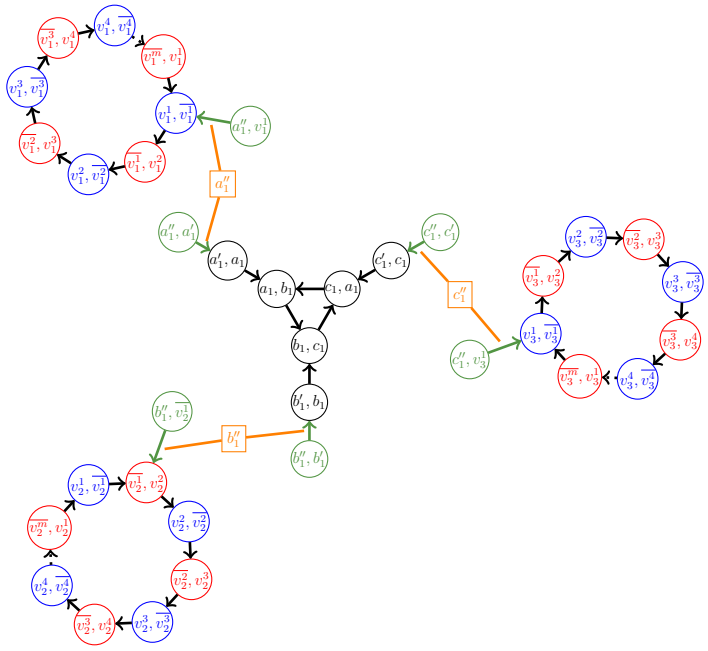


Figure 4.6: Excerpt from the construct showing the gadget for clause $C_1 = (v_1 \vee \bar{v}_2 \vee v_3)$ in the hardness proof for $q^{\mathrm{a}}_{\mathrm{chain}}$. We omit the $A$-tuples that participate in only one join, since they shall never be chose for a minimum contingency set.

$$q_{\mathrm{chain}} :- R(x,y), R(y,z)$$

$$q^{\mathrm{a}}_{\mathrm{chain}} :- A(x), R(x,y), R(y,z)$$

$$q^{\mathrm{b}}_{\mathrm{chain}} :- B(y), R(x,y), R(y,z)$$

$$q^{\mathrm{c}}_{\mathrm{chain}} :- C(z), R(x,y), R(y,z)$$

$$q^{\mathrm{ab}}_{\mathrm{chain}} :- A(x), B(y), R(x,y), R(y,z)$$

$$q^{\mathrm{bc}}_{\mathrm{chain}} :- B(y), C(z), R(x,y), R(y,z)$$

$$q^{\mathrm{ac}}_{\mathrm{chain}} :- A(x), C(z), R(x,y), R(y,z)$$

$$q^{\mathrm{abc}}_{\mathrm{chain}} :- A(x), B(y), C(z), R(x,y), R(y,z)$$

We next show resilience for all of them is NP-complete.

**Proposition 4.23.** $\mathtt{RES}(q^{\mathrm{a}}_{\mathrm{chain}})$ *is* NP*-complete.*

*Proof.* We again define a reduction from 3SAT, using gadgets similar to the one in Proposition 4.2. The variable gadget remains such that a minimum cover will choose either blue nodes (variable is set to `true`), or red nodes (variable is set to `false`). The clause gadget (black nodes) is chosen as to enforce a clause: if one or more of the outermost black nodes are chosen, then the minimum cover is 5, otherwise 6.

We next reduce 3SAT to $\texttt{RES}(q^{\text{a}}_{\text{chain}})$. Let $\psi$ be a 3CNF formula with $n$ variables $v_1, \ldots, v_n$ and $m$ clauses $C_1, \ldots, C_m$. Our reduction will map any such $\psi$ to a pair $(D_\psi, k_\psi)$ where $D_\psi$ is a database satisfying $q^{\text{a}}_{\text{chain}}$, and

$$\psi \in \text{3SAT} \quad \Leftrightarrow \quad (D_\psi, k_\psi) \in \texttt{RES}(q^{\text{a}}_{\text{chain}})$$

In our construction, if $\psi \in$ 3SAT, then the size of each minimum contingency set for $q^{\text{a}}_{\text{chain}}$ in $D_\psi$ will be $k_\psi = (n+5)m$, whereas if $\psi \notin$ 3SAT, then the size of all contingency sets for $q^{\text{a}}_{\text{chain}}$ in $D_\psi$ will be greater than $k_\psi$.

1. Variable gadget: For each variable $v_i$ and each $j \in [m]$ insert the following tuples into the database: $R(v_i^j, \overline{v_i^j})$, $R(\overline{v_i^j}, v_i^{j+1})$ and $A(v_i^j)$, $A(\overline{v_i^j})$. If $j+1 > m$, then make the superscript 1. The resulting joins between the tuples form a cycle of length $2m$. The minimum contingency sets are to either choose all tuples $R(v_i^j, \overline{v_i^j})$ representing a variable to have assignment `true`, or all tuples $R(\overline{v_i^j}, v_i^{j+1})$ representing a variable to have assignment `false`. Note that any $A$-tuple only joins once, therefore it is better to choose an $R$-tuple, since all of these join at least twice.
2. Clause gadget: For each clause $j \in [m]$ insert the following tuples into the database: $R(a_j, b_j)$, $R(b_j, c_j)$, $R(c_j, a_j)$, $R(a'_j, a_j)$, $R(b'_j, b_j)$, $R(c'_j, c_j)$, $A(a_j)$, $A(b_j)$, $A(c_j)$, $A(a'_j)$, $A(b'_j)$, $A(c'_j)$. The resulting joins form a triangle. If either of the $R(*', *)$ is removed, then the remaining joins can be destroyed by choosing only 2 or more tuples, otherwise we need 3. Similar to the variable gadget, $A$-tuples are not an optimal choice because they only participate in one join each.

3. Connecting the gadgets: For each variable $i$ that appears in clause $j$ at position 1, add the following tuples: $R(a''_j, a'_j)$ and $A(a''_j)$. If $v_i$ appears as positive add tuple $R(a''_j, v_i^j)$, if it appear as negative add tuple $R(a''_j, \overline{v_i^j})$. Analogously use $b'_j, b''_j$ or $c'_j, c''_j$ instead of $a'_j, a''_j$ for positions 2 and 3 instead of position 1.

Observe that if the clause is not satisfied, then we need to choose the $A$-tuples (orange squares in Fig. 4.6), and not choose the outer black nodes ($R$-tuples) in the clause gadget, resulting in choosing 6 tuples in total in order to delete all the joins, otherwise we just need 5 tuples. □

**Proposition 4.24.** $\texttt{RES}(q^{\text{b}}_{\text{chain}})$ *is* NP*-complete.*

*Proof.* For this case we are going to use almost the same reduction as the one used for $\texttt{RES}(q_{\text{chain}})$, just with the added $B$-tuples. Then we argue that there is always a min $\Gamma$ that only uses $R$-tuples.

Let $\psi$ be a 3CNF formula with $n$ variables $v_1, \ldots, v_n$ and $m$ clauses $C_1, \ldots, C_m$. Our reduction will map any such $\psi$ to a pair $(D_\psi, k_\psi)$ where $D_\psi$ is a database satisfying $q^{\text{b}}_{\text{chain}}$, and

$$\psi \in \text{3SAT} \quad \Leftrightarrow \quad (D_\psi, k_\psi) \in \texttt{RES}(q^{\text{b}}_{\text{chain}})$$

In our construction, if $\psi \in$ 3SAT, then the size of each minimum contingency set for $q^{\text{b}}_{\text{chain}}$ in $D_\psi$ will be $k_\psi = (n+5)m$, whereas if $\psi \notin$ 3SAT, then the size of all contingency sets for $q^{\text{b}}_{\text{chain}}$ in $D_\psi$ will be greater than $k_\psi$.

First, include in $D_\psi$ all the same $R$-tuples included in the proof of Proposition 4.2. In addition to that add the following $B$-tuples:

1. Variable gadget: For each variable $v_i$ and each $j \in [m]$ insert the following two tuples into the database: $B(v_i^j)$ and $B(\overline{v_i^j})$.
2. Clause gadget: For each clause $j \in [m]$ insert the following 6 tuples into the database: $B(a_j)$, $B(b_j)$, $B(c_j)$, $B(a'_j)$, $B(b'_j)$, $B(c'_j)$.

By adding those tuples, we obtain the same structure and joins of the reduction for $\texttt{RES}(q_{\text{chain}})$. Now suppose that $t = B(d)$ is in a minimum contingency set $\Gamma$. If $d = v_i^j$

(or $\overline{v_i^j}$) for some $i, j$, we know that $t$ must join with $t' = R(\overline{v_i^{j-1}}, v_i^j)$ (or $R(\overline{v_i^j}, v_i^j)$) by our construction. Thus, we can exchange $t$ for $t'$ and obtain contingency set $\Gamma'$. Similar, if $d \in \{a, b, c, a', b', c'\}$, then $t$ must join with tuple $R(d, *)$, since there is only tuple of that kind for each possible value of $d$.

This shows that there is a minimum contingency set for $D_\psi$ without $B$-tuples, and the properties of the reduction in Proposition 4.2 also hold in this case. □

**Proposition 4.25.** $\texttt{RES}(q^{\text{ab}}_{\text{chain}})$ *is* NP*-complete.*

*Proof.* We define a reduction from 3SAT. First, use the same $D_\psi$ as in the reduction for $\texttt{RES}(q^{\text{a}}_{\text{chain}})$, just adding the appropriate $B$-tuples, i.e., $B$-tuples that preserve the joins.

Now note that for any $t = B(d) \in D_\psi$, there is only one $R$-tuples such that $t' = R(d, *)$, therefore $t$ must join with $t'$. Therefore, any occurrence of $B$-tuple in a contingency set can be exchanged by its correspondent $R$-tuple, and we guarantee this reduction has the same properties as the one used in Proposition 4.23. □

**Proposition 4.26.** $\texttt{RES}(q^{\text{ac}}_{\text{chain}})$ *is* NP*-complete.*

*Proof.* We define a reduction from 3SAT. As in the previous cases, the variable gadget remains such that a minimum cover will choose either blue nodes (variable is set to `true`), or red nodes (variable is set to `false`). The clause gadget (center black nodes) is chosen as to enforce a clause: if one or more of the outermost joins (black edges) are deleted by choosing the corresponding $A$-tuple (orange square), then the minimum cover for the black subgraph is 2, otherwise 3.

We next reduce 3SAT to $\texttt{RES}(q^{\text{ac}}_{\text{chain}})$. Let $\psi$ be a 3CNF formula with $n$ variables $v_1, \ldots, v_n$ and $m$ clauses $C_1, \ldots, C_m$. Our reduction will map any such $\psi$ to a pair $(D_\psi, k_\psi)$ where $D_\psi$ is a database satisfying $q_{\text{chain}}$, and

$$\psi \in \text{3SAT} \quad \Leftrightarrow \quad (D_\psi, k_\psi) \in \texttt{RES}(q^{\text{ac}}_{\text{chain}})$$

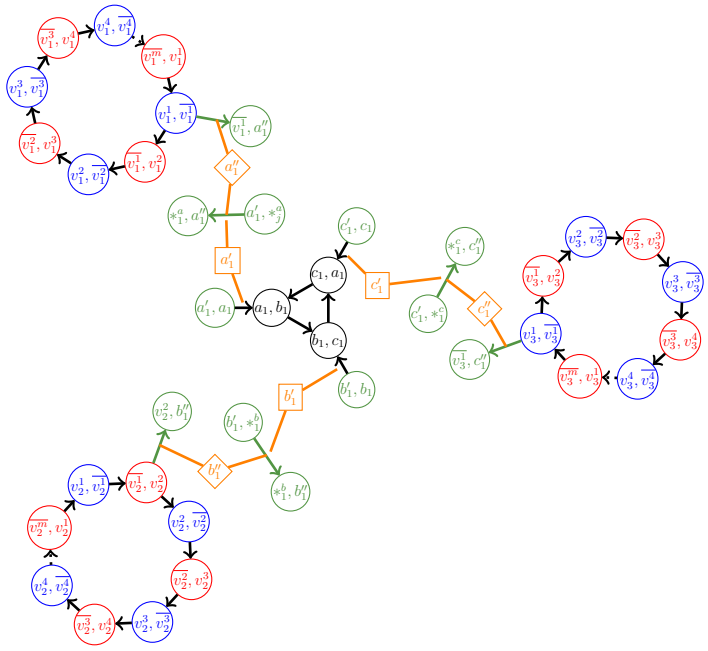


Figure 4.7: Excerpt from the construct showing the gadget for clause $C_1 = (v_1 \vee \bar{v}_2 \vee v_3)$ in the hardness proof for $q^{\text{ac}}_{\text{chain}}$. We omit the $A$-tuples and $C$-tuples that would not be chosen for a minimum contingency set.

In our construction, if $\psi \in$ 3SAT, then the size of each minimum contingency set for $q^{\text{ac}}_{\text{chain}}$ in $D_\psi$ will be $k_\psi = (n+5)m$, whereas if $\psi \notin$ 3SAT, then the size of all contingency sets for $q^{\text{ac}}_{\text{chain}}$ in $D_\psi$ will be greater than $k_\psi$.

1. Variable gadget: For each variable $v_i$ and each $j \in [m]$ insert the following tuples into the database: $R(v_i^j, \overline{v_i^j})$, $R(\overline{v_i^j}, v_i^{j+1})$ and $A(v_i^j)$, $A(\overline{v_i^j})$ and $C(v_i^j)$, $C(\overline{v_i^j})$. If $j+1 > m$, then make the superscript 1. The resulting joins between the tuples form a cycle of length $2m$. The minimum contingency sets are to either choose all tuples $R(v_i^j, \overline{v_i^j})$ representing a variable to have assignment `true`, or all tuples $R(\overline{v_i^j}, v_i^{j+1})$ representing a variable to have assignment `false`. If we only consider those tuples, note that $A$- and $C$-tuples participate in only one join, so the optimal choice is to delete $R$-tuples.
2. Clause gadget: For each clause $j \in [m]$ insert the following tuples into the database: $R(a_j, b_j)$, $R(b_j, c_j)$, $R(c_j, a_j)$, $R(a'_j, a_j)$, $R(b'_j, b_j)$, $R(c'_j, c_j)$, $A(a_j)$,

$A(b_j)$, $A(c_j)$, $A(a'_j)$, $A(b'_j)$, $A(c'_j)$, $C(a_j)$, $C(b_j)$, $C(c_j)$. The resulting joins form a triangle. If either of the $A(*')$ is removed, then the remaining joins can be destroyed by choosing only 2 or more tuples, otherwise we need 3. We later argue that these tuples only need be $R$-tuples.

3. Connecting the gadgets: For each variable $i$ that appears in clause $j$ at position 1, add the following tuples: $R(a'_j, *^a_j), R(*^a_j, a''_j)$ and $C(a''_j)$. If $v_i$ appears as positive add tuple $R(\overline{v^j_i}, a''_j)$, if it appear as negative add tuple $R(v^j_i, a''_j)$. Analogously use $b'_j, b''_j$ or $c'_j, c''_j$ instead of $a'_j, a''_j$ for positions 2 and 3 instead of position 1.

With our gadget, if the clause cannot be satisfied, then we need to choose all the $C$-tuples (orange diamonds on Fig. 4.7), since we can delete two joins by doing deleting each. In that case, in order to delete the remaining joins we need to delete 3 tuples, namely the 3 black nodes in the triangle, resulting on the total deletion of 6 tuples.

We now need to argue that, besides the tuples depicted in Fig. 4.7, we do not need other $A$- or $C$-tuples for a minimum contingency set. Assume there is a tuple $t = A(d)$ in a min $\Gamma$. Given that $d \notin \{a'_j, b'_j, c'_j\}$, our construction guarantees there is only one $R$-tuple such that $t' = R(d, -)$, therefore we can have $\Gamma' = \Gamma - t + t'$, and $\Gamma'$ is also a minimum contingency set. Similarly, if there is a tuple $t = C(d)$ in $\Gamma$, and assuming $d \notin \{a''_j, b''_j, c''_j\}$, there is only one $R$-tuple $t' = R(-, d)$, and therefore the same follows. □

**Proposition 4.27.** $\texttt{RES}(q^{\text{abc}}_{\text{chain}})$ *is* NP*-complete.*

*Proof.* We define a reduction from 3SAT, using almost the same construction as the one in Proposition 4.26. We just add the appropriate $B$-tuples and show that there is a minimum contingency set that does not contain those.

Consider $D_\psi$ as initially defined in Proposition 4.26. Now we include the appropriate $B$-tuples:

1. Variable gadget: For each variable $v_i$ and each $j \in [m]$ insert the following tuples into the database: $B(v_i^j)$, $B(\overline{v_i^j})$
2. Clause gadget: For each clause $j \in [m]$ insert the following tuples into the database: $B(a_j)$, $B(b_j)$, $B(c_j)$.
3. Connecting the gadgets: For each variable $i$ that appears in clause $j$ at position 1, add tuple $B(*_j^a)$. Analogously $B(*_j^b)$ and $B(*_j^c)$ for positions 2 and 3, respectively.

By adding those $B$-tuples we obtain the same joins we saw in the reduction for $\texttt{RES}(q^{\text{ac}}_{\text{chain}})$. With this construction we guarantee that for any tuple $t = B(d)$, there is either only one tuple $R(d, -)$ or only one tuple $R(-, d)$, which means we can always choose one of those $R$-tuples instead and obtain another minimum contingency set without $B$-tuples. □

**Lemma 4.28.** *Let $q$ be a query such that its binary graph is isomorphic to a chain, with or without loops (unary atoms). If $q$ has self-joins with endogenous relations, then* $\texttt{RES}(q)$ *is* NP*-complete.*

*Proof.* Since we know there are endogenous self-joins, we can have the following cases:

**Case 1:** $q$ has unary relations $A(x), A(y)$. We can define a reduction from $q_{\text{vc}}$. Given a database $D$ we want to define a database $D'$ such that

$$(D, k) \in \texttt{RES}(q_{\text{vc}}) \Leftrightarrow (D', k) \in \texttt{RES}(q)$$

We can assume w.l.o.g. that there is no loop in $D, q_{\text{vc}}$, i.e., $D \not\models q_{\text{vc}}(a, a)$, for any $a$. For each atom $R_i(v_1, \ldots, v_k)$ occurring in $q$, we define

$$R_i = \left\{ (t(v_1, a, b), \ldots, t(v_k, a, b)) \;\middle|\; D \models q_{\text{vc}}(a, b) \right\}$$

where

$$t(v,a,b) \stackrel{\text{def}}{=} \begin{cases} a & \text{if } v = x \\ b & \text{if } v = y \\ \langle ab \rangle_v & \text{otherwise} \end{cases}$$

In other words, $x$ maps to $a$, $y$ maps to $b$, and any other variable $v$ maps to $\langle ab \rangle_v$.

Because the query is a chain and minimal, we can show this reduction does not introduce joins with new values, similar to what we have in Lemma 4.22. Therefore, we can argue there is a min $\Gamma$ that only uses $A$-tuples with domain values, and so $\Gamma$ for $D, q_{\text{vc}}$ is also contingency set for $D', q$, and vice-versa.

**Case 2:** $q$ has binary relations $R(x,y), R(z,w)$, and no $R$-atoms share variables. We can also define a reduction from $q_{\text{vc}}$, analogous to the one for case 1.

**Case 3:** $q$ has binary relations $R(x,y), R(y,z)$. We can define a reduction from $q_{\text{chain}}$.

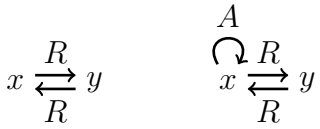
□

### 4.4.2 Permutations

Now we introduce the notion of permutations. We call a subquery with two binary atoms and two variables $R(x,y), R(y,x)$ a *permutation*. The name "permutation" indicates that the variable order $(x,y)$ is permuted between the two occurrences of the same relation. A simple example is the following query:

$$q_p :- R(x,y), R(y,x)$$

As we can see in its binary graph representation (Fig. 4.8a), the main difference in comparison to chains is that we have cycles connecting two consecutive nodes (variables). Note that if the cycle was formed by non-consecutive nodes, then the query would not be linear.




(a) $q_p$ (b) $q_p^2$

Figure 4.8: Easy permutations

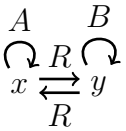



Figure 4.9: Query $q_{\text{perm}}$ is the smallest example of a query with permutations that is NP-complete

**Proposition 4.29.** $\texttt{RES}(q_p)$ *is in* P.

*Proof.* Given a database $D$ satisfying $q_p$, each tuple that is part of a witness for $e$ in $D$ is part of exactly one witness. Therefore the size of a minimum contingency set for $e$ in $D$ is exactly the number of witnesses. □

Let's now start augmenting query $q_p$ and see if and at what point adding more atoms will change the complexity. First example is the following (Fig. 4.8b):

$$q_p^2 :- A(x), R(x, y), R(y, x)$$

**Proposition 4.30.** $\texttt{RES}(q_p^2)$ *is in* P.

*Proof.* Given a database $D$ satisfying $q_p^2$, for each join $\langle a, b\rangle$, we have 2 possible choices. Either $A(a)$ will be in the min $\Gamma$ or one of $R(a, b)$ and $R(b, a)$ but never both. Therefore we can reduce $\texttt{RES}(q_p^2)$ to a network flow where we represent each $A$-tuple as an edge, and each pair $R(a, b), R(b, a)$ as one single edge $\{a, b\}$, since we never need to pick both $R$-tuples. □

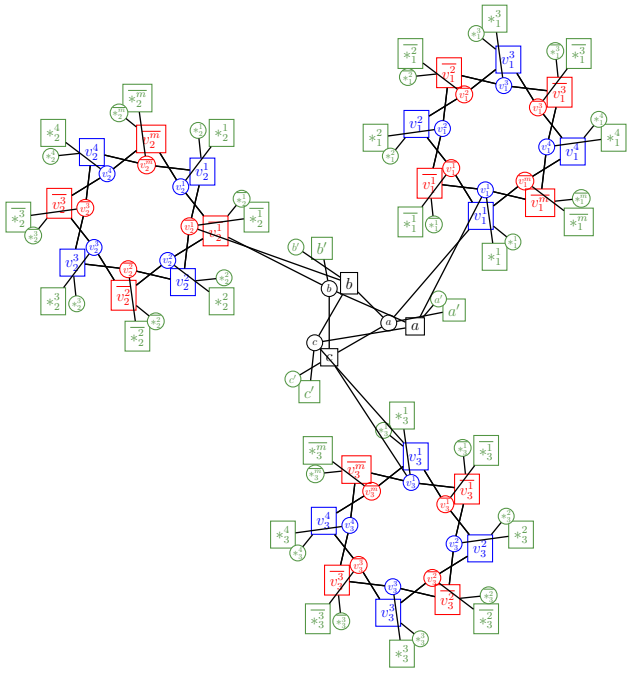


Figure 4.10: Excerpt from the construct showing the gadget for clause $C_1 = (v_1 \vee \bar{v}_2 \vee v_3)$ in the hardness proof for $q_{\text{perm}}$. Circles represent $A$-tuples and squares $B$-tuples. $R$-tuples are omitted as they can be inferred by the edges between circles and squares.

We will next give a hardness result for the simplest hard query containing a permutation (shown in Fig. 4.9):

$$q_{\text{perm}} :- A(x), R(x, y), R(y, x), B(y)$$

**Proposition 4.31.** $\texttt{RES}(q_{\text{perm}})$ *is* NP*-complete.*

*Proof.* We define a reduction from 3SAT to $\texttt{RES}(q_{\text{perm}})$. Similar to the previous cases, we want to create variable gadgets such that a minimum cover will choose either blue nodes (variable is set to `true`), or red nodes (variable is set to `false`), and a clause gadget (black nodes) such that if the clause is satisfied, then the minimum cover is 5, otherwise 6.

Let $\psi$ be a 3CNF formula with $n$ variables $v_1, \ldots, v_n$ and $m$ clauses $C_1, \ldots, C_m$. Our reduction will map any such $\psi$ to a pair $(D_\psi, k_\psi)$ where $D_\psi$ is a database satisfying $q_{\text{perm}}$, and

$$\psi \in \text{3SAT} \quad \Leftrightarrow \quad (D_\psi, k_\psi) \in \texttt{RES}(q_{\text{perm}})$$

In our construction, if $\psi \in \text{3SAT}$, then the size of each minimum contingency set for $q_{\text{perm}}$ in $D_\psi$ will be $k_\psi = (3n+5)m$, whereas if $\psi \notin \text{3SAT}$, then the size of all contingency sets for $q_{\text{chain}}$ in $D_\psi$ will be greater than $k_\psi$.

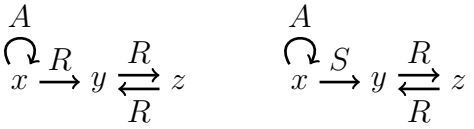
1. Variable gadget: For each variable $v_i$ and each $j \in [m]$ insert the following tuples into the database: $A(v_i^j)$, $B(v_i^j)$, $A(\overline{v_i^j})$, $B(\overline{v_i^j})$ and $R(v_i^j, \overline{v_i^j})$, $R(\overline{v_i^j}, v_i^j)$, $R(v_i^{j+1}, \overline{v_i^j})$, $R(\overline{v_i^j}, v_i^{j+1})$. If $j+1 > m$, then make the superscript 1.

   We want to join those tuples such that the minimum contingency sets are to either choose all tuples $A(v_i^j), B(v_i^j)$ representing a variable to have assignment `true`, or all tuples $A(\overline{v_i^j}), B(\overline{v_i^j})$ representing a variable to have assignment `false`, plus some $R$-tuples. To obtain that property, we need the following additional tuples: $A(*_i^j)$, $B(*_i^j)$, $A(\overline{*_i^j})$, $B(\overline{*_i^j})$ and $R(*_i^j, v_i^j)$, $R(v_i^j, *_i^j)$, $R(\overline{*_i^j}, \overline{v_i^j})$, $R(\overline{v_i^j}, \overline{*_i^j})$.

   With this construction we guarantee that we can "cover" the variable gadget by choosing either all positive $A, B$-tuples plus the $m$ tuples $R(\overline{*_i^j}, \overline{v_i^j})$, or all negative $A, B$-tuples plus the $m$ tuples $R(*_i^j, v_i^j)$. In both cases, we choose $3m$ tuples.

2. Clause gadget: For each clause $j \in [m]$ insert the following tuples into the database: $A(a_j)$, $B(a_j)$, $A(b_j)$, $B(b_j)$, $A(c_j)$, $B(c_j)$, $R(a_j, b_j)$, $R(b_j, a_j)$, $R(b_j, c_j)$, $R(c_j, b_j)$, $R(c_j, a_j)$, $R(a_j, c_j)$ and $A(a'_j)$, $B(a'_j)$, $A(b'_j)$, $B(b'_j)$, $A(c'_j)$, $B(c'_j)$, $R(a_j, a'_j)$, $R(a'_j, a_j)$, $R(b_j, b'_j)$, $R(b'_j, b_j)$, $R(c_j, c'_j)$, $R(c'_j, c_j)$ and

   For this gadget, we have 3 options to choose only 5 tuples in order to delete all the joins. For example: $A(a_j)$, $B(a_j)$, $A(b_j)$, $B(b_j), R(c_j, c'_j)$.




(a) $q_{ar-p}$ (b) $q_{as-p}$

Figure 4.11: Example of queries with unbounded permutation that are in PTIME

3. Connecting the gadgets: For each variable $i$ that appears in clause $j$ at position 1, add the following tuples: $R(v_i^j, a_j), R(a_j, v_i^j)$ if $v_i$ appears as positive, and $R(\overline{v_i^j}, a_j), R(a_j, \overline{v_i^j})$ if it appear as negative. Analogously use $b_j$ or $c_j$ instead of $a_j$ for positions 2 and 3 instead of position 1.

After connecting the variable gadgets with the clause gadgets, the joins are formed such that if a clause cannot be satisfied, then we need to pick all $A$- and $B$-tuples from the clause gadget (the black triangle), totaling 6 tuples. Otherwise, we can delete all joins by picking 5 tuples, namely 2 pairs of $A, B$-tuples and one $R$-tuple. □

The main difference between queries $q_p, q_p^2$ and $q_{\text{perm}}$ is the presence of unary atoms. In $q_{\text{perm}}$ both nodes of the permutation are bounded, whereas in the other two it is not. We say that the presence of bounded permutations can lead to hardness.

**Lemma 4.32** (Bounded permutation)**.** *Let $q$ be a linear query that contains a bounded permutation with $R$ and there is no other permutation with $R$ in $q$. Then* $\texttt{RES}(q)$ *is* NP*-complete.*

*Proof sketch.* We can show that $\texttt{RES}(q_{\text{perm}}) \leq \texttt{RES}(q)$. The restriction of not having other permutations with $R$ guarantees that no new joins will be created when we use the padding strategy for the reduction. □

Now we have two other cases, when the permutation is unbounded, and when it is bounded only on one side. First consider the following queries:

$$q_{as-p} :- A(x), S(x,y), R(y,z), R(z,y)$$

$$q_{ar-p} :- A(x), R(x,y), R(y,z), R(z,y)$$

We can see in Fig. 4.11 that in both queries the permutation is unbounded. However in Fig. 4.11a, the permutation connects to an unary atom $A$ through an atom $R$. Both queries are in P, but we need different variations of network flow to compute the minimum contingency set.

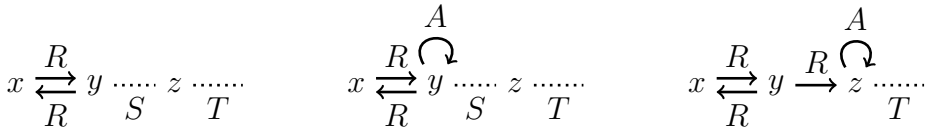
**Proposition 4.33.** $\texttt{RES}(q_{as-p})$ *is in* P.

*Proof.* First observe that $S$ is exogenous because is dominated by $A$. It is easy to see that we can construct a network flow similar to the one we defined in Proposition 4.30, with edges for $A$-tuples and one single edge for each pair $R(a,b), R(b,a)$. □

**Proposition 4.34.** $\texttt{RES}(q_{ar-p})$ *is in* P.

*Proof.* Let $D$ be a database such that $D \models q_{ar-p}$. We refer to tuples $R(a,b)$ as 2-way tuples if $R(b,a)$ is also in $D$, and as 1-way tuples if $R(b,a)$ is not. We construct a flow graph by creating 1-weight edges for each tuple $A(a)$, and edges $\{a,b\}$ for pairs of 2-way tuples. There are $\infty$-weight edges between an $A(x)$-edge and a $\{u,v\}$-edge if and only if $x \in \{u,v\}$ or there is a 1-way tuple $R(x,u)$ or $R(x,v)$. Note that 1-way tuples are exogenous, since we can always pick an $A$-tuple instead, so we do not need to include them in the flow graph. Suppose an $\{a,b\}$-edge is in the min-cut. Since those represent 2 tuples, we must decide which of $R(a,b)$ or $R(b,a)$ we should pick. If $A(a) \in D - \Gamma$, then choose $R(a,b)$, otherwise choose $R(b,a)$.

Now we show that $\Gamma$ obtained in this way is a minimum contingency set. Since each path from source to target in the network flow represents a join, we know that $\Gamma$ will "cut" those paths and therefore delete those joins. Now consider the following join that was not represented in the flow graph because $R(a,b)$ is a 2-way tuples: $\langle a,b,c\rangle = A(a), R(a,b), R(b,c), R(c,b)$. Since $R(a,b)$ is a 2-way tuple, we know the




(a) $q_p$ as endpoint (b) $q_p^2$ as endpoint (c) $q_{ar-p}$ as endpoint

Figure 4.12: Easy patterns with permutations

join $\langle a, b, a\rangle = A(a), R(a, b), R(b, a), R(a, b)$ exists and is represented in the flow graph, so either $A(a)$ or $\{a, b\}$ appears in the cut. If $A(a)$ appears in the cut, the join $\langle a, b, c\rangle$ is also deleted. If $\{a, b\}$ is chosen and $A(a)$ is not, then we know choose tuple $R(a, b)$ from the pair, also deleting join $\langle a, b, c\rangle$. □

We can generalize those cases in the following lemma:

**Lemma 4.35.** *Let $q$ be a linear query that contains as one of its endpoints one of the following patterns:*

1. $q_p :- R(x, y), R(y, x)$
2. $q_p^2 :- A(x), R(x, y), R(y, x)$
3. $q_{ar-p} :- A(x), R(x, y), R(y, z), R(z, y)$

*If $q$ does not have any other self-join except the ones we see above, then* $\texttt{RES}(q)$ *is in* PTIME.

*Proof.* Since the query is linear and we restrict the self-joins to be only the ones appearing in the pattern, we can easily extend the network flow construction for the entire query based on the flow for each of the cases (see Proposition 4.29 for case 1, Proposition 4.30 for case 2, and Proposition 4.34 for case 3). Because there is no other self-join in the query, the algorithm will give as solution the optimal contingency set. □

As an illustration of how those queries might look like see Fig. 4.12. Note that for case 3 (Fig. 4.12c), we need to have the $A$-loop, otherwise we cannot guarantee the construction will work. For example, consider

$$q :- A(w, x), R(x, y), R(y, z), R(z, y)$$

We conjecture $\texttt{RES}(q)$ is in PTIME but we do not know how to construct the network flow for this case.

#### 4.4.2.1 Other hard cases with permutation

In this section, we will see various case of hard queries that contain permutations, both bounded and unbounded. First let's look into some unbounded permutation case (Fig. 4.13):

$$q_{s-p-t} :- A(x), S(x, y), R(y, z), R(z, y), T(z, w), C(w)$$
$$q_{r-p-r} :- A(x), R(x, y), R(y, z), R(z, y), R(z, w), C(w)$$

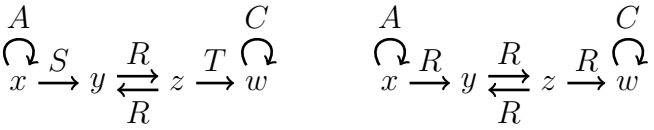
Both of those queries are NP-complete and here is a straightforward reduction from $\texttt{RES}(q_{\text{perm}})$.

**Proposition 4.36.** $\texttt{RES}(q_{s-p-t})$ *and* $\texttt{RES}(q_{r-p-r})$ *are* NP*-complete.*

*Proof.* We can show reductions from $\texttt{RES}(q_{\text{perm}})$. □

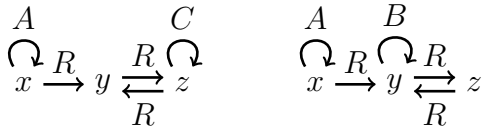
Now we will see some cases of bounded on one side that are hard. Consider queries (Fig. 4.14):

$$q_1 :- A(x), R(x, y), R(y, z), R(z, y), C(z)$$
$$q_2 :- A(x), R(x, y), B(y), R(y, z), R(z, y)$$




(a) $q_{s-p-t}$ (b) $q_{r-p-r}$

Figure 4.13: Unbounded hard permutation




(a) $q_1$ (b) $q_2$

Figure 4.14: Bounded on one side: hard cases

**Proposition 4.37.** $\texttt{RES}(q_1)$ *and* $\texttt{RES}(q_2)$ *are* NP*-complete.*

*Proof Sketch.* There is a reduction from $\texttt{RES}(q_{\text{perm}})$ to $\texttt{RES}(q_1)$. For $q_2$ we use a reduction from Max-2SAT. □

Contiguous permutation (Fig. 4.15):

$$q_3 :- A(x), R(x,y), R(y,x), B(y), R(y,z), R(z,y), C(z)$$

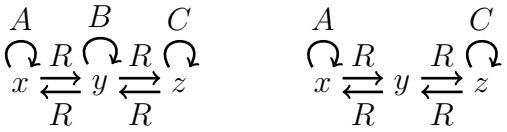
$$q_4 :- A(x), R(x,y), R(y,x), R(y,z), R(z,y), C(z)$$

**Proposition 4.38.** $\texttt{RES}(q_3)$ *and* $\texttt{RES}(q_4)$ *are* NP*-complete.*

*Proof Sketch.* There is a reduction from $q_{\text{perm}}$ to $q_3$. For $q_4$ we use a reduction from Max-2SAT. □

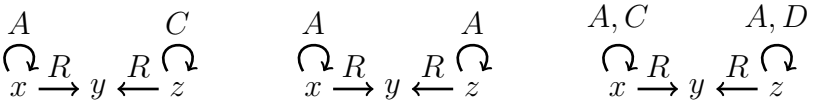
### 4.4.3 Confluences

The third structure we are going to present is what we call confluence. Two atoms form a *confluence* if they refer to the same relation, i.e. form a self-join, and share a variable in the same position.




(a) $q_3$ (b) $q_4$

Figure 4.15: Contiguous permutations




(a) $q_{\text{conf}}$ (b) $q^a_{\text{conf}}$ (c) $q^2_{\text{conf}}$

Figure 4.16: Examples of easy queries with confluences

The simplest minimal query with a confluence is (Fig. 4.16a):

$$q_{\text{conf}} :- A(x), R(x, y), R(z, y), C(z)$$

**Proposition 4.39.** *$R$ is exogenous in $q_{\text{conf}}$.*

*Proof.* Let $\Gamma$ be a minimum contingency set containing tuple $R(1, 2)$.

*Case 1:* $D$ contains only $A(1)$ or $C(1)$ but not both. WLOG, suppose it contains only $A(1)$. We can then obtain a contingency set $\Gamma' = (\Gamma - R(1, 2)) \cup A(1)$ of size $k$. Similar if it contains only $C(1)$.

*Case 2:* $D$ contains both $A(1)$ and $C(1)$. Consider $\Gamma' = (\Gamma \cup A(1)) - R(1, 2)$ and $\Gamma'' = (\Gamma \cup C(1)) - R(1, 2)$, and suppose that neither of those is a contingency set. Then we have $A(i), R(i, 2), R(1, 2), C(1)$ in $D - \Gamma'$ and $A(1), R(1, 2), R(j, 2), C(j)$ in $D - \Gamma''$, with $i \neq j$. However, the existence of those joins implies that $D - \Gamma$ has the join $A(i), R(i, 2), R(j, 2), C(j)$ contradicting the fact that $\Gamma$ is a contingency set. Therefore, at least one of $\Gamma', \Gamma''$ must be a contingency set and we can replace $R(1, 2)$ by $A(1)$ or $C(1)$. □

**Proposition 4.40.** $\texttt{RES}(q_{\text{conf}})$ *is in* P.

*Proof.* Since $R$ can be made exogenous, solving resilience for this query is the same as solving vertex cover in a bipartite graph, and therefore is in P. □

Proposition 4.39 is an example that our definition of domination, Definition 4.12, is a sufficient but not necessary condition for a relation to be exogenous.

Observe that if we had $A(z)$, instead of $C(z)$ in $q_{\text{conf}}$ (Fig. 4.16b), the query would not be minimal:

$$\begin{aligned} q^a_{\text{conf}} &:- A(x), R(x,y), R(z,y), A(z) \\ &\equiv A(x), R(x,y) \end{aligned}$$

However, we can add another unary atom to make the query minimal as we can see in the following (Fig. 4.16c):

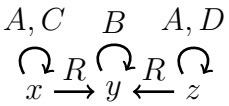
$$q^2_{\text{conf}} :- C(x), A(x), R(x,y), R(z,y), A(z), D(z)$$

**Proposition 4.41.** $\texttt{RES}(q^2_{\text{conf}})$ *is in* P.

*Proof.* First note that $C, D$ and $R$ are exogenous relations because they are dominated by $A$, so there is minimum contingency set with only $A$-tuples. If an $A$-tuple joins with itself, then we can guarantee this tuple will be in $\Gamma$. After including all such $A$-tuples in $\Gamma$, we can split the remaining ones as left and right tuples, meaning that they join with a $C$-tuple and a $D$-tuple, respectively. Since we have 2 partitions, we can compute the minimum vertex cover and add include that in $\Gamma$. □

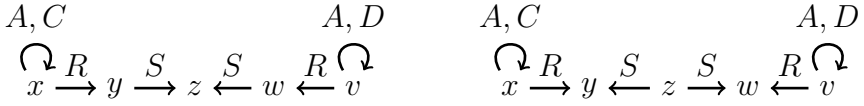
We also have queries with confluence that are NP-complete:

$$q^{2,b}_{\text{conf}} :- C(x), A(x), R(x,y), B(y), R(z,y), A(z), D(z)$$




(a) $q^{2,b}_{\text{conf}}$

Figure 4.17: Simple hard confluence




(a) $q^{r,s(1)}_{\text{conf}}$ (b) $q^{r,s(2)}_{\text{conf}}$

Figure 4.18: Example of hard queries with confluences

**Proposition 4.42.** $\texttt{RES}(q^{2,b}_{\text{conf}})$ *is* NP*-complete.*

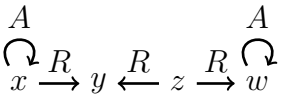
*Proof sketch.* Reduction from 3SAT. □

We can generalize Proposition 4.42 for any query that has a middle point such that the left and right sides, from the middle point, are symmetric and, when considered in isolation, each side has no self-joins. Let's look at the following example:

**Example 4.43.** *Consider queries (Fig. 4.18):*

$$q^{r,s(1)}_{\text{conf}} :- A(x), C(x), R(x,y), S(y,z), S(w,z), R(w,v), D(v), A(v)$$
$$q^{r,s(2)}_{\text{conf}} :- A(x), C(x), R(x,y), S(z,y), S(z,w), R(w,v), D(v), A(v)$$

*Taking $z$ as the middle point in both queries, we can see that the left and right side are symmetric. In other words, we could "fold" those queries at point $z$ and the edges would match, with the exception of the loops. We can show that* $\texttt{RES}(q^{r,s(1)}_{\text{conf}})$ *and* $\texttt{RES}(q^{r,s(2)}_{\text{conf}})$ *are both* NP*-complete with a reduction from* $\texttt{RES}(q^{2,b}_{\text{conf}})$*. It is important to note that we must have atoms $A(x), C(x)$ and $A(w), D(w)$ together. If the query did not have $C, D$, then it would not be minimal. On the other hand, without the $A$, we do not how to define the reduction from* $\texttt{RES}(q^{2,b}_{\text{conf}})$*.*




(a) $q^3_{\text{conf}}$

Figure 4.19: Hard query with 2 confluences

**Lemma 4.44.** *Let $q$ be a query such that there exists a middle point that split the query in two symmetric ones. Then* $\texttt{RES}(q)$ *is* NP*-complete.*

*Proof sketch.* We can always define a reduction from $\texttt{RES}(q^{2,b}_{\text{conf}})$ to $\texttt{RES}(q)$. □

In certain cases where the query has confluences but there is no middle point, we can still show hardness. Consider query (Fig. 4.19)

$$q^3_{\text{conf}} :- A(x), R(x,y), R(z,y), R(z,w), A(w)$$

Note that $q^3_{\text{conf}}$ has no middle point that can split the query in two symmetric ones. We say then that there is a non-symmetric path between $A(x)$ and $A(w)$. In the general case we have:

**Lemma 4.45.** *Let $q$ be a minimal linear query with two atoms $A(x)$, $A(y)$, which are the endpoints in its binary graph, and there is a non-symmetric path between them. Then* $\texttt{RES}(q)$ *is* NP*-complete.*

*Proof sketch.* We can define a reduction from $\texttt{RES}(q_{\text{vc}})$ to $\texttt{RES}(q)$. □

## 4.5 Dichotomy conjecture

In previous sections we have shown various cases of binary queries without variable repetition, and for which ones we know how to characterize complexity, either NP-complete and PTIME. This knowledge allowed us to build a better intuition about this fragment of the problem and to conjecture that there is a NP vs P dichotomy.

**Conjecture 4.46.** *Let $q$ be a self-join binary query without variable repetition that is minimal and connected. If at least one of the following is true, then* $\texttt{RES}(q)$ *is* NP*-complete. Otherwise, it is in* PTIME*.*

1. *$q$ has a triad;*
2. *$q$ has a chain;*
3. *$q$ has a hard permutation;*
4. *$q$ has a hard confluence.*

Each of the cases for hardness above relates to results or conjectures we have made throughout this chapter. Case 1 refers to the non-linear queries we saw in Section 4.3 and how we needed different strategies to show hardness but, essentially, having a triad implied hardness. Case 2 refers to Section 4.4.1, in particular, Lemma 4.28. The idea is that if a linear query only have chain-like connections, that query will be hard even if it doesn't have $q_{\text{chain}}$ as a subquery. Case 3 includes the hard permutation cases we saw in Section 4.4.2. When we say a query has a hard permutation, we mean that the query contains a permutation that is connected on both sides to relations different than the ones participating on the permutation itself. Similarly, case 4 refers to the hard confluences we saw in Section 4.4.3.

## 4.6 A special case: R, R queries

In this section we analyze the simple case of queries with only two atoms but unbounded arity. As we are considering self-joins, both atoms will refer to the same relation. We will see that, even in this simple case, some of the notions we explored on the binary case already apply, like variable order and repetition.

We define $\text{set}(\mathbf{x}) = \{x_1, x_2, \ldots, x_k\}$ to be the set of variables that occur in tuple $\mathbf{x}$. The following lemmas simplify the queries in such a way that preserves certain complexity properties.

**Lemma 4.47** (Eliminate "matching" variables)**.** *Consider* $q :- R(w, \mathbf{x}), R(w, \mathbf{y})$ *such that* $w \notin \text{set}(\mathbf{x}) \cup \text{set}(\mathbf{y})$*, and* $q' :- R'(\mathbf{x}), R'(\mathbf{y})$*. Then* $\texttt{RES}(q) \equiv \texttt{RES}(q')$*.*

*Proof.* We can reduce $\texttt{RES}(q')$ to $\texttt{RES}(q)$ by defining a new database $D' = \{(a_0, t) \mid t \in D\}$. Adding an extra column with a fixed constant value, $a_0$, does not significantly change the set of witnesses. Thus, the size of the minimum contingency set is preserved. In the opposite direction, given a database $D$ we can partition the tuples accordingly to the value of its first attribute. Let $R' = \{(t_1^a, \ldots, t_k^a) \mid \exists a R(a, t_1, \ldots, t_k) \in D\}$. Then the size of a minimum contingency set for $q$ in $D$ is equal to the size of a minimum contingency set for $q'$ in $D' = (R')$. □

Note that the above lemma is a special case of isolated variables (Definition 4.8), therefore we can prove equivalence instead of only one direction reduction.

**Lemma 4.48** (Eliminate repeated columns)**.** *Consider* $q :- R(w_1, w_1, \mathbf{x}), R(w_2, w_2, \mathbf{y})$ *and* $q' :- R'(w_1, \mathbf{x}), R'(w_2, \mathbf{y})$*. Then* $\texttt{RES}(q) \equiv \texttt{RES}(q')$*.*

*Proof.* For database $D = (R)$ satisfying $q$ we can construct $D' = (R')$ satisfying $q'$ with the same size minimum contingency set by letting $R'$ be the projection of $R$ on all columns except the first. Similarly, we can go from $R'$ to $R$ by adding a copy of the first column: $R := \{(a_1, a_1, a_2, \ldots, a_k) \mid (a_1, a_2, \ldots, a_k) \in R'\}$. Again, the size of the minimal contingency set for $q$ in $D$ is the same as for $q'$ in $D'$. □

Now we define the concept of unique variables. The presence or absence of unique variables plays an important role in our analysis but we do not know if they will be central to our final dichotomy result.

**Definition 4.49** (Unique variables)**.** *Consider query* $q :- R(\mathbf{v}_1), R(\mathbf{v}_2)$*. We say that a variable* $x$ *is* unique *to the first atom if* $x \in \text{set}(\mathbf{v}_1)$ *and* $x \notin \text{set}(\mathbf{v}_2)$*. Analogously* $y$ *is unique to the second atom if* $y \in \text{set}(\mathbf{v}_2)$ *and* $y \notin \text{set}(\mathbf{v}_1)$*.*

Even though unique variables will be important in our analysis, we would like to eliminate them when they are isolated.

**Example 4.50.** *Consider query* $q :- R(x, w, y)R(y, k, z)$*. If we look closer, we can note that this query is similar to query* $q_{\text{chain}}$*, they only differ by the "extra" column in the middle, in other words,* $w, k$ *are isolated variables. By deleting those variables we obtain exactly* $q_{\text{chain}}$*, thus showing* $\texttt{RES}(q)$ *is* NP*-complete via Lemma 4.9.*

The following proposition shows that if the atoms have no unique variable, i.e. a variable occurring in the first atom must occur in the second atom and vice-versa, then the problem is easy. This case generalizes query $e_2$.

**Proposition 4.51.** *Let* $q :- R(\mathbf{x}), R(\mathbf{y})$ *be such that* $\text{set}(\mathbf{x}) = \text{set}(\mathbf{y})$*. Then* $\texttt{RES}(q)$ *is in* PTIME*.*

*Proof.* Let $D = (R)$ be any database satisfying $q$. Observe that if $R(\mathbf{s})R(\mathbf{t})$ is any witness of $q$ in $D$, then $\text{set}(\mathbf{s}) = \text{set}(\mathbf{t})$, that is exactly the same domain values occur in these two tuples. Put another way, if $\text{set}(\mathbf{s}) \neq \text{set}(\mathbf{t})$, then $\mathbf{s}$ and $\mathbf{t}$ come from different connected components of the join graph of $q$ on $D$. Notice, furthermore, that for any $\mathbf{s}$, the number of tuples, $\bar{t}$, such that $\text{set}(\mathbf{s}) = \text{set}(\mathbf{t})$ is at most the fixed constant $k!$, where $k$ is the arity of $R$. Thus, we can compute a minimum contingency set for $q$ over $D$, by computing the minimum contingency set for the tuples from each $\text{set}(\mathbf{s})$ independently. That is $\texttt{RES}(q)$ consists of polynomially many independent constant-size problems. □

Now we investigate the case where there are unique variables.

**Proposition 4.52.** *Let* $q :- R(\mathbf{v}_1), R(\mathbf{v}_2)$ *be a query without isolated variables such that the following holds:*

1. *There exists* $x$ *unique to* $\mathbf{v}_1$ *and* $z$ *unique to* $\mathbf{v}_2$
2. *There is* $y$ *shared by* $\mathbf{v}_1$ *and* $\mathbf{v}_2$

*3. There is no variable repetition*

*Then* $\texttt{RES}(q)$ *is* NP*-complete.*

*Proof.* The queries described here are a generalization of query $q_{\text{chain}}$. As we show in Proposition 4.2, $\texttt{RES}(q_{\text{chain}})$ is hard via a reduction from 3SAT, but here we are going to use a reduction from vertex cover. We define the reduction $\texttt{RES}(q_{\text{vc}}) \leq \texttt{RES}(q)$ as follows:

Given $D = (A, R)$, let $D' = (R')$ and $r$ be the arity of $R'$. If $r$ is odd we have

$$\begin{aligned} R' \quad = \quad & \big\{(a_1, a_2, \ldots, a_r) \;\big|\; a \in A\big\} \cup \\ & \big\{(a_2, a_3, \ldots, a_r, b_1), (a_3, \ldots, a_r, b_1, b'), \ldots, (a_r, b_1, \ldots, b_{r-1}) \;\big|\; (a, b) \in R\big\} \end{aligned}$$

If $r$ is even, we need to add an extra tuple to have an even number of intermediate tuples between $(a_1, \ldots, a_r)$ and $(b_1, \ldots, b_r)$.

$$\begin{aligned} R' \quad = \quad & \big\{(a_1, a_2, \ldots, a_r) \;\big|\; a \in V\big\} \cup \\ & \big\{(a_2, a_3, \ldots, a_r, \langle ab\rangle), (a_3, \ldots, a_r, \langle ab\rangle, b_1), \ldots, (\langle ab\rangle, b_1, \ldots, b_{r-1}) \;\big|\; (a, b) \in R\big\} \end{aligned}$$

We claim that

$$(D, k) \in \texttt{RES}(q_{\text{vc}}) \quad \Leftrightarrow \quad (D', k_r|E| + k) \in \texttt{RES}(q),$$

where $k_r = \lfloor r/2 \rfloor$.

($\Rightarrow$) A contingency set for $D$ is a vertex cover $S$. We will construct a contingency set $\Gamma$ for $D'$ based on the vertex cover $S$. For each vertex $a \in S$, add $R'(a_1, \ldots, a_r)$ to $\Gamma$. As $S$ is a cover, by definition, for each edge $(a, b)$ in $D$ we have $R'(a_1, \ldots, a_r)$ or $R'(b_1, \ldots, b_r)$ in $\Gamma$. Assume w.l.o.g. that $R'(a_1, \ldots, a_r) \in \Gamma$. If $r$ is odd we need to add tuples $R'(a_i, \ldots, b_j)$ when $i$ is odd and there are exactly $(r-1)/2$ of those. If $r$

is even we need to add tuples $R'(a_i, \ldots, b_j)$ when $j$ is odd and there are exactly $r/2$ of those. Suppose that $\Gamma$ we obtained is not a contingency set. That means there are tuples $t_1, t_2$ that joint under $D' - \Gamma$. For $t_1$ and $t_2$ to join, by our construction, they would need to be associated with the same edge, let's say $(c, d)$. As $S$ was a cover, one of the vertices $c$ or $d$ is in $S$, and therefore one of $R'(c_1, \ldots, c_r)$ or $R'(d_1, \ldots, d_r)$ is in $\Gamma$. However, if that is the case, we eliminated all the joins that occur "within" that edge by including half of the tuples associated with edge $(c, d)$. Thus, $\Gamma$ is a contingency set of the appropriate size.

($\Leftarrow$) Let $\Gamma$ be a contingency set for $D'$ with $k + k_r|E|$. First note that, with our construction, we always have an odd number of edges $l$ between a node $R'(a_1, \ldots, a_r)$ and $R'(b_1, \ldots, b_r)$, which means that we need $l/2 + 1$ vertices to cover all the edges. Even though locally we could do that by only choosing internal nodes, with a global perspective, choosing at least one of the extremities is either the same or better. We can say then that for each edge in $D$ we will pick $l/2$ internal edges in $D'$ which gives us $k_r|E|$ vertices. The $k$ tuples left to be picked must be from extremities. Let $S$ be a set formed by the nodes which correspond to the last $k$ tuples in $\Gamma$ and suppose $S$ is not a cover. Then there is an edge $(c, d)$ such that $c, d \notin S$. If that is the case then there is at least one edge (join) left in $D - \Gamma$, which is a contradiction with the fact that $\Gamma$ is a contingency set. □

The property of queries we are capturing with Proposition 4.52 is a notion of path connecting the attributes of each atom. In the following example this will be more clear.

**Example 4.53.** *Let's look into a query we already know,* $q_{\text{chain}} :- R(x, y), R(y, z)$. *In this query, we have a unique variable occurring in the first atom,* $x$, *and a unique variable occurring in the second one,* $z$, *and those queries are connected by variable*

*y. We could augment this query by keeping the unique ones and including more intermediate "steps" as in the following query:*

$$q_{\text{k-chain}} :- R(x, y_1, y_2, \ldots, y_k), R(y_1, \ldots, y_{k-1}, y_k, z)$$

*Query* $q_{\text{k-chain}}$ *satisfies the conditions described on the Proposition 4.52 and, therefore,* $\texttt{RES}(q_{\text{k-chain}})$ *is* NP*-complete.*

# CHAPTER 5

# COMPLEXITY OF RESPONSIBILITY FOR SJ-FREE QUERIES

In this chapter, we prove the analogous characterizations of the complexity of responsibility. As we will see, responsibility is a bit more delicate than resilience; yet, in the end, the final theorems are similar.

We first concentrate on the difference between resilience and responsibility. Recall the following two queries:

$$q_{\text{rats}} :- A(x), R(x,y), S(y,z), T(z,x)$$
$$q'_{\text{rats}} :- A(x), R^{\text{x}}(x,y), S(y,z), T^{\text{x}}(z,x)$$

We saw earlier that $\texttt{RES}(q_{\text{rats}})$ is in PTIME (Corollary 3.22). The reason is that atom $A$ dominates $R$ and $T$ and thus the complexity of $\texttt{RES}(q_{\text{rats}})$ is unchanged when we make $R$ and $T$ exogenous (Proposition 3.9), i.e., $\texttt{RES}(q_{\text{rats}}) \equiv \texttt{RES}(q'_{\text{rats}})$. Obviously $q'_{\text{rats}}$ is triad-free. Thus, by Theorem 3.24, $\texttt{RES}(q'_{\text{rats}})$ and $\texttt{RES}(q_{\text{rats}})$ are in PTIME. We now show, however, that $\texttt{RSP}(q_{\text{rats}})$ is NP-complete.

**Proposition 5.1.** $\texttt{RSP}(q_{rats})$ *is* NP*-complete.*

*Proof.* We reduce 3SAT to $\texttt{RSP}(q_{\text{rats}})$. Let $\psi$ be a 3-CNF formula with variables $v_1, \ldots, v_n$ and clauses $C_1, \ldots, C_m$. The reduction will map $\psi$ to $f(\psi) = (D, \mathbf{s_0}, k)$ with $\mathbf{s_0} = S(b_0, c_0)$, where we will construct $D = (A, R, S, T)$ to have a contingency set for $\mathbf{s_0}$ of size $k$ iff $\psi \in$ 3SAT (we explain the choice of value $k$ later in the proof). We let $a_0$ be the unique element of the domain of $D$ that joins with $\mathbf{s_0}$.

In $q_{\text{rats}}$, $A$ dominates $R$, but when we are building a contingency set $\Gamma$ for $\mathbf{s_0}$, we may require some tuples of the form $R(a_0, b)$. Note that these cannot be replaced by the tuple $A(a_0)$, because that would remove the only witness $(a_0, b_0, c_0)$ that contains our tuple $\mathbf{s}_0$. This explains why $\texttt{RES}(q_{\text{rats}}) \in \mathsf{PTIME}$ while $\texttt{RSP}(q_{\text{rats}})$ is $\mathsf{NP}$-complete, and it is the key idea behind the reduction we now produce.

For each variable $v_\ell$ occurring in $\psi$, we build the gadget $G_\ell$ as follows: $G_\ell$ consists of $2t$ $b_j^\ell$ values for $y$ and $2t$ $c_j^\ell$ values for $z$ ($1 \leq j \leq 2t$) where $t$ is a constant to be specified later. We include the $2t$ pairs $R(a_0, b_j^\ell)$ and the $2t$ pairs $T(c_j^\ell, a_0)$, $1 \leq j \leq 2t$. (See Fig. 5.1 where these pairs are drawn as edges from $a_0$ to each $b_j^\ell$ and from each $c_j^\ell$ to $a_0$, respectively. Notice that the value $a_0$ is shown twice for better illustration.)

Next, we include all the pairs $S(b_j^\ell, c_{j'}^\ell)$, $1 \leq j, j' \leq t$. These are drawn in Fig. 5.1 as a complete bipartite graph between the vertex sets $\{b_1^\ell, \ldots, b_t^\ell\}$ and $\{c_1^\ell, \ldots, c_t^\ell\}$.

Finally we add two matchings of size $t$ which we name the "$v_\ell$ matching" and the "$\overline{v_\ell}$ matching," respectively:

$$v_\ell \text{ matching}: \quad S(b_1^\ell, c_{t+1}^\ell), \ldots, S(b_t^\ell, c_{2t}^\ell)$$
$$\overline{v_\ell} \text{ matching}: \quad S(b_{t+1}^\ell, c_1^\ell), \ldots, S(b_{2t}^\ell, c_t^\ell)$$

Notice that in Fig. 5.1, the $v_\ell$ matchings are connecting the upper left corner with the lower right corner, whereas the $\overline{v_\ell}$ matchings are connecting the other two corners.

Any minimum contingency set must remove all of the witnesses from $G_\ell$. Such a minimum contingency set must remove either all the pairs $R(a_0, b_1^\ell), \ldots R(a_0, b_t^\ell)$ or all the pairs $T(c_1^\ell, a_0), \ldots T(c_t^\ell, a_0)$, i.e., one side or the other of the complete bipartite graph. After this, $t$ witnesses remain, either involving the $v_\ell$ matching (if the $T(c_i^\ell, a_0)$'s were chosen), or otherwise the $\overline{v_\ell}$ matching. Only the $S$-tuples will be useful for the clause gadgets, so the optimal choice will be to choose the $t$ $S$-tuples marked $v_\ell$ or the $t$ $S$-tuples marked $\overline{v_\ell}$. Any optimal minimal contingency set thus corresponds to a truth assignment to the boolean variables $v_1, \ldots, v_n$.

So far, we have described the gadgets $G_1, \ldots G_n$ and shown that any minimum contingency set for this part of $D$ corresponds to a truth assignment for the variables $v_1, \ldots, v_n$. We next introduce the clause gadgets and choose the value $k$, so that contingency sets for $D$ of size $k$ will correspond exactly to truth assignments that satisfy all of the clauses of $\psi$.

We now describe the clause gadgets. Suppose, for example, that $C_s = v_1 \vee \overline{v_2} \vee v_3$ with $s \in [m]$. Then 7 of the eight possible truth assignments to $v_1, v_2, v_3$ satisfy $C_s$, i.e., all but the assignment $\alpha_2$ (010 in binary). For each of these 7 good assignments: $\alpha_i$, $i \in \{0, \ldots 7\} - \{2\}$, we add an element $a_{s,i}$ to $A$ and we add the tuples to $R$ and $T$ so that $a_{s,i}$ participates in three witnesses, each of which shares an $S$ tuple with a witness from each of the three variable gadgets that agree with assignment $\alpha_i$. For example, assignment $\alpha_6$ (110 in binary) makes $v_1, v_2$ true and $v_3$ false, so $a_{s,6}$ joins with $S(b^1_{r(s,6)}, c^1_{t+r(s,6)})$, $S(b^2_{r(s,6)}, c^2_{t+r(s,6)})$, and $S(b^3_{t+r(s,6)}, c^3_{r(s,6)})$. Here $r(s,i)$ is a function that chooses a unique element of the matching $v_j$ or $\overline{v_j}$ appropriate to assignment $\alpha_i$ of clause $s$ (see Fig. 5.2).

The key property of the $C_s$ gadget is that, if the chosen truth assignment satisfies $C_s$, then we do not need to worry about the $a_{s,i}$ corresponding to the chosen assignment, and may choose only 6 $a_{s,i}$'s from $A$ for the contingency set. However, if the chosen assignment does not satisfy $C_s$, then all 7 of the $a_{s_i}$'s must be chosen!

We can let $t = 8m$ and $k = (2t)n + 6m = (16n + 6)m$. Our construction insures that $(D, \mathbf{s_0}, k) \in \texttt{RSP}(q_{\text{rats}})$ iff $\psi \in$ 3SAT. □

The proof of Proposition 5.1 shows that domination does not work the same way for responsibility as it does for resilience. In particular, the analogy of Proposition 3.9 (Domination for Resilience) does not hold for responsibility.

We next show that a modified version of domination still works for responsibility. Recall the query $q_{\text{brats}}$ and define the query $q_{\text{br}^x\text{ats}}$ as follows:

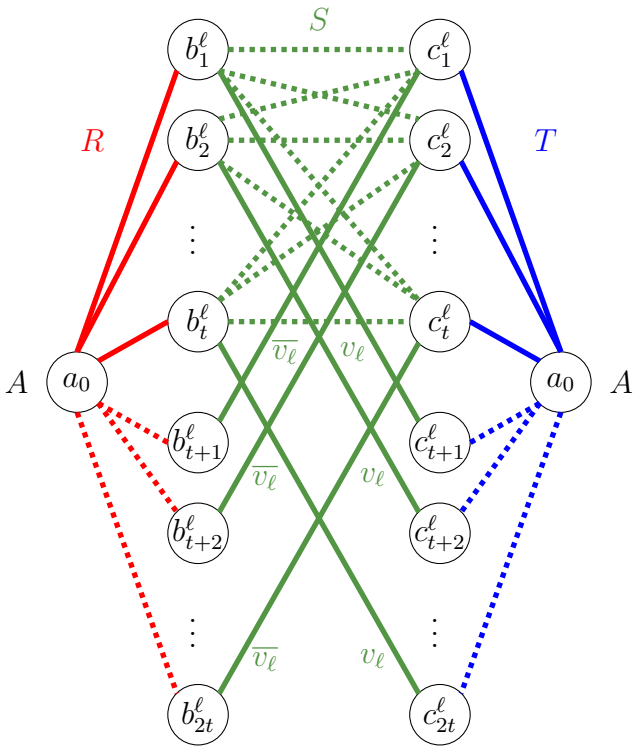



Figure 5.1: The $q_{\text{rats}}$ variable gadget $G_\ell$ for variable $v_\ell$. Red, green, and blue lines correspond to tuples from $R$, $S$, and $T$, respectively. Dotted lines will never need to be chosen in minimum contingency sets of $f(\psi)$.

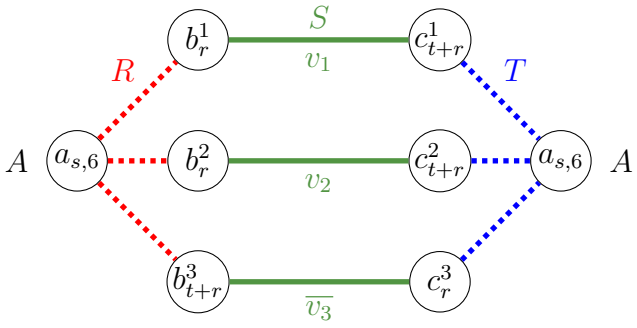



Figure 5.2: The $q_{\text{rats}}$ clause gadget corresponding to clause $C_s = v_1 \vee \overline{v_2} \vee v_3$ and truth assignment $\alpha_6 = \{\langle v_1, 1\rangle, \langle v_2, 1\rangle, \langle v_3, 0\rangle\}$. $A(a_{s,6})$ must be in the minimum contingency set unless the chosen truth assignment is $\alpha_6$.

$$q_{\text{br}^x\text{ats}} :- A(x), R^{\text{x}}(x,y), B(y), S(y,z), T(z,x)\ .$$

Notice that $\texttt{var}(A) \subset \texttt{var}(R)$ and $\texttt{var}(B) \subset \texttt{var}(R)$ and that also $\texttt{var}(R) \subseteq \texttt{var}(A) \cup \texttt{var}(B)$.

**Proposition 5.2** ($\texttt{RSP}(q_{\text{brats}})$)**.** *The complexity of responsibility for $q_{brats}$ is unchanged if we make $R$ exogenous, i.e.,* $\texttt{RSP}(q_{brats}) \equiv \texttt{RSP}(q_{br^x ats})$ .

*Proof.* Let $D \models q_{\text{brats}}$ and let $\mathbf{t}$ be a tuple that participates in a witness that $D \models q_{\text{brats}}$. We will show that there is a minimum contingency set $\Gamma'$ for $\mathbf{t}$ that contains no tuples from $R$. Let $\Gamma$ be a minimum contingency set for $\mathbf{t}$ that contains as few tuples from $R$ as possible. Suppose that $R(a_1, b_1) \in \Gamma$. Let $\mathbf{j}$ be a witness that $(D - \Gamma) \models q_{\text{brats}}$ and let $a_0, b_0, c_0$ be the projection of $\mathbf{j}$ onto components $x, y, z$, respectively. Thus, $A(a_0), R(a_0, b_0)$ and $B(b_0)$ are all in $D - \Gamma$. In particular, $R(a_1, b_1) \neq R(a_0, b_0)$. Let $\Gamma'$ be the result of replacing $R(a_1, b_1)$ by $A(a_1)$ if $a_1 \neq a_0$, and by $B(b_1)$ otherwise, in which case $b_1 \neq b_0$. Thus $\Gamma'$ is still a minimum contingency set for $\mathbf{t}$ and it contains fewer tuples from $R$, contradicting the fact that $\Gamma$ had the fewest possible such tuples. Thus, tuples from $R$ are never needed in any minimum contingency set for $\mathbf{t}$. Thus, as claimed, the complexity of $\texttt{RSP}(q_{\text{brats}})$ is unchanged when we make $R$ exogenous. □

We are now ready to formalize "*full domination*", the version of domination that works for responsibility the way that domination works for resilience.

For example, in the query $q_{\text{brats}}$, the relation $R$ is fully dominated because every variable in $\texttt{var}(R)$ is "covered" by some other endogenous relation (Proposition 5.2).[1] Here are three more examples where $R$ is fully dominated ($s_1, s_2, s_3$) and one where it is not ($n_4$):

[1] Contrast this with the definition of domination (Definition 3.8) which only requires that some subset of the variables is covered by another relation.

$$s_1 :- A(x), R(x,y,w), B(y), S(y,z), T(z,x)$$

$$s_2 :- A(x), R(x,y,w), Q^{\mathrm{x}}(w), B(y), S(y,z), T(z,x)$$

$$s_3 :- A(x), R(x,y,w), Q^{\mathrm{x}}(w,x), B(y), S(y,z), T(z,x)$$

$$n_4 :- A(x), R(x,y,w), Q^{\mathrm{x}}(w,z), B(y), S(y,z), T(z,x)$$

In a query $q$, we call a variable $w \in \mathtt{var}(R)$ "*solitary*" if it cannot reach another endogenous atom without following one of the edges in $\mathtt{var}(R) - \{w\}$. Then, in each of $s_1, s_2, s_3$, the variable $w$ is solitary, but $w$ is not solitary in $n_4$.

**Definition 5.3** (Full domination)**.** *Let $F$ be an atom of query $q$. $F$ is* fully dominated *iff for all non-solitary variables $y \in \mathtt{var}(F)$ there is another atom $A$ such that $y \in \mathtt{var}(A) \subset \mathtt{var}(F)$.*

Observe that relation $R$ is fully dominated in $q_{\text{brats}}$, as well as in $s_1, s_2, s_3$, but not in $n_4$. On the other hand, $R$ is not fully dominated in $q_{\text{rats}}$ because $y$ is connected to $S(y,z)$ and thus not solitary and not covered by any smaller atom.

We now show that fully dominated atoms may be made exogenous.

**Lemma 5.4** (Full domination)**.** *Let $F$ be a fully dominated atom in an sj-free CQ $q$. Let $q'$ be the modified query in which $F$ is made exogenous. Then $\mathtt{RSP}(q) \equiv \mathtt{RSP}(q')$.*

*Proof.* We have to show that $\mathtt{RSP}(q) \leq \mathtt{RSP}(q')$ and $\mathtt{RSP}(q') \leq \mathtt{RSP}(q)$. Suppose we are given $(D, S(\mathbf{t}))$ and we are interested in the responsibility of tuple $S(\mathbf{t})$. There are two cases. In each case, we will show how, given one of $k, k'$, to produce the other, such that:

$$(D, \mathbf{t}, k) \in \mathtt{RSP}(q) \quad \Leftrightarrow \quad (D', \mathbf{t}, k') \in \mathtt{RSP}(q') \tag{5.5}$$

*Case 1*: $F \neq S$: We show that as in the proof of Prop. 5.2, there is no need to include any tuples from $F$ in a minimum contingency set $\Gamma$ for $q$ in $D$. As in that

proof, we let $\mathbf{j}$ be a witness for $(D - \Gamma) \models q$ and suppose that $F(\mathbf{f}) \in \Gamma$. Thus, $\mathbf{j}$ and $\mathbf{f}$ must disagree on the assignment of at least one variable.

*(a)*: Suppose they differ on some non-solitary variable $y$ of $F$. Let $A$ be the atom that covers $y$ and we can replace $F(\mathbf{f})$ by the tuple $\pi_{\mathtt{var}(A)}(\mathbf{f})$ of $A$. Thus, the sizes of the minimum contingency sets on the two sides are identical and letting $k = k'$ and $D = D'$, Eq. 5.5 holds.

*(b)*: Suppose on the contrary that $\mathbf{j}$ and $\mathbf{f}$ agree on all the non-solitary variables of $F$. Note that since $S$ is endogenous, no non-solitary variable of $F$ can occur in $S$[2]. Thus, the only place that $\mathbf{j}$ and $\mathbf{f}$ disagree is on non-solitary variables of $F$ which do not occur in $S$. Let $F(\mathbf{f_0})$ be the tuple of $F$ that agrees with $\mathbf{j}$. Then $\mathbf{f}$ and $\mathbf{f_0}$ agree on all variables except for solitary variables of $F$. Thus, since removing $S(\mathbf{t})$ from $D - (\Gamma - \{F(\mathbf{f})\})$ removes all witnesses of $D \models q$ that extend $\mathbf{f_0}$, it must also remove all witnesses that extend $\mathbf{f}$, i.e., $\mathbf{f}$ is not useful so it does not occur in $\Gamma$.

*Case 2*: $F = S$: In this case, some tuples of $F$ may need to be in $\Gamma$. Let $I$ be the solitary variables of $F$ and let $W = \left\{\mathbf{f} \in F \;\middle|\; \mathbf{f} \text{ useful} ; \mathbf{f} \neq \mathbf{t} \wedge \pi_{\bar{I}}(\mathbf{f}) = \pi_{\bar{I}}(\mathbf{t})\right\}$. These are the tuples of $F$ which agree with $\mathbf{t}$ on all but the solitary variables of $F$. $W$ must be contained in every contingency set for $(D, \mathbf{t})$. Thus, we let $k = k' + |W|$ and $F' = F - W$. Eq. 5.5 holds. (The point of $\mathbf{f}$ being useful in the definition of $W$ is that solitary variables may occur in some exogenous relations which could already exclude certain values, and thus tuples with those values are not useful so they do not need to be in the contingency set.) □

[2]We are allowing the computation of the responsibility of tuples from exogenous relations just to make the proofs simpler. Notice that we never change the relation $S$ whose tuples we are computing the responsibility of. Thus, if we must make $S$ exogenous, we do so as the last fully-dominated atom we make exogenous.

## 5.1 Triads and hardness

Now that we have established that fully dominated atoms can be made exogenous without changing the complexity of the responsibility problem of a query, we proceed to prove a complexity dichotomy for responsibility.

When studying responsibility, we will insist from now on that every fully dominated atom is exogenous, and analogously to the resilience case, this will be consider the normal form of a query. For example, $q_{\text{rats}}$ has no fully dominated atoms, so it is already in its normal form and it has a triad: $\{R, S, T\}$. Note that we cannot have two elements in a triad such that $\texttt{var}(S_1) \subset \texttt{var}(S_2)$ because removing $\texttt{var}(S_2)$ would isolate $S_1$. Thus $\{R, S, T\}$ is the unique triad of $q_{\text{rats}}$. On the other hand, $R$ is fully dominated in $q_{\text{brats}}$, so we transform it to triad-free $q_{\text{br}^x\text{ats}}$:

$$q_{\text{br}^x\text{ats}} :- A(x), R^{\text{x}}(x, y), B(y), S(y, z), T(z, x) \ .$$

We now show that $\texttt{RSP}(q)$ is NP-complete if $q$ has a triad.

**Lemma 5.6** (Triads make $\texttt{RSP}(q)$ hard)**.** *Let $q$ be an sj-free CQ where all fully dominated atoms are exogenous. If $q$ has a triad, then $\texttt{RSP}(q)$ is NP-complete.*

*Proof.* Depending on which of the following cases the query falls into, we build a reduction to $\texttt{RSP}(q)$ from $\texttt{RSP}(q_\triangle), \texttt{RSP}(q_{\text{rats}})$ or $\texttt{RSP}(q_{\text{T}})$. Let $\mathcal{T} = \{S_0, S_1, S_2\}$ be a triad in query $q$.

*Case 1:* There is no endogenous atom $A$ such that $\texttt{var}(A) \subseteq \texttt{var}(S_i) \cap \texttt{var}(S_j)$, for some $i \neq j$. We will show that $\texttt{RSP}(q_\triangle) \leq \texttt{RSP}(q)$.

Given $D, \mathbf{t}, k$ we must produce $D', \mathbf{t'}, k'$ such that

$$(D, \mathbf{t}, k) \in \texttt{RSP}(q_\triangle) \quad \leftrightarrow \quad (D', \mathbf{t'}, k') \in \texttt{RSP}(q) \ . \tag{5.7}$$

Note that we may assume that $\mathbf{t} = R(a_0, b_0)$ for some values $a_0, b_0$, i.e., that $\mathbf{t}$ is a tuple from $R$, because we know that $\texttt{RSP}(q_\triangle)$ is hard no matter which relation we choose the tuple from (Prop. 2.12).

In this case, we construct $D'$ exactly as we did in Lemma 3.10 (Cases 1 or 2), and as we did there, we let $k' = k$. The only difference is that we must define $\mathbf{t'}$ from $\mathbf{t}$. This is easy: recall that $\mathbf{t} = R(a_0, b_0)$. We let $\mathbf{t'} = S_0(\langle a_0 b_0 \rangle, a_0, b_0)$, i.e., the corresponding tuple of $S_0$. Thus, we have exactly simulated $q_\triangle$ in $q$, so Eq. 5.7 holds.

*Case 2:* There is an endogenous atom $A$ and some $i \neq j$, such that $\mathtt{var}(A) \subseteq \mathtt{var}(S_i) \cap \mathtt{var}(S_j)$, but only for a unique pair $i \neq j$. We show that $\mathtt{RSP}(q_{\text{rats}}) \leq \mathtt{RSP}(q)$. Let the pair be $0, 2$, i.e., $\mathtt{var}(A) \subseteq \mathtt{var}(S_0) \cap \mathtt{var}(S_2)$.

Again, we are given $D, \mathbf{t}, k$, where $\mathbf{t} = R(a_0, b_0)$. We produce $D', \mathbf{t'}$, but now such that,

$$(D, \mathbf{t}, k) \in \mathtt{RSP}(q_{\text{rats}}) \quad \Leftrightarrow \quad (D', \mathbf{t'}, k) \in \mathtt{RSP}(q) \ . \tag{5.8}$$

We produce $D'$ and $\mathbf{t'}$ exactly as in Case 1, and we again have that all the witnesses and minimum contingency sets for $q_{\text{rats}}$ wrt $D, \mathbf{t}$ are preserved for $q$ wrt $D', \mathbf{t'}$. Thus Eq. 5.8 holds.

Finally, we are left with,

*Case 3:* There are endogenous atoms $A, B$ such that WLOG $\mathtt{var}(A) \subseteq \mathtt{var}(S_0) \cap \mathtt{var}(S_2)$, and $\mathtt{var}(B) \subseteq \mathtt{var}(S_0) \cap \mathtt{var}(S_1)$.

We know that $S_0$ is not fully dominated. Thus, there must exist a non-solitary variable $w \in \mathtt{var}(S_0)$ such that $w \notin \mathtt{var}(A) \cup \mathtt{var}(B)$. Since $w$ is not fully dominated, there must be an endogenous atom $C \neq S_0$ such that $C$ is reachable from $S_0$ without using edges from $\mathtt{var}(A) \cup \mathtt{var}(B)$. Thus we have located a tripod sitting in the hypergraph of $q$ (see Fig. 5.3). It thus follow from Prop. 3.5, that $\mathtt{RSP}(q)$ is NP-complete as well. □

## 5.2 The polynomial case

As we saw in the previous section, the presence of triads in a query makes the responsibility problem NP-complete. In the responsibility setting, we require full

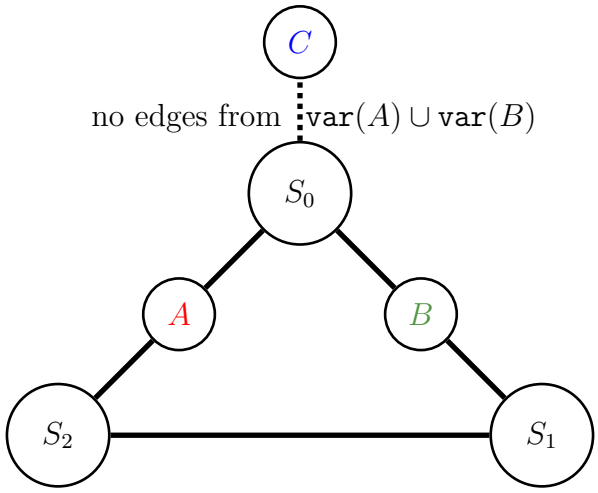



Figure 5.3: Case 3 of the proof of Lemma 5.6. There is a tripod sitting in the hypergraph of $q$.

domination to make an atom exogenous. This means that more atoms may remain endogenous, so there can be more triads. The query $q_{\text{rats}}$ is an example: for resilience we use domination and after applying domination, $q_{\text{rats}}$ has no triads and thus $\texttt{RES}(q_{\text{rats}}) \in \mathsf{PTIME}$. However, if we may only apply full domination, then $q_{\text{rats}}$ keeps the triad $R, S, T$ and thus $\texttt{RSP}(q_{\text{rats}})$ is $\mathsf{NP}$-complete.

We now want to prove the polynomial case for responsibility. Recall that in the proof of Lemma 3.23, we showed the following:

**Corollary 5.9.** *Let $q$ be a CQ that has no triad. Then we can transform $q$, via a series of dissociations, to a linear query $q'$.*

Then, since dissociations cannot make the resilience problem of a sj-free CQ easier (Lemma 3.21), it followed that $\texttt{RES}(q) \in \mathsf{PTIME}$ for any such triad-free query, $q$.

To prove that for any triad-free, sj-free CQ $q$, $\texttt{RSP}(q) \in \mathsf{PTIME}$, it suffices to prove that dissociations cannot make the responsibility problem of such queries easier [22].

### 5.2.1 A generalization of responsibility

We want to prove that if $q'$ is obtained from $q$ through dissociation, then $\texttt{RSP}(q) \leq \texttt{RSP}(q')$. In the proof of the similar result for resilience we did the following. We let $R^{\mathrm{x}}(\mathbf{z})$ be the atom that was changed to $R^{\mathrm{x}\prime}(\mathbf{z}, v)$. We then reduced $\texttt{RES}(q)$ to $\texttt{RES}(q')$ by mapping $(D, k)$ to $(D', k)$ where $D'$ is the same as $D$ with the exception that we let $R' = \big\{(\mathbf{t}, d) \;\big|\; R(\mathbf{t}) \in D; d \in \texttt{dom}(D)\big\}$. This transformation does not change the witness set nor the contingency sets, because, by the way we formed $R'$ from $R$, the conjunct $R'(\mathbf{z}, v)$ places the same restriction on $D'$ that $R(\mathbf{z})$ places on $D$.

This proof goes through fine for responsibility except in one case, namely if the tuple $\mathbf{t}$ that we are computing the responsibility of belongs to $R$, the exogenous relation to which we have added the new variable $v$.[3]

When $\mathbf{t} \in R$, we would like to transform it to $\mathbf{t'} \in R'$ by appending a value, $a_i$, corresponding to the new variable, $v$. However, this will change responsibility in an unclear way. In particular, the responsibility of $\mathbf{t}$ does not correspond to the responsibility of $(\mathbf{t}, a)$ for any particular $a$. It rather corresponds to the responsibility of $(\mathbf{t}, a)$ for *all possible* $a$'s.

To solve our problem, we need to generalize the notion of responsibility to include wildcards.

**Definition 5.10** (tuples with wildcards)**.** *Let $D$ be a database containing a relation, $R(x_1, \ldots, x_c)$. Let $\tau = (s_1, \ldots, s_c)$ be a tuple such that each $s_i \in \texttt{dom}(D) \cup \{*\}$, i.e., $\tau$ may have elements in the domain in some attributes and the wildcard $*$ in others. We call $\tau$ a "*tuple with wildcards*". We say that a tuple $(a_1, \ldots, a_c) \in R$ "*matches*" $\tau$ iff for all $i$, $a_i = s_i$ or $s_i = *$. When $D$ and $R$ are understood, $\tau$ represents a set of tuples from $R$, $\langle\tau\rangle = \big\{\mathbf{a} \in R \;\big|\; \mathbf{a}$ matches $\tau\big\}$.*

---

[3]The reader may wonder why we might need to compute the responsibility of an exogenous tuple. The answer is that the tuple originally might have come from an endogenous relation which we transformed to an exogenous one using full domination.

For example, the tuple with wildcard $(a, *)$ matches all pairs from $R$ whose first coordinate is $a$. We generalize responsibility to allow us to compute the responsibility of a set of tuples denoted by a tuple with wildcards:

**Definition 5.11** ($\texttt{RSP}^*$)**.** *Let $D$ be a database containing a relation $R$, $q$ a query for $D$, and $\tau$ a tuple with wildcards. Then $(D, \tau, k) \in \texttt{RSP}^*(q)$ iff there exists a contingency set $\Gamma$ of size $k$ such that $(D - \Gamma) \models q$ and $(D - (\Gamma \cup \langle\tau\rangle)) \not\models q$.*

Since $\texttt{RSP}^*(q)$ is just a generalization of $\texttt{RSP}(q)$, it is immediate that $\texttt{RSP}(q) \leq \texttt{RSP}^*(q)$. Thus, $\texttt{RSP}^*(q)$ is NP-complete whenever $\texttt{RSP}(q)$ is:

**Corollary 5.12** ($\texttt{RSP}^*$ hardness)**.** *Let $q$ be an sj-free CQ all of whose fully dominated atoms are exogenous. If $q$ has a triad then $\texttt{RSP}^*(q)$ is NP-complete.*

From our previous discussion, it now follows that dissociation does not make $\texttt{RSP}^*(q)$ easier:

**Lemma 5.13** (Dissociation and $\texttt{RSP}^*$)**.** *If $q'$ is obtained from $q$ through dissociation, then $\texttt{RSP}^*(q) \leq \texttt{RSP}^*(q')$.*

Furthermore, linear queries are still easy for responsibility:

**Lemma 5.14** (Linear queries and $\texttt{RSP}^*$)**.** *For any linear sj-free CQ $q$, $\texttt{RSP}^*(q)$ is in* PTIME*.*

**Corollary 5.15.** *If $q$ has no triad, then $\texttt{RSP}^*(q)$ can be made linear by using dissociations, and is thus in* PTIME*. Therefore so is $\texttt{RSP}(q)$.*

We have thus proved our desired dichotomy for responsibility, and as a bonus, we have proved it for groups of tuples with wildcards as well:

**Theorem 5.16** (Responsibility Dichotomy)**.** *Let $q$ be an sj-free CQ, and let $q'$ be the result of making all fully dominated atoms exogenous. If $q'$ contains a triad then $\texttt{RSP}(q)$ and $\texttt{RSP}^*(q)$ are NP-complete. Otherwise, $\texttt{RSP}(q)$ and $\texttt{RSP}^*(q)$ are* PTIME*.*

It follows from Corollary 5.15 and Corollary 5.12 that $\texttt{RSP}^*(q) \equiv \texttt{RSP}(q)$ for all sj-free CQ, $q$. Note that it is not at all clear how one would build a reduction from $\texttt{RSP}^*(q)$ to $\texttt{RSP}(q)$. However, our characterization of the complexity of $\texttt{RSP}(q)$ and $\texttt{RSP}^*(q)$ gives us this result: After all fully dominated atoms are made exogenous, if there is a triad, then $\texttt{RSP}(q)$ is NP-complete, thus so is $\texttt{RSP}^*(q)$. If there is no triad, then $\texttt{RSP}^*(q) \in \mathsf{PTIME}$, thus so is $\texttt{RSP}(q)$:

**Corollary 5.17.** *For all sj-free CQ $q$, $\texttt{RSP}(q) \equiv \texttt{RSP}^*(q)$.*

## 5.3 Dichotomy for responsibility with FDs

Our final theorem is that the dichotomy for responsibility continues to hold in the presence of FDs:

**Theorem 5.18** (FD Responsibility Dichotomy)**.** *Let $(q; \Phi)$ be an sj-free CQ with functional dependencies. Let $(q^*, \Phi)$ be its closure under induced rewrites, and such that all fully dominated atoms of $q^*$ are exogenous. If $q^*$ has a triad then $\texttt{RSP}(q; \Phi)$ is NP-complete. Otherwise, $\texttt{RSP}(q; \Phi) \in \mathsf{PTIME}$.*

*Proof.* Since FDs only make $\texttt{RSP}(q)$ easier, we know that if $q^*$ has no triad then $\texttt{RSP}(q^*)$ is easy, thus so is $\texttt{RSP}(q^*; \Phi)$ and thus also $\texttt{RSP}(q; \Phi)$. For the converse, we show that the reduction, $f$, from one of $\texttt{RSP}(q_\triangle), \texttt{RSP}(q_{\text{rats}}), \texttt{RSP}(q_{\text{T}})$ to $\texttt{RSP}(q)$ which we built in Lemma 5.6 always produces databases, $D'$, that satisfy $\Phi$. The proof is almost exactly as in Lemma 3.33. Note that in the proof of Lemma 5.6, we use the same reduction in all three cases, i.e., no matter if we are reducing from $\texttt{RSP}(q_\triangle), \texttt{RSP}(q_{\text{rats}})$, or $\texttt{RSP}(q_{\text{T}})$. □

# CHAPTER 6

# CONCLUSIONS AND FUTURE WORK

In this thesis we focus on analyzing the complexity of the Resilience problem for conjunctive queries and we conjecture there is a NP vs P dichotomy for this problem. We split the class of conjunctive queries into self-join free (sj-free) and self-join queries.

We first analyzed the sj-free case and we proved there is a dichotomy. In more details we show:

- We define the concept of *triad*, which we use to show which sj-free queries are NP-complete;
- We completely characterize the complexity of sj-free queries, proving there is a dichotomy;
- The triad criterion is an easy to check property - polynomial on the size of the query, and by consequence, it is easy to check the complexity of a given query;
- We extend the results for sj-free queries to include the presence of functional dependencies;
- We show a dichotomy result for the related problem of Responsibility.

In a second phase, we analyzed the self-join case and we quickly understood self-join queries have a far more complex structure than the sj-free ones. We decided to restrict ourselves to binary queries without variable repetition and, even though we haven't proved a complete dichotomy yet, we have strong indications our conjecture holds true. Among our findings, we would like to highlight:

- We show that we need to consider minimal queries in order to analyze this problem correctly. This fact already contrasts with the sj-free case. Since it is

NP-complete to find the minimal equivalent of a given conjunctive query, it will be NP-complete to determine what the complexity of a given query is;

- We show that triads are still meaningful in this setting, however not a sufficient criterion for proving NP-completeness;
- We define 3 query structures, called chain, permutation and confluence, which help us determine the complexity of a query in various ways;
- We state a dichotomy conjecture for binary queries with self-joins.

## 6.1 Future work

We ultimately want to prove our dichotomy conjecture for self-join binary queries, and possibly extend that result to all of self-join conjunctive queries. The following are problems that relate to our investigations and which seem like natural next steps in this line of research.

### 6.1.1 Approximations and generalizations

We have shown a dichotomy for the resilience problem, which allows us to say we cannot solve resilience for a certain set of queries, unless PTIME =NP. But there is still hope! We can try to get good approximations for those cases. As resilience is related to vertex cover, we will start by applying the same ideas for approximations for vertex cover in here and see what kind of approximation we get. On the other hand, we know that some queries are hard to approximate for deletion propagation with view side-effects [32, 33], so we can expect a similar structure in this case too.

We can also think about a straightforward generalization of the deletion propagation problem: *"What if we are interested in deleting more than one tuple from the view?"* Intuitively, one could say that the problems are equivalent, since you could consider one tuple at a time, but that is not the case. The multi-tuple deletion propagation problem has been studied for the view side-effect variant [34] and, in this

work, it was shown to have a different characterization from the one found for the single-tuple case.

The multi-tuple version of deletion propagation is a more realistic version of the problems and we could analyze what changes arise for the source side-effects variant. Another possible generalization is to consider changing the result of many views at a time.

### 6.1.2 Deletion propagation with view side-effects

As we discussed on the related work (Section 2.3.1), the view side-effect variant of deletion propagation has been studied and similar dichotomy results were obtained. There are still many open problems, in particular: (1) How does the dichotomy change, if at all, when we introduce the concept of forbidden tuples in the definition of the problem? and (2) Is there a dichotomy result for the class of conjunctive queries with self-joins? The results obtained in [32, 33] consider all tuples endogenous and do not tackle the problem for self-joins.

For the first question, it is plausible to conjecture that considering some tuples as exogenous can have an impact on the dichotomy, since that is true for the source side-effect variant. For the second question, we can think of how our ideas for solving the self-join case could carry on to the view side-effect variant.

### 6.1.3 Refine our dichotomy

Dichotomies are an interesting phenomenon in the study of complexity classes. One of the most known results in the area is Schaefer's dichotomy theorem showing that, given certain restrictions on the relations that can be used, a Boolean constraint satisfaction problem (CSP) is either in PTIME or is NP-complete [38]. A refinement of this result [3] shows every Boolean CSP problem is either trivial or complete for one of the following classes: NP, P, $\oplus$L, NL, L.

Given that the resilience problem has a dichotomy, we can try to express it as a CSP and from there identify which queries will fall into the lower complexity classes, in particular the trivial case. It is often the case that polynomial-time is not low enough to imply efficiency when considering large quantities of data, therefore finding the cases with very low complexity would have a practical impact.

### 6.1.4 Connections with vertex cover in hypergraphs

Resilience and vertex cover in hypergraphs are closely related, as we can informally see in some of the examples and proofs. It would be nice if we could make this relation formal, by defining a reduction or something of the sort. The whole point is to find out if we could use the triad concept to define classes of graphs in which the vertex cover problem is easy. We need to remember that when we are talking about triads, we are talking about the query hypergraph but the vertex cover refers to the graph we obtain when we consider a query and a database together.

# BIBLIOGRAPHY


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

[2] D. Agarwal, D. Barman, D. Gunopulos, N. E. Young, F. Korn, and D. Srivastava. Efficient and effective explanation of change in hierarchical summaries. In *KDD*, pages 6–15, 2007.

[3] E. Allender, M. Bauland, N. Immerman, H. Schnoor, and H. Vollmer. The complexity of satisfiability problems: Refining schaefer's theorem. *J. Comput. Syst. Sci.*, 75(4):245–254, June 2009.

[4] A. Amarilli, M. Monet, and P. Senellart. Conjunctive queries on probabilistic graphs: Combined complexity. In *Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, PODS '17, pages 217–232, New York, NY, USA, 2017. ACM.

[5] F. Bancilhon and N. Spyratos. Update semantics of relational views. *ACM TODS*, 6(4):557–575, Dec. 1981.

[6] D. Barman, F. Korn, D. Srivastava, D. Gunopulos, N. E. Young, and D. Agarwal. Parsimonious explanations of change in hierarchical data. In *ICDE*, pages 1273–1275, 2007.

[7] G. Bender, L. Kot, and J. Gehrke. Explainable security for relational databases. *SIGMOD*, pages 1411–1422, 2014.

[8] P. Buneman, S. Khanna, and W. C. Tan. Why and where: A characterization of data provenance. In *ICDT*, pages 316–330, 2001.

[9] P. Buneman, S. Khanna, and W.-C. Tan. On propagation of deletions and annotations through views. In *PODS*, pages 150–158, 2002.

[10] A. K. Chandra and P. M. Merlin. Optimal implementation of conjunctive queries in relational data bases. In *STOC*, pages 77–90, 1977.

[11] A. Chapman and H. V. Jagadish. Why not? In *SIGMOD*, pages 523–534, 2009.

[12] J. Cheney, L. Chiticariu, and W. C. Tan. Provenance in databases: Why, how, and where. *Foundations and Trends in Databases*, 1(4):379–474, 2009.

[13] H. Chockler and J. Y. Halpern. Responsibility and blame: A structural-model approach. *J. Artif. Intell. Res. (JAIR)*, 22:93–115, 2004.

[14] G. Cong, W. Fan, F. Geerts, J. Li, and J. Luo. On the complexity of view update analysis and its application to annotation propagation. *IEEE TKDE*, 24(3):506–519, 2012.

[15] S. S. Cosmadakis and C. H. Papadimitriou. Updates of relational views. *J. ACM*, 31(4):742–760, Sept. 1984.

[16] Y. Cui, J. Widom, and J. L. Wiener. Tracing the lineage of view data in a warehousing environment. *ACM TODS*, 25(2):179–227, 2000.

[17] U. Dayal and P. A. Bernstein. On the correct translation of update operations on relational views. *ACM TODS*, 7(3):381–416, 1982.

[18] T. Eiter and T. Lukasiewicz. Complexity results for structure-based causality. *Artif. Intell.*, 142(1):53–89, 2002.

[19] T. Eiter and T. Lukasiewicz. Causes and explanations in the structural-model approach: Tractable cases. *Artif. Intell.*, 170(6-7):542–580, 2006.

[20] D. Fabbri and K. LeFevre. Explanation-based auditing. *PVLDB*, 5(1):1–12, 2011.

[21] R. Fagin, J. D. Ullman, and M. Y. Vardi. On the semantics of updates in databases. In *PODS*, pages 352–365, 1983.

[22] C. Freire, W. Gatterbauer, N. Immerman, and A. Meliou. A characterization of the complexity of resilience and responsibility for self-join-free conjunctive queries. *arXiv*, 1507.00674:1–36, 2015.

[23] C. Freire, W. Gatterbauer, N. Immerman, and A. Meliou. The complexity of resilience and responsibility for self-join-free conjunctive queries. *Proc. VLDB Endow.*, 9(3):180–191, Nov. 2015.

[24] T. J. Green, G. Karvounarakis, and V. Tannen. Provenance semirings. In *PODS*, pages 31–40, 2007.

[25] J. Y. Halpern and J. Pearl. Causes and explanations: A structural-model approach. Part I: Causes. *Brit. J. Phil. Sci.*, 56:843–887, 2005.

[26] M. Herschel and M. A. Hernández. Explaining missing answers to SPJUA queries. *PVLDB*, 3(1):185–196, 2010.

[27] M. Herschel, M. A. Hernández, and W. C. Tan. Artemis: A system for analyzing missing answers. *PVLDB*, 2(2):1550–1553, 2009.

[28] J. Huang, T. Chen, A. Doan, and J. F. Naughton. On the provenance of non-answers to queries over extracted data. *PVLDB*, 1(1):736–747, 2008.

[29] N. Immerman. *Descriptive Complexity*. Springer, 1999.

[30] A. M. Keller. Algorithms for translating view updates to database updates for views involving selections, projections, and joins. In *PODS*, pages 154–163, 1985.

[31] N. Khoussainova, M. Balazinska, and D. Suciu. Perfxplain: debugging mapreduce job performance. *PVLDB*, 5(7):598–609, 2012.

[32] B. Kimelfeld. A dichotomy in the complexity of deletion propagation with functional dependencies. In *PODS*, pages 191–202, 2012.

[33] B. Kimelfeld, J. Vondrák, and R. Williams. Maximizing conjunctive views in deletion propagation. *ACM TODS*, 37(4):24:1–24:37, 2012.

[34] B. Kimelfeld, J. Vondrák, and D. P. Woodruff. Multi-tuple deletion propagation: Approximations and complexity. *PVLDB*, 6(13):1558–1569, 2013.

[35] A. Meliou, W. Gatterbauer, K. F. Moore, and D. Suciu. The complexity of causality and responsibility for query answers and non-answers. *PVLDB*, 4(1):34–45, 2010.

[36] A. Meliou, W. Gatterbauer, S. Nath, and D. Suciu. Tracing data errors with view-conditioned causality. In *SIGMOD*, pages 505–516, 2011.

[37] S. Roy and D. Suciu. A formal approach to finding explanations for database queries. In *SIGMOD*, pages 1579–1590, 2014.

[38] T. J. Schaefer. The complexity of satisfiability problems. In *Proceedings of the Tenth Annual ACM Symposium on Theory of Computing*, STOC '78, pages 216–226, New York, NY, USA, 1978. ACM.

[39] S. Thirumuruganathan, M. Das, S. Desai, S. Amer-Yahia, G. Das, and C. Yu. Maprat: meaningful explanation, interactive exploration and geo-visualization of collaborative ratings. *PVLDB*, 5(12):1986–1989, 2012.

[40] Q. T. Tran and C.-Y. Chan. How to conquer why-not questions. In *SIGMOD*, pages 15–26, 2010.

[41] E. Wu and S. Madden. Scorpion: Explaining away outliers in aggregate queries. *PVLDB*, 6(8):553–564, 2013.