Department of Computer Science

A GENERAL FRAMEWORK FOR INCONSISTENCY-TOLERANT QUERY ANSWERING IN DATALOG+/−

Thomas Lukasiewicz
Maria Vanina Martinez
Gerardo I. Simari

RR-14-04

Department of Computer Science, University of Oxford
Wolfson Building, Parks Road, Oxford OX1 3QD

DEPARTMENT OF COMPUTER SCIENCE RESEARCH REPORT

DEPARTMENT OF COMPUTER SCIENCE RESEARCH REPORT 14-04, MAY 2014

A GENERAL FRAMEWORK FOR INCONSISTENCY-TOLERANT QUERY ANSWERING IN DATALOG+/–

(PRELIMINARY VERSION, 21 MAY 2014)

Thomas Lukasiewicz [1] Maria Vanina Martinez [2] Gerardo I. Simari [3]

Abstract. Inconsistency management in knowledge bases is an important problem that has been studied for a long time. During the recent years, additional interest in this topic has been sparked with the advent of the Semantic Web. In this paper, we study different semantics for query answering in inconsistent Datalog+/– ontologies. Datalog+/– is a family of ontology languages that is in particular useful for representing and reasoning over lightweight ontologies in the Semantic Web. We develop a general framework for inconsistency management in Datalog+/– ontologies based on incision functions from belief revision, in which we can characterize several query answering semantics as special cases: (i) consistent answers, originally developed for relational databases and recently adopted for some classes of description logics (DLs); (ii) intersection semantics, a sound approximation of consistent answers; and (iii) lazy answers, a novel semantics proposed as an alternative to approximations to consistent answers that, taking the union of lazy answers, can be used to obtain a good compromise between quality of answers and computation time for some fragments of Datalog+/–. We also provide complexity results for query answering under the different semantics, including data tractability results and first-order rewritability for query answering under the intersection semantics for linear Datalog+/–.

[1]Department of Computer Science, University of Oxford, UK; e-mail: thomas.lukasiewicz@cs.ox.ac.uk.
[2]Department of Computer Science, University of Oxford, UK; e-mail: vanina.martinez@cs.ox.ac.uk.
[3]Department of Computer Science, University of Oxford, UK; e-mail: gerardo.simari@cs.ox.ac.uk.

Acknowledgements: This paper is a significantly extended and revised version of the results first published in [1] and [2]. This work was supported by the Engineering and Physical Sciences Research Council (EPSRC) grant EP/J008346/1 ("PrOQAW: Probabilistic Ontological Query Answering on the Web"), the European Research Council under the EU's 7th Framework Programme (FP7/2007-2013/ERC) grant 246858 ("DIADEM"), a Google Research Award, and a Yahoo! Research Fellowship.

Contents

1 Introduction

It has been widely acknowledged in both the Semantic Web and databases communities that inconsistency is an issue that cannot be ignored. Knowledge bases in the Semantic Web and databases in the form of ontologies are becoming increasingly popular, and when integrating data from many different sources, either as a means to populate an ontology or simply to answer queries, integrity constraints are very likely to be violated in practice. In this paper, we address the problem of handling inconsistency in ontologies for the Semantic Web, where scalability is an important issue.

We adopt the recently developed Datalog+/– family of ontology languages [3]. In particular, we focus on the *guarded* and *linear* fragments of Datalog+/–, which guarantee termination of query answering procedures in polynomial time in the data complexity and first-order rewritability, respectively. Datalog+/– enables a modular rule-based style of knowledge representation, and it can represent syntactical fragments of first-order logic so that answering a BCQ Q under a set Σ of Datalog+/– rules for an input database D is equivalent to the classical entailment check $D \cup \Sigma \models Q$. Furthermore, its properties of decidability of query answering and good query answering complexity in the data complexity allows to realistically assume that the database D is the only really large object in the input. These properties, together with its expressive power, make Datalog+/– very useful in modeling real applications such as ontology querying, Web data extraction, data exchange, ontology-based data access, and data integration. The results in this paper extend to the (entire) *DL-Lite* family of description logics (DLs), since linear Datalog+/– is strictly more expressive than the whole *DL-Lite* family; furthermore, the results for guarded Datalog+/– extend to $\mathcal{EL}$, since guarded Datalog+/– is strictly more expressive than this language.

The following example shows a simple Datalog+/– ontology; the language and standard semantics for query answering in Datalog+/– ontologies is recalled in the next section.

Example 1 A (guarded) Datalog+/– ontology $KB = (D, \Sigma_T \cup \Sigma_E \cup \Sigma_{NC})$ is given below. Here, the formulas in Σ_T, Σ_{NC}, and Σ_E correspond to tuple-generating dependencies (TGDs), negative constraints, and equality generating dependencies (EGDs), respectively. We recall the formal definitions for such constraints in the following section.

$$\begin{aligned}
D \;&= \{\mathit{directs}(john, d_1),\ \mathit{directs}(tom, d_1),\ \mathit{directs}(tom, d_2),\ \mathit{manager}(tom, d_1),\\
&\quad\ \mathit{supervises}(tom, john),\ \mathit{works\_in}(john, d_1),\ \mathit{works\_in}(tom, d_1)\};\\
\Sigma_T \;&= \{\sigma_1 : \mathit{works\_in}(X, D) \rightarrow \mathit{emp}(X),\ \sigma_2 : \mathit{directs}(X, D) \rightarrow \mathit{emp}(X),\\
&\quad\ \sigma_3 : \mathit{directs}(X, D)\ \wedge\ \mathit{works\_in}(X, D) \rightarrow \mathit{manager}(X, D)\};\\
\Sigma_{NC} &= \{\upsilon_1 : \mathit{supervises}(X, Y) \wedge \mathit{manager}(Y, D) \rightarrow \bot,\\
&\quad\ \upsilon_2 : \mathit{supervises}(X, Y) \wedge \mathit{works\_in}(X, D) \wedge \mathit{directs}(Y, D) \rightarrow \bot\};\\
\Sigma_E \;&= \{\upsilon_3 : \mathit{directs}(X, D) \wedge \mathit{directs}(X, D') \rightarrow D = D'\}.
\end{aligned}$$

Here, the formulas in Σ_T say that every person working for a department is an employee (σ_1), that every person directing a department is an employee (σ_2), and that each person that directs a department and works in that department is the manager of that department (σ_3). The formula υ_1 in Σ_{NC} states that if X supervises Y, then Y cannot be a manager, while υ_2 says that if Y is supervised by someone in a department, then Y cannot direct that department. Finally, formula υ_3 in Σ_E states that the same person cannot direct two different departments. As we show later, this ontology is inconsistent. For instance, the atoms *directs*$(john, d_1)$ and *works_in*$(john, d_1)$ trigger the application of σ_3, producing *manager*$(john, d_1)$, but that together with *supervises*$(tom, john)$ (which belongs to D) violates υ_1. ■

Two research areas are especially relevant to our work. The first is *belief revision*, an area of study in artificial intelligence (AI) and philosophy, which deals with the general problem of extending, contracting, or revising a knowledge base composed of logical formulas. The second is from databases, and focuses on finding consistent answers to possibly inconsistent databases; this can be done *on the fly* during the query answering process, or over a database that has been previously treated to excise the pieces of information causing the inconsistencies. We discuss related work in more detail in Section 6.

In this paper, we develop the first inconsistency-tolerant semantics for query answering in Datalog+/– ontologies, which is based on a general framework for handling inconsistency via the application of functions inspired by the concept of incision functions from belief revision. This framework allows us to study inconsistency-tolerant semantics from a more abstract point of view, deviating the focus of attention from repairs and consistent answers.

Furthermore, we provide a *query rewriting* approach to inconsistency-tolerant query answering under a particular semantics for some fragments of Datalog+/–, i.e., a given query is rewritten into another query, which fully embeds any underlying ontological knowledge and that, evaluated on the data, returns the consistent answers under the intersection semantics. The result of this rewriting process is a first-order (FO) query. FO-rewritability of queries is an important property, since the rewritten query can immediately be translated into standard SQL. In this way, we reduce the problem of query answering over an ontology to the standard evaluation of an SQL query in (possibly highly optimized) relational database management systems. Several works have already provided algorithms for FO rewriting for fragments of Datalog+/–; however, all of them focus on the standard semantics for query answering, while here we extend the rewriting process of those sublanguages to specific inconsistency tolerant semantics to answer conjunctive queries.

The main contributions are briefly as follows:

- We develop a general framework for handling inconsistency in Datalog+/–, which is based on incision functions, and captures several different inconsistency-tolerant semantics previously developed in the literature.

- Within this framework, we propose a new inconsistency-tolerant semantics (lazy answers) that, taking the union of lazy answers, can be used to obtain a good compromise between answer quality and tractability for fragments of Datalog+/–, shifting the focus away from traditional repairs and consistent answers.

- We provide complexity results for the problem of query answering in linear and guarded Datalog+/– ontologies under the different semantics that we define within the framework, including tractability results for the lazy semantics for atomic BCQs in linear Datalog+/–, and FO-rewritability of BCQs under the intersection semantics for linear Datalog+/–.

- Finally, we show that FO query rewriting is feasible under the intersection semantics for FO-rewritable (under standard query answering semantics) fragments of Datalog+/–.

The paper is organized as follows: Section 2 recalls the basics on Datalog+/– from [3]. In Section 3, we develop a general framework for inconsistency management in Datalog+/– ontologies based on the notion of incision functions from the belief revision literature and characterize consistent answers and intersection semantics as special cases of incision functions. Section 4 presents the *lazy* semantics for consistent query answering in Datalog+/– ontologies and studies its properties, complexity, and comparison with the other two semantics. In Section 5 we develop an efficient first-order rewritability approach for query answering under the intersection semantics for the linear fragment of Datalog+/–. Finally, Sections 6 and 7 discuss related work and conclusions, respectively.

2 Preliminaries

We briefly recall some basics on Datalog+/– [3], namely, on relational databases, (Boolean) conjunctive queries ((B)CQs), tuple- and equality-generating dependencies (TGDs and EGDs, respectively), negative constraints, the chase, and ontologies in Datalog+/–.

Databases and Queries. We assume (i) an infinite universe of *(data) constants* Δ (which constitute the "normal" domain of a database), (ii) an infinite set of *(labeled) nulls* Δ_N (used as "fresh" Skolem terms, which are placeholders for unknown values, and can thus be seen as variables), and (iii) an infinite set of variables $\mathcal{V}$ (used in queries, dependencies, and constraints). Different constants represent different values (*unique name assumption*), while different nulls may represent the same value. We assume a lexicographic order on $\Delta \cup \Delta_N$, with every symbol in Δ_N following all symbols in Δ. We denote by $\mathbf{X}$ sequences of variables $X_1, \ldots, X_k$ with $k \geqslant 0$.

We assume a *relational schema* $\mathcal{R}$, which is a finite set of *predicate symbols* (or simply *predicates*). A *term* t is a constant, null, or variable. An *atomic formula* (or *atom*) $\mathbf{a}$ has the form $P(t_1, ..., t_n)$, where P is an n-ary predicate, and $t_1, ..., t_n$ are terms. A conjunction of atoms is often identified with the set of all its atoms.

A *database (instance)* D for a relational schema $\mathcal{R}$ is a (possibly infinite) set of atoms with predicates from $\mathcal{R}$ and arguments from Δ. A *conjunctive query (CQ)* over $\mathcal{R}$ has the form $Q(\mathbf{X}) = \exists \mathbf{Y}\, \Phi(\mathbf{X}, \mathbf{Y})$, where $\Phi(\mathbf{X}, \mathbf{Y})$ is a conjunction of atoms (possibly equalities, but not inequalities) with the variables $\mathbf{X}$ and $\mathbf{Y}$, and possibly constants, but without nulls. A *Boolean CQ (BCQ)* over $\mathcal{R}$ is a CQ of the form $Q()$, often written as the set of all its atoms, without quantifiers. Answers to CQs and BCQs are defined via *homomorphisms*, which are mappings $\mu\colon \Delta \cup \Delta_N \cup \mathcal{V} \to \Delta \cup \Delta_N \cup \mathcal{V}$ such that (i) $c \in \Delta$ implies $\mu(c) = c$, (ii) $c \in \Delta_N$ implies $\mu(c) \in \Delta \cup \Delta_N$, and (iii) μ is naturally extended to atoms, sets of atoms, and conjunctions of atoms. The set of all *answers* to a CQ $Q(\mathbf{X}) = \exists \mathbf{Y}\, \Phi(\mathbf{X}, \mathbf{Y})$ over a database D, denoted $Q(D)$, is the set of all tuples $\mathbf{t}$ over Δ for which there exists a homomorphism $\mu\colon \mathbf{X} \cup \mathbf{Y} \to \Delta \cup \Delta_N$ such that $\mu(\Phi(\mathbf{X}, \mathbf{Y})) \subseteq D$ and $\mu(\mathbf{X}) = \mathbf{t}$. The *answer* to a BCQ $Q()$ over a database D is *Yes*, denoted $D \models Q$, iff $Q(D) \neq \emptyset$. For the sake of simplicity in the presentation, we sometimes denote the set of answers to a CQ $Q(\mathbf{X}) = \exists \mathbf{Y}\, \Phi(\mathbf{X}, \mathbf{Y})$ with the set of atoms in the conjunction $\mu(\Phi(\mathbf{X}, \mathbf{Y}))$.

Given a relational schema $\mathcal{R}$, a *tuple-generating dependency (TGD)* σ is a first-order formula of the form $\forall \mathbf{X} \forall \mathbf{Y}\, \Phi(\mathbf{X}, \mathbf{Y}) \to \exists \mathbf{Z}\, \Psi(\mathbf{X}, \mathbf{Z})$, where $\Phi(\mathbf{X}, \mathbf{Y})$ and $\Psi(\mathbf{X}, \mathbf{Z})$ are conjunctions of atoms over $\mathcal{R}$ (without nulls), called the *body* and the *head* of σ, denoted $body(\sigma)$ and $head(\sigma)$, respectively. Such σ is satisfied in a database D for $\mathcal{R}$ iff, whenever there exists a homomorphism h that maps the atoms of $\Phi(\mathbf{X}, \mathbf{Y})$ to atoms of D, there exists an extension h' of h that maps the atoms of $\Psi(\mathbf{X}, \mathbf{Z})$ to atoms of D. All sets of TGDs are finite here. Since TGDs can be reduced to TGDs with only single atoms in their heads, in the sequel, every TGD has w.l.o.g. a single atom in its head. A TGD σ is *guarded* iff it contains an atom in its body that contains all universally quantified variables of σ. The leftmost such atom is the *guard atom* (or *guard*) of σ. A TGD σ is *linear* iff it contains only a single atom in its body.

Query answering under TGDs, i.e., the evaluation of CQs and BCQs on databases under a set of TGDs is defined as follows. For a database D for $\mathcal{R}$, and a set of TGDs Σ on $\mathcal{R}$, the set of *models* of D and Σ, denoted $mods(D, \Sigma)$, is the set of all (possibly infinite) databases B such that (i) $D \subseteq B$ and (ii) every $\sigma \in \Sigma$ is satisfied in B. The set of *answers* for a CQ Q to D and Σ, denoted $ans(Q, D, \Sigma)$, is the set of all tuples $\mathbf{a}$ such that $\mathbf{a} \in Q(B)$ for all $B \in mods(D, \Sigma)$. The *answer* for a BCQ Q to D and Σ is *Yes*, denoted $D \cup \Sigma \models Q$, iff $ans(Q, D, \Sigma) \neq \emptyset$. Note that query answering under general TGDs is undecidable [4], even when the schema and TGDs are fixed [5]. The two problems of CQ and BCQ evaluation under TGDs are LOGSPACE-equivalent [6, 7]. Moreover, the query output tuple (QOT) problem (as a decision version of CQ evaluation) and BCQ evaluation are AC_0-reducible to each other. Henceforth, we thus focus only on BCQ evaluation, and any complexity results carry over to the other problems. Decidability of query answering for the guarded case follows from a bounded tree-width property. The data complexity of query answering in this case is P-complete.

Negative constraints (or simply *constraints*) γ are first-order formulas $\forall \mathbf{X} \Phi(\mathbf{X}) \rightarrow \perp$, where $\Phi(\mathbf{X})$ (called the *body* of γ) is a conjunction of atoms (without nulls and not necessarily guarded). Under the standard semantics of query answering of BCQs in Datalog+/– with TGDs, adding negative constraints is computationally easy, as for each constraint $\forall \mathbf{X} \Phi(\mathbf{X}) \rightarrow \perp$, we only have to check that the BCQ $\Phi(\mathbf{X})$ evaluates to false in D under Σ; if one of these checks fails, then the answer to the original BCQ Q is true, otherwise the constraints can simply be ignored when answering the BCQ Q.

Equality-generating dependencies (*EGDs*) σ, are first-order formulas $\forall \mathbf{X}\, \Phi(\mathbf{X}) \rightarrow X_i = X_j$, where $\Phi(\mathbf{X})$, called the *body* of σ, denoted *body*(σ), is a (without nulls and not necessarily guarded) conjunction of atoms, and X_i and X_j are variables from $\mathbf{X}$. Such σ is satisfied in a database D for $\mathcal{R}$ iff, whenever there exists a homomorphism h such that $h(\Phi(\mathbf{X}, \mathbf{Y})) \subseteq D$, it holds that $h(X_i) = h(X_j)$. Adding EGDs over databases with guarded TGDs along with negative constraints does not increase the complexity of BCQ query answering as long as they are *non-conflicting* [3]. Intuitively, this ensures that, if the chase (see below) fails (due to strong violations of EGDs), then it already fails on the database D, and if it does not fail, then whenever "new" atoms (from the logical point of view) are created in the chase by the application of the EGD chase rule, atoms that are logically equivalent to the new ones are guaranteed to be generated also in the absence of the EGDs. This guarantees that EGDs do not have any impact on the chase with respect to query answering. Non-conflicting EGDs can be expressed as negative constraints of the form $\forall \mathbf{X}\, \Phi(\mathbf{X}), X_i \neq X_j \rightarrow \perp$. In the following, for ease of presentation, all non-conflicting EGDs are expressed as such special forms of negative constraints.

We usually omit the universal quantifiers in TGDs, negative constraints, and EGDs, and we implicitly assume that all sets of dependencies and/or constraints are finite.

The Chase. The *chase* was first introduced to enable checking implication of dependencies, and later also for checking query containment. By "chase", we refer both to the chase procedure and to its output. The TGD chase works on a database via so-called TGD *chase rules* (see [3] for an extended chase with also EGD chase rules).

TGD Chase Rule. Let D be a database, and σ a TGD of the form $\Phi(\mathbf{X}, \mathbf{Y}) \rightarrow \exists \mathbf{Z}\, \Psi(\mathbf{X}, \mathbf{Z})$. Then, σ is *applicable* to D if there exists a homomorphism h that maps the atoms of $\Phi(\mathbf{X}, \mathbf{Y})$ to atoms of D. Let σ be applicable to D, and h_1 be a homomorphism that extends h as follows: for each $X_i \in \mathbf{X}$, $h_1(X_i) = h(X_i)$; for each $Z_j \in \mathbf{Z}$, $h_1(Z_j) = z_j$, where z_j is a "fresh" null, i.e., $z_j \in \Delta_N$, z_j does not occur in D, and z_j lexicographically follows all other nulls already introduced. The *application of* σ on D adds to D the atom $h_1(\Psi(\mathbf{X}, \mathbf{Z}))$ if not already in D.

The chase algorithm for a database D and a set of TGDs Σ consists of an exhaustive application of the TGD chase rule in a breadth-first (level-saturating) fashion, which outputs a (possibly infinite) chase for D and Σ. Formally, the *chase of level up to* 0 of D relative to Σ, denoted *chase*$^0(D, \Sigma)$, is defined as D, assigning to every atom in D the *(derivation) level* 0. For every $k \geqslant 1$, the *chase of level up to* k of D relative to Σ, denoted *chase*$^k(D, \Sigma)$, is constructed as follows: let $I_1, \ldots, I_n$ be all possible images of bodies of TGDs in Σ relative to some homomorphism such that (i) $I_1, \ldots, I_n \subseteq$ *chase*$^{k-1}(D, \Sigma)$ and (ii) the highest level of an atom in every I_i is $k-1$; then, perform every corresponding TGD application on *chase*$^{k-1}(D, \Sigma)$, choosing the applied TGDs and homomorphisms in a (fixed) linear and lexicographic order, respectively, and assigning to every new atom the *(derivation) level* k. The *chase* of D relative to Σ, denoted $chase(D, \Sigma)$, is defined as the limit of *chase*$^k(D, \Sigma)$ for $k \rightarrow \infty$.

The (possibly infinite) chase relative to TGDs is a *universal model*, i.e., there exists a homomorphism from $chase(D, \Sigma)$ onto every $B \in mods(D, \Sigma)$ [3]. This implies that BCQs Q over D and Σ can be evaluated on the chase for D and Σ, i.e., $D \cup \Sigma \models Q$ is equivalent to $chase(D, \Sigma) \models Q$. For guarded TGDs Σ, such BCQs Q can be evaluated on an initial fragment of $chase(D, \Sigma)$ of constant depth $k \cdot |Q|$, which is possible in polynomial time in the data complexity.

The *chase graph* for D and Σ is the directed graph consisting of $chase(D, \Sigma)$ as the set of nodes and having an arrow from a to b iff b is obtained from a and possibly other atoms by a one-step application of a TGD in Σ.

Datalog+/– Ontologies. A *Datalog+/– ontology* $KB = (D, \Sigma)$, where $\Sigma = \Sigma_T \cup \Sigma_{NC}$, consists of a database D, a finite set of TGDs Σ_T, and a finite set of negative constraints and non-conflicting EGDs Σ_{NC}. We say KB is *guarded* (resp., *linear*) iff Σ_T is guarded (resp., linear). Example 1 illustrates a simple Datalog+/– ontology, which is used in the sequel as a running example.

Depending on the expressive power of the underlying formalism, some works on inconsistency handling in DLs allow for both terminological axioms (TBox) and assertional axioms (ABox) to be inconsistent. In this work, we make the usual assumption that Σ contains integrity constraints expressing the semantics of the data in D, and thus that Σ is itself consistent; inconsistencies can only arise when D and Σ are considered together. We now define the notion of *consistency* in Datalog+/– ontologies.

Definition 1 (Consistency) A Datalog+/– ontology $KB = (D, \Sigma)$ is *consistent* iff $mods(D, \Sigma) \neq \emptyset$.

Note that if $\Sigma_{NC} = \emptyset$, then $mods(D, \Sigma)$ is not empty. Different works on inconsistency handling in DLs allow for inconsistency to occur for different reasons.

Normalization of Negative Constraints.

Definition 2 Let Σ_{NC} be a set of negative constraints and non-conflicting EGDs, $\upsilon \in \Sigma_{NC}$, and Q be a BCQ. Let $\sim_\upsilon$ be an equivalence relation on the arguments in the body of υ and the constants in Σ_{NC} and Q such that every equivalence class contains at most one constant. A *normalization instance* of υ relative to such $\sim_\upsilon$ is obtained from υ by replacing every argument in the body of υ by a representative of its equivalence class (which is a constant if the equivalence class contains a constant) and adding to the body the conjunction of all $s \neq t$ for any two different representatives s and t such that s is a variable occurring in the instance, and t is either a variable occurring in the instance or a constant in Σ_{NC} and Q. The *normalization* of υ, denoted $\mathcal{N}(\upsilon, Q)$, is the set of all such instances of υ subject to all equivalence relations $\sim_\upsilon$. The *normalization* of Σ_{NC} is $\mathcal{N}(\Sigma_{NC}, Q) = \bigcup_{\upsilon \in \Sigma_{NC}} \mathcal{N}(\upsilon, Q)$.

Example 2 *Consider the set of negative constraints* $\Sigma_{NC} = \{\upsilon_1 : p(U, U) \rightarrow \bot,\ \upsilon_2 : p(X, Y) \wedge q(X) \rightarrow \bot\}$ *and the BCQ* $Q = \exists X q(X)$. *Its normalization is:*

$$\mathcal{N}(\Sigma_{NC}, Q) = \{\upsilon'_1 : p(U, U) \rightarrow \bot,\ \upsilon'_2 : p(X, X) \wedge q(X) \rightarrow \bot,\ \upsilon'_3 : p(X, Y) \wedge q(X) \wedge\ X \neq Y \rightarrow \bot\}.$$

■

3 Inconsistency-Tolerant Query Answering

The area of *belief change* in AI is closely related to the management of inconsistent information; it aims at adequately modeling the dynamics of the knowledge that constitutes the set of beliefs of an agent when new information comes up. In [8], *kernel* consolidations are defined based on the notion of an *incision function*. Given a knowledge base KB that needs to be consolidated (i.e., KB is inconsistent), the set of kernels is defined as the set of all minimal inconsistent subsets of KB. For each kernel, a set of sentences is removed (i.e., an "incision" is made) such that the remaining formulas in the kernel are consistent; note that it is enough to remove any single formula from the kernel because they are minimal inconsistent sets. The result of consolidating KB is then the set of all formulas in KB that are not removed by the incision function. In this work, we present a framework based on a similar kind of functions to provide alternative query answering semantics in inconsistent Datalog+/– ontologies. The main difference in our proposal is that incisions are performed over inconsistent subsets of the ontology that are not necessarily minimal.

We analyze three types of incision functions that correspond to three different semantics for query answering in inconsistent Datalog+/– ontologies: (i) *consistent answers* semantics, widely adopted in relational databases and DLs, (ii) *intersection semantics*, which is a sound approximation of consistent answers [9], and (iii) a new semantics that relaxes the requirements of the consistent answers semantics, allowing it to be computed in polynomial time for some fragments of Datalog+/–, without compromising the quality of the answers as much as the intersection semantics does.

We first define the notion of a *culprit* relative to a set of constraints IC, which is informally a minimal (under set inclusion) inconsistent subset of the database relative to IC. Note that we define culprits relative to both negative constraints and EGDs.

Definition 3 (Culprit) Given a Datalog+/– ontology $KB = (D, \Sigma_T \cup \Sigma_{NC})$ and $IC \subseteq \Sigma_{NC}$, a *culprit* in KB relative to IC is a set $c \subseteq D$ such that $mods(c, \Sigma_T \cup IC) = \emptyset$, and there is no $c' \subset c$ such that $mods(c', \Sigma_T \cup IC) = \emptyset$. We denote by $culprits(KB, IC)$ (resp., $culprits(KB)$) the set of culprits in KB relative to IC (resp., $IC = \Sigma_{NC}$).

Example 3 For the ontology KB of the running example, the culprits relative to Σ_{NC} are:
$c_1 = \{supervises(tom, john), directs(john, d_1), works\_in(john, d_1)\}$,
$c_2 = \{supervises(tom, john), directs(john, d_1), works\_in(tom, d_1)\}$,
$c_3 = \{directs(tom, d_1), directs(tom, d_2)\}$. ■

The following result shows that the normalization of negative constraints does not change the culprits of an ontology, even in the additional presence of a set of TGDs Σ_T (where the constants in Σ_T are considered in the same way as those in Σ_{NC} and Q).

Lemma 1 *Let $KB = (D, \Sigma_{NC} \cup \Sigma_T)$ be a Datalog+/– ontology, and Q be a BCQ. Then, $culprits(KB) = culprits(KB')$, with $KB' = (D, \Sigma_T \cup \mathcal{N}(\Sigma_{NC}, Q))$.*

We construct *clusters* by grouping together all culprits that share elements. Intuitively, clusters contain only information involved in some inconsistency relative to Σ, i.e., an atom is in a cluster relative to Σ iff it is in contradiction with some other set of atoms in D.

Definition 4 (Cluster [10]) Given a Datalog+/– ontology $KB = (D, \Sigma_T \cup \Sigma_{NC})$ and $IC \subseteq \Sigma_{NC}$, two culprits $c, c' \in culprits(KB, IC)$ *overlap*, denoted $c \ \Theta \ c'$, iff $c \cap c' \neq \emptyset$. Denote by Θ^* the equivalence relation given by the reflexive and transitive closure of Θ. A *cluster* is a set $cl = \bigcup_{c \in e} c$, where e is an equivalence class of Θ^*. We denote by $clusters(KB, IC)$ (resp., $clusters(KB)$) the set of all clusters in KB relative to IC (resp., $IC = \Sigma_{NC}$).

Example 4 The clusters for KB in the running example are $cl_1 = c_3$ and $cl_2 = c_1 \cup c_2$ (cf. Example 3 for culprits c_1, c_2, and c_3). ■

We now recall the definition of *incision function* from [8], adapted for Datalog+/– ontologies. Intuitively, an incision function selects from each cluster a set of atoms to be discarded such that the remaining atoms are consistent relative to Σ.

Definition 5 (Incision Function) Given a Datalog+/– ontology $KB = (D, \Sigma)$, an *incision function* is a function χ that satisfies the following properties:

(1) $\chi(\mathit{clusters}(KB)) \subseteq \bigcup_{cl \in \mathit{clusters}(KB)} cl$, and

(2) $\mathit{mods}(D - \chi(\mathit{clusters}(KB)), \Sigma) \neq \emptyset$.

Note that incision functions in [8] do not explicitly require condition (2) from Definition 5; instead, they require the removal of at least one sentence from each α-kernel. The notion of α-kernel [8] translates in our framework to a minimal set of sentences in D such that, together with Σ, entails the sentence α, where $KB = (D, \Sigma)$. Culprits are then, no more than minimal subsets of D that, together with Σ, entail $\bot$. Here, χ produces incisions over clusters instead, therefore, condition (2) is necessary to ensure that by making the incision, the inconsistency is resolved.

3.1 Relationship to Consistent Answer Semantics

In the area of relational databases, the notion of *repair* was used in order to identify the consistent part of a possibly inconsistent database. A repair is a model of the set of integrity constraints that is maximally close, i.e., "as close as possible" to the original database. Repairs may not be unique, and in the general case, there can be a very large number of them. The most widely accepted semantics for querying a possibly inconsistent database is that of *consistent answers*.

We now define the notion of *data repairs*, which extends the notion of repairs to Datalog+/– ontologies $KB = (D, \Sigma)$. Intuitively, data repairs are maximal consistent subsets of D. We also show that BCQ answering under the consistent answer semantics is co-NP-complete for guarded and linear Datalog+/– in the data complexity.

Definition 6 (Data Repair) A *data repair* for $KB = (D, \Sigma)$ is a set D' such that:

(i) $D' \subseteq D$,

(ii) $\mathit{mods}(D', \Sigma) \neq \emptyset$, and

(iii) there is no $D'' \subseteq D$ such that $D' \subset D''$ and $\mathit{mods}(D'', \Sigma) \neq \emptyset$.

We denote by $\mathit{DRep}(KB)$ the set of all data repairs for KB.

Example 5 The Datalog+/– ontology KB in Example 1 has six data repairs:
$r_1 = \{\mathit{directs}(john, d_1), \mathit{supervises}(tom, john), \mathit{directs}(tom, d_1), \mathit{manager}(tom, d_1)\}$,
$r_2 = \{\mathit{directs}(john, d_1), \mathit{supervises}(tom, john), \mathit{directs}(tom, d_2), \mathit{manager}(tom, d_1)\}$,
$r_3 = \{\mathit{directs}(john, d_1), \mathit{directs}(tom, d_1), \mathit{works\_in}(john, d_1), \mathit{works\_in}(tom, d_1),$
$\mathit{manager}(tom, d_2)\}$,
$r_4 = \{\mathit{directs}(john, d_1), \mathit{directs}(tom, d_2), \mathit{works\_in}(john, d_1), \mathit{works\_in}(tom, d_1),$
$\mathit{manager}(tom, d_2)\}$,
$r_5 = \{\mathit{supervises}(tom, john), \mathit{directs}(tom, d_1), \mathit{works\_in}(john, d_1),$
$\mathit{works\_in}(tom, d_1), \mathit{manager}(tom, d_1)\}$,
$r_6 = \{\mathit{supervises}(tom, john), \mathit{directs}(tom, d_2), \mathit{works\_in}(john, d_1),$
$\mathit{works\_in}(tom, d_1), \mathit{manager}(tom, d_1)\}$. ■

Data repairs play a central role in the notion of *consistent answer* for a query to an ontology, which are intuitively the answers relative to each ontology built from a data repair.

Definition 7 (Consistent Answers) Let $KB = (D, \Sigma)$ be a Datalog+/– ontology, and Q be a BCQ. Then, *Yes* is a *consistent answer* for Q to KB, denoted $KB \models_{Cons} Q$, iff it is an answer for Q to each $KB' = (D', \Sigma)$ with $D' \in \mathit{DRep}(KB)$.

Example 6 Consider the ontology KB from our running example. The atom $\mathit{emp}(john)$ can be derived from every data repair, as each contains either the atom $\mathit{works\_in}(john, d_1)$ or the atom $\mathit{directs}(john, d_1)$. Thus, BCQ $Q = \mathit{emp}(john)$ is true under the consistent answer semantics. ■

In accordance with the principle of *minimal change*, incision functions that make as few changes as possible when applied the set of clusters are called *optimal* incision functions.

Definition 8 (Optimal Incision Function) Given a Datalog+/– ontology $KB = (D, \Sigma)$, an incision function χ is *optimal* iff for every $B \subset \chi(\mathit{clusters}(KB))$, it holds that $mods(D - B, \Sigma) = \emptyset$.

The following theorem shows the relationship between an optimal incision function and data repairs for a Datalog+/– ontology $KB = (D, \Sigma)$. More concretely, every data repair corresponds to the result of removing from D all ground atoms according to some optimal incision $\chi(\mathit{clusters}(KB))$ and vice versa.

Theorem 1 *Let $KB = (D, \Sigma)$ be a Datalog+/– ontology. Then, D' is a data repair, i.e., $D' \in \mathit{DRep}(KB)$, iff there exists an optimal incision function χ_{opt} such that $D' = D - \chi_{opt}(\mathit{clusters}(KB))$.*

The next result shows that deciding consistent answers for guarded and linear Datalog+/– is co-NP-complete in the data complexity.

Theorem 2 *Given a guarded Datalog+/– ontology KB and a BCQ Q, deciding whether $KB \models_{Cons} Q$ is* co-NP-*complete in the data complexity. Hardness hold even when KB is linear.*

3.2 Relationship to Intersection Semantics

An alternative semantics that considers only the atoms that are in the *intersection* of all data repairs was presented in [9] for *DL-Lite* ontologies. This semantics yields a unique way of repairing inconsistency; the consistent answers are intuitively the answers that can be obtained from that unique set. Here, we define the *intersection semantics* for Datalog+/– ontologies KB.

Definition 9 (Intersection Semantics) Let $KB = (D, \Sigma)$ be a Datalog+/– ontology, and Q be a BCQ. Then, *Yes* is a *consistent answer* for Q to KB *under the intersection semantics*, denoted $KB \models_{ICons} Q$, iff it is an answer for Q to $KB_I = (D_I, \Sigma)$, where $D_I = \bigcap\{D' \mid D' \in \mathit{DRep}(KB)\}$.

Example 7 Consider the Datalog+/– ontology $KB = (D, \Sigma)$ of the running example. Analyzing the set of all its data repairs, it is easy to verify that $D_I = \{\mathit{manager}(tom, d_1)\}$. ■

The following theorem shows the relationship between the incision function χ_{all}, which is defined by $\chi_{all}(\mathit{clusters}(KB)) = \bigcup_{cl \in \mathit{clusters}(KB)} cl$, and consistent answers under the intersection semantics. Intuitively, answers relative to the intersection semantics can be obtained by removing from D all atoms participating in some cluster, and answering the query using the resulting database.

Theorem 3 *Let $KB = (D, \Sigma)$ be a Datalog+/– ontology, and Q be a BCQ. Then, we have $KB \models_{\mathit{ICons}} Q$ iff $(D - \chi_{all}(\mathit{clusters}(KB)) \cup \Sigma \models Q$.*

The next result shows that deciding consistent answers on guarded Datalog+/– ontologies is co-NP-complete in the data complexity. Note that in Section 4.1, we show that deciding whether $KB \models_{\mathit{ICons}} Q$ becomes tractable for linear Datalog+/– ontologies.

Theorem 4 *Given a guarded Datalog+/– ontology KB and a BCQ Q, deciding whether $KB \models_{\mathit{ICons}} Q$ is* co-NP*-complete in the data complexity.*

4 Lazy Answers

In the following, we present a new semantics for consistent query answering in Datalog+/– ontologies. The motivation behind this new semantics is that the exact procedure is too expensive to be computed in any reasonable-sized Datalog+/– ontology, but at the same time the intersection semantics is unnecessarily restrictive in the set of answers that can be obtained from a query. We propose an alternative to consistent query answering in Datalog+/– ontologies, under which answers are at least as complete as those that can be obtained under the intersection semantics.

We first define the notion of k-*cut* of clusters. Let $\chi_{k\text{-}cut}$ be a function defined as follows for $cl \in \mathit{cluster}(KB)$:

$$\chi_{k\text{-}cut}(cl) = \begin{cases} \{C_1, \ldots, C_m\} & m \geqslant 1, C_i \subset cl, |C_i| \leqslant k, \\ & s.t.\ mods(cl - C_i, \Sigma) \neq \emptyset \\ & and\ \nexists C'_i\ s.t.\ C'_i \subset C\ and \\ & mods(cl - C'_i, \Sigma) \neq \emptyset\}; \\ \{cl\} & \text{if no such } C_i \text{ exists.} \end{cases} \tag{1}$$

Intuitively, given a cluster cl, its k-cut $\chi_{k\text{-}cut}(cl)$ is the set of minimal subsets of cl of cardinality at most k, such that if they are removed from cl, what is left is consistent with respect to Σ.

We next use the k-cut of clusters to define a new type of incision functions, called k-*lazy functions*, as follows.

Definition 10 Let $KB = (D, \Sigma)$ be a Datalog+/– ontology, and $k \geqslant 0$. A k-*lazy function* for KB is defined as $\chi_{lazy}(k, \mathit{clusters}(KB)) = \bigcup_{cl \in \mathit{clusters}(KB)} c_{cl}$, where $c_{cl} \in \chi_{k\text{-}cut}(cl)$.

The above k-lazy functions are indeed incision functions.

Proposition 1 *Let $KB = (D, \Sigma)$ be a Datalog+/– ontology, and $k \geqslant 0$. All k-lazy functions for KB are incision functions.*

The function χ_{lazy} is the basis of *lazy repairs*, as defined next. Intuitively, k-lazy repairs are built by analyzing ways in which to remove at most k atoms in every cluster.

Definition 11 (k-Lazy Repair) Let $KB = (D, \Sigma)$ be a Datalog+/– ontology, and $k \geqslant 0$. A k-*lazy repair* for KB is any set $D' = D - \chi_{lazy}(k, \mathit{clusters}(KB))$, where $\chi_{lazy}(k, \mathit{clusters}(KB))$ is a k-lazy function for KB. $\mathit{LRep}(k, KB)$ denotes the set of all such repairs.

Example 8 Consider again our running example and the clusters in KB from Example 4. Let $k=1$, then we have that $\chi_{1\text{-}cut}(cl_1) = \{d_1\colon \{\mathit{directs}(tom, d_1)\},\ d_2\colon \{\mathit{directs}(tom, d_2)\}\}$, and that $\chi_{1\text{-}cut}(cl_2) = \{e_1\colon \{\mathit{supervises}(tom, john)\},\ e_2\colon \{\mathit{directs}(john, d_1)\}\}$. There are four possible incisions: $\mathit{ins}_1 = d_1 \cup e_1$, $\mathit{ins}_2 = d_1 \cup e_2$, $\mathit{ins}_3 = d_2 \cup e_1$, and $\mathit{ins}_4 = d_2 \cup e_2$. Thus, there are four 1-lazy repairs, with $\mathit{lrep}_i = D - \mathit{ins}_i$; for example, $\mathit{lrep}_1 = \{\mathit{directs}(john, d_1),\ \mathit{directs}(tom, d_2),\ \mathit{works\_in}(john, d_1),\ \mathit{works\_in}(tom, d_1),\ \mathit{manager}(tom, d_1)\}$. ■

We can now define *k-lazy answers* for a query Q as the set of atoms that are derived from every k-lazy repair.

Definition 12 (k-Lazy Answers) Let $KB = (D, \Sigma)$ be a Datalog+/– ontology, Q be a BCQ, and $k \geqslant 0$. Then, *Yes* is a *k-lazy answer* for Q to KB, denoted $KB \models_{k\text{-}LCons} Q$, iff it is an answer for Q to each $KB' = (D', \Sigma)$ with $D' \in \mathit{LRep}(k, KB)$.

Section 4.1 shows that lazy answers for atomic BCQs can be computed efficiently in linear Datalog+/–. The following proposition states some properties of k-lazy repairs and lazy answers: each lazy repair is consistent relative to Σ, and only atoms that contribute to an inconsistency are removed by a k-lazy function for KB.

Proposition 2 *Let $KB = (D, \Sigma)$ be a Datalog+/– ontology, and $k \geqslant 0$. Then, for every $D' \in \mathit{LRep}(k, KB)$, (a) $mods(D', \Sigma) \neq \emptyset$, and (b) if $\beta \in D$ and $\beta \notin D'$, then there exists some $B \subseteq D$ such that $mods(B, \Sigma) \neq \emptyset$ and $mods(B \cup \{\beta\}, \Sigma) = \emptyset$.*

Proposition 2 shows that lazy repairs satisfy properties that are desirable for any belief change operator to have [8]. However, the incisions performed by function $\chi_{lazy}(k, \mathit{clusters}(KB))$ are not always, minimal relative to set inclusion, i.e., if there is no subset of a cluster of size at most k that satisfies the conditions in Definition 1, then the whole cluster is removed, and therefore not every lazy repair is a data repair.

The consistent answers semantics from Definition 7 is a *cautious semantics* to query answering, since only answers that can be entailed from *every* data repair are deemed consistent. Traditionally, the alternative to this semantics is a *brave* approach, which in our framework would consider an answer as consistent if it can be entailed from *some* data repair. In the case of Example 8 with $k=1$, $\mathit{works\_in}(john, d_1)$ and $\mathit{works\_in}(tom, d_1)$ are lazy consequences of KB, which are clearly not consistent consequences of KB. However, a brave approach for query answering would allow both $\mathit{supervise}(tom, john)$ and $\mathit{directs}(john, d_1)$ as answers. In this respect, lazy answers are a compromise between brave and cautious approaches: although it is "braver" than the cautious approach, *it does not allow to derive mutually inconsistent answers.*

Proposition 3 *Let $KB = (D, \Sigma)$ be a Datalog+/– ontology, Q be a CQ, and $ans_{LCons}(k, Q, D, \Sigma)$ be the set of lazy answers for Q given k. Then, for any $k \geqslant 0$, $mods(ans_{LCons}(k, Q, D, \Sigma), \Sigma) \neq \emptyset$.*

Proof. Suppose by contradiction that $mods(ans_{LCons}(k, Q, D, \Sigma), \Sigma) = \emptyset$, then there is a negative constraint $\upsilon \in \Sigma_{NC}$ for which $body(\upsilon)$ maps through homomorphism h to some atoms in $ans_{LCons}(k, Q, D, \Sigma)$; let these atoms be $b_1, \ldots, b_n$. If $\{b_1, \ldots, b_n\} \subset ans_{LCons}(k, Q, D, \Sigma)$, then by Definition 12, we have that $(D', \Sigma) \models b_1 \wedge \ldots \wedge b_n$ for every $D' \in \mathit{LRep}(k, KB)$. But this means that $body(\upsilon)$ maps to atoms in D' for every $D' \in \mathit{LRep}(k, KB)$, and therefore $mods(D', \Sigma) = \emptyset$, which contradicts result (a) in Proposition 2. □

The next proposition shows that the same property holds if we consider the union of k-lazy answers for different values of k.

Theorem 5 *Let $KB = (D, \Sigma)$ be a Datalog+/– ontology and a Q a CQ. If for any $k \geqslant 0$ we have $\mathcal{U}_k = \bigcup_{0 \leqslant i \leqslant k} ans_{LCons}(k, Q, D, \Sigma)$, then we have that $mods(\mathcal{U}_k, \Sigma) \neq \emptyset$.*

Proof. We proceed by induction on k. Without loss of generality, in the following we assume that Σ_{NC} is a normalized set of negative constraints (cf. Definition 2).

Base Case: If $k = 0$, then by Proposition 2, we have that $ans_{LCons}(0, Q, D, \Sigma)$ is clearly consistent.

Inductive Case: Suppose that $mods(\bigcup_{0 \leqslant i \leqslant k-1} ans_{LCons}(i, Q, D, \Sigma), \Sigma) \neq \emptyset$; we then want to prove that $mods(\bigcup_{0 \leqslant i \leqslant k} ans_{LCons}(i, Q, D, \Sigma), \Sigma) \neq \emptyset$.

Suppose by contradiction that $mods(\bigcup_{0 \leqslant i \leqslant k} ans_{LCons}(i, Q, D, \Sigma), \Sigma) = \emptyset$. By definition, we have that

$$\bigcup_{0 \leqslant i \leqslant k} ans_{LCons}(i, Q, D, \Sigma) = \bigcup_{0 \leqslant i \leqslant k-1} ans_{LCons}(i, Q, D, \Sigma) \cup ans_{LCons}(k, Q, D, \Sigma)$$

and, by the inductive hypothesis and Proposition 2, respectively, we have $\bigcup_{0 \leqslant i \leqslant k-1} ans_{LCons}(i, Q, D, \Sigma)$ and $ans_{LCons}(k, Q, D, \Sigma)$ are consistent relative to Σ. Then it must be the case that the inconsistency appears as the result of taking the union of both sets. This is, there exists $A \subseteq \bigcup_{0 \leqslant i \leqslant k-1} ans_{LCons}(i, Q, D, \Sigma)$ and $B \subseteq ans_{LCons}(k, Q, D, \Sigma)$ such that for some $v \in \Sigma_{NC}$, $body(v)$ homomorphically maps to $A \cup B$ (and it does not map to any proper subset of $A \cup B$). Also, it must that $B \not\subseteq \bigcup_{0 \leqslant i \leqslant k-1} ans_{LCons}(i, Q, D, \Sigma)$ and $A \not\subseteq ans_{LCons}(k, Q, D, \Sigma)$, otherwise it would contradict the inductive hypothesis and Proposition 2.

Now, we look at the k-lazy repairs for KB; if all of them are i-lazy repairs for some $i < k$, then we have $(\bigcup_{0 \leqslant i \leqslant k-1} ans_{LCons}(i, Q, D, \Sigma), \Sigma) \models B$, which is again a contradiction. Therefore, there are new lazy repairs in $LRep(k, KB)$ such that they all derive B, which means that there must exist a cluster $cl \in clusters(KB, \Sigma)$ such that $(cl, \Sigma) \models B$ and there is no $k-1$-cut c such that $mods(cl - c, \Sigma) \neq \emptyset$. However, if this is the case, since A and B are inconsistent together, by Definition 4 we have that $(cl, \Sigma) \models A$ as well; but this is a contradiction since it implies that $A \not\subseteq \bigcup_{0 \leqslant i \leqslant k-1} ans_{LCons}(i, Q, D, \Sigma)$, since there is no i-cut with $i < k$ that satisfies Equation 1 for cl, and thus $\chi_{i-cut}(cl) = \{cl\}$ for all $i < k$. □

Theorem 5 shows that lazy answers can be used to obtain answers that are not consistent answers but are nevertheless consistent as a whole. We say that a (B)CQ Q is entailed under the *union-k-lazy semantics* for some $k \geqslant 0$ iff $(\mathcal{U}_k, \Sigma) \models Q$.

We now analyze the relationship between the different semantics discussed in this paper. The following proposition shows the relationship between the intersection semantics and the lazy semantics.

Proposition 4 *Let $KB = (D, \Sigma)$ be a Datalog+/– ontology, and Q be a BCQ. Then, (a) if $KB \models_{ICons} Q$, then $KB \models_{k\text{-}LCons} Q$, for any $k \geqslant 0$, and (b) $KB \models_{ICons} Q$ iff $KB \models_{0\text{-}LCons} Q$.*

We next show how consistent and lazy answers are related.

Proposition 5 *Let $KB = (D, \Sigma)$ be a Datalog+/– ontology, and Q be a BCQ. There is $k \geqslant 0$ such that $KB \models_{Cons} Q$ iff $KB \models_{k\text{-}LCons} Q$.*

Clearly, Proposition 5 entails that if we take the union of the lazy answers up to the k from the proposition, then the resulting set of lazy answers is complete with respect to the consistent answers.

Example 9 shows that, in our running example, the 2-lazy answers correspond exactly to the consistent answers.

Example 9 In Example 8, if $k = 2$, then we have that $\chi_{2\text{-}cut}(cl_1) = \chi_{1\text{-}cut}(cl_1)$ and $\chi_{2\text{-}cut}(cl_2) = \chi_{1\text{-}cut}(cl_2) \cup \{\{works\_in(tom, d_1), works\_in(john, d_1)\}\}$. We can easily see that $LRep(2, KB) = DRep(KB)$. ■

The following (simpler) example shows the effects of changing the value of k as well as the results from Theorem 5.

Example 10 Consider the CQ $Q(X,Y) = p(X) \wedge q(Y)$ and an Datalog+/– ontology $KB = (D, \Sigma)$:

$$\begin{aligned} D &= \{p(a),\, p(b),\, p(c),\, p(d),\, p(e),\, p(f),\, q(g),\, q(h),\, q(i),\, q(j)\};\\ \Sigma_T &= \{\};\\ \Sigma_{NC} &= \{p(a) \wedge p(b) \rightarrow \bot,\, p(b) \wedge p(d) \rightarrow \bot,\, p(d) \wedge p(e) \rightarrow \bot,\, p(d) \wedge p(f) \rightarrow \bot\\ &\quad q(g) \wedge q(h) \rightarrow \bot,\, q(h) \wedge q(i) \rightarrow \bot\} \end{aligned}$$

The set of clusters in *KB* is *clusters*$(KB, \Sigma) = \{cl_1 : \{p(a), p(b), p(d), p(e), p(f)\}, cl_2 : \{q(g), q(h), q(i)\}$. For $k = 0$, the only 0-lazy repair is *lrep*$_0 = \{p(c), q(j)\}$, which coincides with D_I; the set of 0-lazy answers (and the answers under the intersection semantics) to $Q(X,Y)$ is $\{p(c), q(j)\}$.

For $k = 1$, note that there is no way of removing one element from cl_1 making the rest consistent; therefore, the only possible cut removes the whole cluster. On the other hand, there is one 1-cut for cl_2, namely $\{q(h)\}$. Therefore, we have only one 1-lazy repair *lrep*$_1 = \{p(c), q(j), q(i), q(g)\}$. The set of 1-lazy answers to $Q(X,Y)$ is $\{p(c), q(j), q(i), q(g)\}$.

With $k = 2$, there are two 2-cuts for cl_1 and two for cl_2: $\chi_{2\text{-}cut}(cl_1) = \{\{p(a), p(d)\}, \{p(b), p(d)\}\}$ and $\chi_{2\text{-}cut}(cl_2) = \{\{q(h)\}, \{q(g), q(i)\}\}$. In this case there are four 2-lazy repairs and the set of 2-lazy answers to $Q(X,Y)$ is $\{p(c), p(e), p(f), q(j)\}$.

For $k = 3$, we have $\chi_{3\text{-}cut}(cl_1) = \{\{p(a), p(d)\}, \{p(b), p(d)\}, \{p(b), p(e), p(f)\}\}$ and $\chi_{3\text{-}cut}(cl_2) = \chi_{2\text{-}cut}(cl_2) = \{\{q(h)\}, \{q(g), q(i)\}\}$. The set of 3-lazy repairs coincide with the set of repairs and therefore the set of 3-lazy answers to $Q(X,Y)$ is the set of consistent answers, namely $\{p(c), q(j)\}$.

Finally, $\bigcup_{0 \leqslant i \leqslant 3} ans_{LCons}(i, Q, D, \Sigma) = \{p(c), q(j), q(i), q(g), p(e), p(f)\}$, which is clearly consistent relative to Σ_{NC}. ■

The following theorem shows that BCQ answering for guarded Datalog+/– ontologies under k-lazy semantics is co-NP-hard. Despite this result, we show in the next section that for linear Datalog+/–, atomic BCQ answering under k-lazy answers is tractable.

Theorem 6 *Given a guarded Datalog+/– ontology KB and a BCQ Q, deciding KB* $\models_{k\text{-}LCons}$ *Q is* co-NP-*hard in the data complexity.*

After the formal presentation of lazy answers, based on the concept of incision functions, we can now provide an algorithm that computes lazy answers to conjunctive queries to Datalog+/– ontologies. The algorithm uses the concept of *finite chase graph* [3] for a given ontology $KB = (D, \Sigma)$, which is a graph consisting of the necessary finite part of $chase(D, \Sigma)$ relative to query Q, i.e., the finite part of the chase graph for D and Σ such that $chase(D, \Sigma) \models Q$.

Algorithm lazyConsistent (see Fig. 1) first computes the set of clusters in *KB* using subroutine findClusters. Next, for each cluster, function $\chi_{k\text{-}cut}$ is constructed by removing each possible subset (of size at most k) of the cluster in turn and checking if the remaining tuples are consistent (and that the subset in question is not a superset of an incision already found – cf. line 7). If so, then the set is added as a possible "incision" for the cluster. The loop in line 10 considers all possible ways of choosing one incision for each cluster that has incisions associated with it. A lazy repair then arises from each such possible combination by removing the incisions from D. The answer is finally computed using these repairs.

Algorithm findClusters (Figure 2) computes the set of clusters in a given guarded Datalog+/– ontology. The algorithm works in the following way: for each ground negative constraint, the chase graph w.r.t. the constraint's body is obtained, i.e., taking the body of a constraint as a conjunctive query. If

```
Algorithm lazyConsistent($KB = (D, \Sigma), Q = (\bigwedge_{i=1}^{n} a_i), k$)
 1. incisions:= new (empty) mapping of type $\langle$atomSet, set of atomSet$\rangle$;
 2. LRepairs:= new (empty) set of type atomSet;
 3. clusters:= findClusters($KB$);
 4. for every $cl \in clusters$ do
 5.     for $i = 1$ to $\min(k, |cl|)$ do
 6.         for every set $A \subseteq cl$ with $|A| = i$ do
 7.             if ($\nexists A' \in incisions(cl)$ such that $A' \subset A$) and
                  $mods(cl - A, \Sigma) \neq \emptyset$ then
 8.                 set $incisions(cl) := incisions(cl) \cup A$ ;
 9. fixedClusters := union of all sets $C \in clusters$ such that $incisions(C) = \emptyset$;
10. for every set of atoms $R_i$ resulting from every possible way of choosing an
             atom from each $v$ in $domain(incisions)$ do
11.     LRepairs:= $LRepairs \cup \{(D - R_i) - fixedClusters\}$;
12. return Yes iff Yes is an answer for $Q$ to $(lrep, \Sigma)$ $\forall lrep \in LRepairs$;
```

Figure 1: Computing k-lazy answers for a BCQ Q to KB.

$chase(KB, body(\upsilon)) \models body(\upsilon)$, then we know that constraint υ is violated in KB. Note that the grounding of negative constraints can be obtained by producing a query $Q(\mathbf{X})$ consisting of the body of the non-ground constraint with $\mathbf{X}$ containing all universally quantified variables in the constraint. The set of answers for $Q(\mathbf{X})$ over KB gives us the constants needed to correctly ground the set of constraints. As we mentioned above, the notion of α-kernel comes from [11] and it represents a minimal set of sentences in the knowledge base that entail sentence α. In the same spirit, algorithm findKernels (Figure 3) allows to compute the set of minimal subsets of D (ground atoms belonging to level 0 of the chase graph G) that, together with Σ_T, entail the body of a ground constraint υ; that is, $body(\upsilon)$-kernels correspond to the set $culprits(KB, \{\upsilon\})$. These sets are identified by "following up" the edges in the chase graph from the atoms in $body(\upsilon)$ until D (level 0 in the chase graph) is reached. Note however, that we cannot just do the same for every constraint and then merge the resulting sets that overlap in order to compute the clusters. The problem is that, in general, it is not that case that $culprits(KB) = \bigcup_{\upsilon \in \Sigma_{NC}} culprits(KB, \{\upsilon\})$. For this reason Algorithm findClusters performs further checking of minimality among "potential" culprits before merging overlapping ones. The following example shows how Algorithm findClusters works over our running example.

Example 11 Consider the Datalog+/– ontology from Example 1 with the addition of the following constraint (only for this example) υ_4 : $manager(X, D) \wedge manager(Y, D) \wedge supervises(X, Y) \rightarrow X = Y$; formula υ_4 says that two different persons cannot be both managers and one supervise the other at the same time. Algorithm findClusters first computes the set of ground negative constraints corresponding to KB. We have then, $\Sigma'_{NC} =$

$\{ \upsilon_1 : supervises(tom, john) \wedge manager(john, d_1) \rightarrow \bot,$
$\upsilon_2 : supervises(tom, john) \wedge works\_in(tom, d_1) \wedge directs(john, d_1) \rightarrow \bot,$
$\upsilon_3 : directs(tom, d_1) \wedge directs(tom, d_2) \wedge d_1 \neq d_2 \rightarrow \bot,$
$\upsilon_4 : manager(tom, d_1) \wedge manager(john, d_1) \wedge supervises(tom, john) \wedge tom \neq john \rightarrow \bot\}$

Second, the chase graph for each individual ground constraint is computed with respect to KB. Figure 4 shows the chase graph computed for constraint $\upsilon_1 \in \Sigma'_{NC}$, i.e, $G = chase(KB, Q() = body(\upsilon_1))$.

The calls to Algorithm findKernels within Algorithm findClusters compute the following sets:

```
Algorithm findClusters($KB = (D, \Sigma)$)
 1. $PCul = \emptyset$;
 2. Let $\Sigma'_{NC}$ be the set of ground negative constraints and EGDs from $KB$;
 3. for every ground $\upsilon \in \Sigma'_{NC}$ do
 4.     compute chase graph $g$:= $chaseGraph\,(body(\upsilon), D, \Sigma_T)$;
 5.     $ker_\upsilon$:= $new\ (empty)$ mapping of type $\langle atom, atomSet\rangle$;
 6.     $PC_\upsilon$:= findKernels$(\Sigma, g, body(\upsilon), ker_\upsilon)$;
 7.     for every $c \in PC_\upsilon$ do
 8.         if no $c' \in PCul$ is such that $c' \subset c$ then
 9.             $PCul = PCul \cup \{c\}$;
10.         if there exists $c' \in PCul$ such that $c \subset c'$ then
11.             $PCul = (PCul - c') \cup \{c\}$;
12. Merge pairs of elements in $PCul$ that overlap (repeat until no more exist);
13. return $PCul$;
```

Figure 2: Computing clusters of a guarded Datalog+/– ontology KB.

findKernels$(\Sigma_T, G, body(\upsilon_1)) = \{cul_1 : \{supervises(tom, john), directs(john, d_1), works_in(john, d_1)\}\}$,
findKernels$(\Sigma_T, G, body(\upsilon_2)) = \{cul_2 : \{supervises(tom, john), directs(john, d_1), works_in(tom, d_1)\}\}$,
findKernels$(\Sigma_T, G, body(\upsilon_3)) = \{cul_3 : \{directs(tom, d_1), directs(tom, d_2)\}\}$,
findKernels$(\Sigma_T, G, body(\upsilon_4)) = \{cul_4 : \{supervises(tom, john), directs(john, d_1), works_in(john, d_1), manager(tom, d_1)\}, cul_5 : \{supervises(tom, john), directs(john, d_1), works_in(john, d_1), works_in(tom, d_1), directs(tom, d_1)\}\}$.

After analyzing constraints υ_1, υ_2, and υ_3, we have that $PCul = \{cul_1, cul_2, cul_3\}$. The minimality check in line 6 fails for cul_4 and cul_5 since $cul_1 \subset cul_4$ and $cul_1 \subset cul_5$, then neither cul_4 nor cul_4 are added to $PCul$. When the algorithm reaches line 11 we have $PCul = culprits(KB)$ (cf. Example 3), and finally the algorithm returns $\{cul_1 \cup cul_2, cul_3\} = clusters(KB)$ (cf. Example 4). ■

The following proposition shows that Algorithm findClusters correctly computes the set of clusters in KB.

Proposition 6 *Let $KB = (D, \Sigma)$ be a guarded Datalog+/– ontology. Algorithm* findClusters*(KB) correctly computes the set $clusters(KB)$.*

The next result shows that lazyConsistent is correct.

Theorem 7 *Let $KB = (D, \Sigma)$ be a guarded Datalog+/– ontology, Q be a BCQ, and $k \geqslant 0$. Algorithm* lazyConsistent *correctly computes the k-lazy answers for Q to KB.*

4.1 Tractable k-lazy answers for Linear Datalog+/–

In this section we analyze k-lazy answers for a particular sublanguage of Datalog+/– called *linear* Datalog+/– ; linear Datalog+/– consists of linear TGDs, i.e., TGDs that have a unique atom in the body. We show that for linear Datalog+/– ontologies k-lazy answers for BCQs can be computed efficiently, i.e., in polynomial time in the data complexity.

```
**Algorithm** findKernels($TGDs\ \Sigma_T, chaseGraph\ G, Ground\ \alpha = \bigwedge_{i=1,\dots,n} a_i,$
                          $Mapping\ of\ type\ \langle atom, atomSet\rangle\ kernels$)
 1. for every $a_i \in \alpha$ such that $a_i$ belongs to level 0 of $G$ do
 2.     insert $(a_i, \{\{a_i\}\})$ in kernels;
 3. for every TGD $\sigma \in \Sigma_T$ do
 4.     for every $a_i \in \alpha$ such that homomorphism $h$ maps $head(\sigma)$ to $a_i$ do
 5.         let $b_1, ..., b_m$ be nodes in $G$ such that $\exists$ an edge $(b_j, a_i)$ in $G$ with label $\sigma$;
 6.         $ker_\sigma$:= findKernels$(G, h(\bigwedge_{i=1,\dots,m} b_i), kernels)$;
 7.         if $kernels[a_i] \neq null$ then $kernels[a_i]$:= $kernels[a_i] \cup ker_\sigma$;
 8.         else insert $(a_i, ker_\sigma)$ in kernels;
 9. $out$:= $\{ \bigcup_{a_i \in \alpha} ker_i \mid ker_i \in kernels[a_i] \}$;
10. remove from out non minimal sets i.e., no set $c$ remains such that $c \subseteq c'$ and $c' \in out$;
11. return out;
```

Figure 3: Computing the α-kernels for a conjunction of ground atoms α in a guarded Datalog+/– ontology.

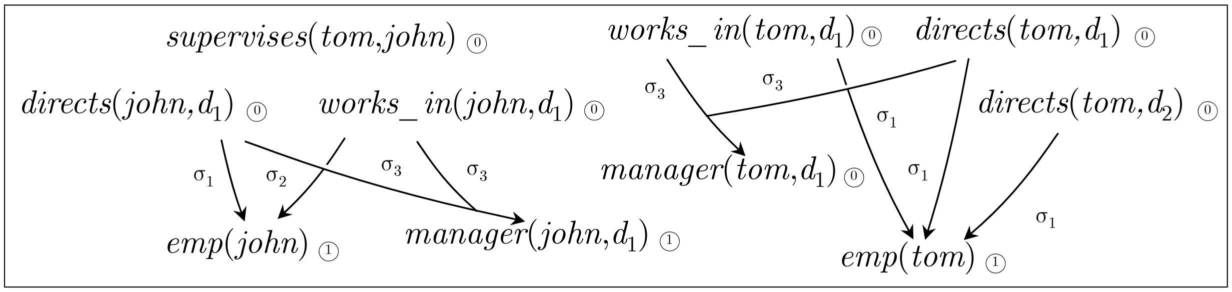

Figure 4: Chase graph for the Datalog+/– ontology KB from Example 1.

The following result shows that the clusters in a linear Datalog+/– ontology can be computed in polynomial time. The main reason for this is that due to the restrictions of linear TGDs there is only a polynomial number of ways of deriving an atom, and since we assume that the set Σ is fixed, we can follow up the derivations of each atom in the body of each constraint in Σ_{NC} in polynomial time in the size of the database.

Proposition 7 *Let $KB = (D, \Sigma)$ be a linear Datalog+/– ontology. Algorithm* findClusters*(KB) runs in polynomial time in $|D|$.*

We can also show that k-lazy answers can be decided in polynomial time in the data complexity. This refers to the parameterized complexity of the problem (the size of the cut as the parameter), and follows, in part, from the set of clusters and their k-cuts being computable in polynomial time in the data complexity for linear Datalog+/–. Furthermore, in the linear case, derivations from different clusters without their k-cuts can be handled independently from each other. This property follows directly from the following result.

Lemma 2 *Let $KB = (D, \Sigma)$ be a linear Datalog+/– ontology, Q a BCQ and $S = s_1, \dots, s_n$ be such that $\bigcup_{s_i \in S} s_i \subseteq D$ and the s_i's are pairwise disjoint. Then, $\bigcup_{s_i \in S} chase(s_i, \Sigma) = chase(\bigcup_{s_i \in S} s_i, \Sigma)$.*

Algorithm lazyConsistent can be adapted to run in polynomial time as shown in Figure 5. It is important to note that the property mentioned above, exploited in the algorithm, works only for linear TGDs; in

```
Algorithm lazyConsistentLinear$(KB = (D, \Sigma), Q = (\bigwedge_{i=1}^{n} a_i), k)$
 1. incisions:= new (empty) mapping of type $\langle \mathrm{atomSet}, \mathrm{set\ of\ atomSet} \rangle$;
 2. clusters:= findClusters$(KB)$;
 3. compute mapping incisions; // as done in lines 4–8 in Algorithm lazyConsistent
 4. if $(D_I = D - \bigcup_{cl \in clusters} cl, \Sigma) \models Q$ then return Yes;
 5. cons:= $\emptyset$;
 6. for every $cl \in clusters$ do
 7.     if $incisions(cl) \neq \emptyset$ then
 8.         consClust:= $chaseInt(KB, D_I, cl, incisions(cl))$;
 9.         cons:= $cons \cup consClust$;
10. if $cons \models Q$ then return Yes;
11. else return No;
```

Figure 5: Computing k-lazy answers for a BCQ Q to a linear Datalog+/– ontology KB.

the general case, for guarded TGDs, it is necessary to look at every possible combination of cuts across all clusters, as it is done in Algorithm lazyConsistent. The procedure used in line 8 of Algorithm lazyConsistentLinear computes the intersection of consequences yielded by each cut to a cluster, which is equivalent to computing the intersection of their groundings. The following definition states which atoms belong to this set.

Definition 13 Given a linear Datalog+/– ontology $KB = (D, \Sigma)$, a set of atoms D_I, and a cluster cl with a corresponding set of k-cuts $incisions(cl)$, an atom a belongs the set $chaseInt(KB, D_I, cl, incisions(cl))$ iff for every $c_j \in incisions(cl)$ there exists atom b_j such that there exists a homomorphism between a and b_j with $b_j \in cl - c_j$.

The above definition aims to capture situations as the one shown in the following example.

Example 12 Suppose that for a given cluster cl we have $incisions(cl) = \{c_1, c_2, c_3, c_4\}$ such that $chase(D_I \cup (cl - c_1), \Sigma) = \{p(a, Y), q(b)\}$, $chase(D_I \cup (cl - c_2), \Sigma) = \{q(a), p(X, Y)\}$, $chase(D_I \cup (cl - c_3), \Sigma) = \{p(a, c), q(X)\}$, and $chase(D_I \cup (cl - c_4), \Sigma) = \{p(Y, Z), q(Y)\}\}$. Then, the output of $chaseInt$ is $\{p(a, c)\}$. Note that if we only consider cuts c_1, c_2, and c_4, then the output is $\{p(a, Y)\}$, whereas if we only consider c_2 and c_4 we get $\{q(a), p(X, Y)\}$. ■

Proposition 8 *Let $KB = (D, \Sigma)$ be a linear Datalog+/– ontology. Algorithm* lazyConsistentLinear(KB) *correctly computes the k-lazy answers for Q to KB in polynomial time in the data complexity.*

5 Query Rewriting for FO-rewritable Fragments of Datalog+/– under the Intersection Semantics

First-order (FO) rewritability of queries over an ontology allows to transform them into FO queries that can be executed over the database alone, i.e., the new queries embed the dependencies and constraints of the ontology. Since an FO query can be translated into an equivalent SQL expression, query answering can be delegated to a traditional relational DBMS, thus exploiting any underlying optimizations. The sublanguage of Datalog+/– with linear TGDs is FO-rewritable [12]. Recently, [13] presents a rewriting algorithm, inspired by resolution in logic programming, which deals with so-called *sticky-join* sets of TGDs, a non-guarded fragment of Datalog+/– defined by a testable condition based on variable marking; this language includes the sets of linear TGDs and allows restricted forms of joins. However, this algorithm corresponds to the standard (non-inconsistency-tolerant) semantics for query answering. More recently, [14] presents a rewriting procedure for inconsistency-tolerant query answering in *DL-Lite*$_A$ ontologies under the intersection semantics; *DL-Lite*$_A$ belongs to the *DL-Lite* family of tractable DLs, which can all be expressed in linear Datalog+/– (with negative constraints).

Under standard query answering in Datalog+/–, a class of TGDs is FO-rewritable iff for every set of TGDs Σ in that class, and every BCQ Q, there exists a FO query Q_Σ such that $(D, \Sigma) \models Q$ iff $D \models Q_\Sigma$, for every database D. In this section, we show that any class of TGDs that is FO-rewritable under standard query answering is FO-rewritable for *consistent query answering under the intersection semantics*. This means that, for every set Σ composed of arbitrary sets Σ_T and Σ_{NC} of TGDs and negative constraints, respectively, and every BCQ Q, there exists an FO query Q_Σ such that $(D, \Sigma) \models_{ICons} Q$ iff $D \models Q_\Sigma$, for every database D, as long as standard query answering is FO-rewritable for the class of of TGDs to which Σ_T belongs. Here, Q_Σ encodes the set of TGDs and the *enforcement* of the negative constraints so that they reflect the intersection semantics for query answering.

In the following, let *FORew* be the set of sublanguages of Datalog+/– that are FO-rewritable and for which the size of the output query is polynomial in the data complexity. Each $\mathcal{S} \in$ *FORew* determines a class of TGDs, denoted as $\mathcal{S}$ TGDs. We sometimes refer to ontologies defined over a language $\mathcal{S} \in$ *FORew* as $\mathcal{S}$ Datalog+/– ontologies.

The rewriting of a BCQ Q relative to a set Σ of $\mathcal{S}$ TGDs, with $\mathcal{S} \in$ *FORew* and negative constraints is accomplished in two steps. First, we analyze how to rewrite the negative constraints into Q in a way that the rewriting enforces the intersection semantics, i.e., $KB \models_{ICons} Q$ iff $D \models Q'$, where Q' is the rewriting of Q obtained by enforcing the constraints in Σ_{NC}. This is done independently of the set of TGDs. Second, both the query and the negative constraints in Σ_{NC} may need to be rewritten relative to the set of TGDs. For this second part, we assume that for each sublanguage $\mathcal{S} \in$ *FORew* there exists an algorithm TGD-rewrite$_\mathcal{S}$ such that its output TGD-rewrite$_\mathcal{S}(IC)$ contains the set of first order formulas that correspond to the rewriting of the set of (negative) constraints *IC*. An example of such an algorithm can be found in [13] for the set of sticky-join TGDs.

5.1 TGD-Free Case

We first focus on the FO rewriting of a BCQ Q relative to an ontology without TGDs Σ_{NC} by enforcing the negative constraints in Σ_{NC}, i.e., on obtaining $(D, \Sigma_{NC}) \models_{ICons} Q$ iff $D \models Q'$, where Q' is the enforcement of Σ_{NC} in Q. Intuitively, in order to compute the intersection semantics, we seek to establish a correspondence between the minimization of negative constraints in query answering process and the minimization inherently encoded in culprits. For sake of clarity of presentation and without loss of generality, given the result in Lemma 1, in this section we will consider *normalized* (negative) constraints, as defined in Definition 2.

Example 13 *Consider the Datalog+/– ontology KB from Example 1, KB is a multi-linear Datalog+/– ontology [3] (cf. Section 5.3). Given a BCQ Q = supervises(tom, john), the normalized instances of constraint υ_1 within $\mathcal{N}(\Sigma_{NC}\, Q)$, are the following:*

$\upsilon_{1,1}$: *supervises*(*tom*, *john*) $\wedge$ *manager*(*john*, D)$\rightarrow \bot$,

$\upsilon_{1,2}$: *supervises*(*john*, *tom*) $\wedge$ *manager*(*tom*, D) $\rightarrow \bot$,

$\upsilon_{1,3}$: *supervises*(*tom*, *tom*) $\wedge$ *manager*(*tom*, D) $\rightarrow \bot$, *and*

$\upsilon_{1,4}$: *supervises*(*john*, *john*) $\wedge$ *manager*(*john*, D) $\rightarrow \bot$. ■

```
Algorithm Enforcement(BCQ $Q = \exists G$, normalized negative constraints IC)
Here, $G$ is a quantifier-free formula, and $\exists G$ is the existential closure of $G$.
 1. $F := G$;
 2. for every $\upsilon \in IC$ do
 3.   for every $C \subseteq Q$, $C \neq \emptyset$, that unifies with $B \subseteq body(\upsilon)$ via mgu $\gamma_{C,B}$ do
 4.     if for no $\upsilon' \in IC$, $body(\upsilon')$ maps isomorphically to $B' \subset body(\upsilon)$ then
 5.       $F := F \wedge \neg\exists_{\overline{G}}\,((\bigwedge_{X \in var(C)} X = \gamma_{C,B}(X)) \wedge \gamma_{C,B}(body(\upsilon)))$
            (where $\exists_{\overline{G}}\, R$ is the existential closure of $R$
            relative to all variables in $R$ that are not in $G$);
 6. return $\exists F$.
```

Figure 6: Computing the enforcement of a normalized set of NCs *IC* relative to a BCQ Q.

The next step in the FO rewriting of a BCQ Q by enforcing the negative constraints is to identify the set of normalized negative constraints that must be *enforced* in Q, i.e., the normalized negative constraints that must be satisfied so that only the consistent answers under the intersection semantics are entailed from D.

As usual, two atoms a and b *unify* iff there exists a substitution γ (called a *unifier* for a and b) such that $\gamma(a) = \gamma(b)$. A *most general unifier (mgu)* is a unifier for a and b, denoted $\gamma_{a,b}$, such that for any other unifier γ for a and b, there exists a substitution γ' such that $\gamma = \gamma' \circ \gamma_{a,b}$. Note that mgus are unique up to variable renaming.

Definition 14 Let Σ_{NC} be a set of negative constraints, $\upsilon \in \mathcal{N}(\Sigma_{NC}, Q)$, and Q be a BCQ. Then, υ *needs to be enforced* in Q iff there exists $C \subseteq Q$, $C \neq \emptyset$, such that C unifies with $B \subseteq body(\upsilon)$ via some mgu $\gamma_{C,B}$, and there exists no $\upsilon' \in \mathcal{N}(\Sigma_{NC}, Q)$ such that $body(\upsilon')$ maps isomorphically to $B' \subset body(\upsilon)$.

Example 14 Consider the Datalog+/– ontology KB from Example 13, BCQ $Q = supervises(tom, john)$ and the normalization of constraint υ_1 showed in Example 13. Though we have not shown the complete set $\mathcal{N}(\Sigma_{NC}, Q)$, we can easily see that the only normalized instances of constraint υ_1 that must be enforced are $\upsilon_{1,1}$ and $\upsilon_{1,3}$. If we take, $\upsilon_{1,1}$, for instance, we have that the set $C_1 = \{supervises(tom, john)\}$ unifies with $B_1 = \{supervises(tom, john)\}$ via the mgu $\gamma_{C_1,B_1} = \{\}$. ∎

Example 15 Consider $\mathcal{N}(\Sigma_{NC}, Q)$ from Example 2, and the BCQ $Q = \exists X q(X)$. Then, neither υ'_1 nor υ'_2 need to be enforced in Q, while υ'_3 needs to be enforced in Q. ∎

Algorithm Enforcement (see Fig. 6) performs the rewriting of a query Q by embedding all negative constraints that must be enforced. The following example shows how the algorithm works for the ontology KB from Example 1.

Example 16 Coming back to Example 14, the enforcement of Σ_{NC} in query $Q = supervises(tom, john) \wedge works\_in(tom, d_1)$, returned by Algorithm Enforcement in Fig. 6, is given by:
$(Q \wedge \forall D, Y \neg\big(supervises(tom, john) \wedge \wedge works\_in(tom, d_1) \wedge manager(john, D)\big) \wedge$
$\neg\big(supervises(tom, john) \wedge works\_in(tom, d_1) \wedge directs(Y, d_1)\big) \wedge$
$\neg\big(supervises(tom, john) \wedge works\_in(tom, D) \wedge directs(Y, D) \wedge D \neq d_1\big)\big)$.

Ignoring the TGDs Σ_T, we have $(D, \Sigma_{NC}) \not\models_{ICons} Q$, since $Q \notin D_I = \{manager(tom, d_1)\}$; D_I is the intersection of all data repairs in (D, Σ_{NC}). As expected, we see that $D \not\models \mathsf{Enforcement}(Q, \mathcal{N}(\Sigma_{NC}, Q))$. ∎

We now establish the correctness of Algorithm $\mathsf{Enforcement}$. The following proposition is used in the proof of the main correctness result in Theorem 8 below. It states that to answer a query under the intersection semantics, it is only necessary to look at the set of normalized negative constraints that need to be enforced in Q.

Proposition 9 *Let $KB = (D, \Sigma_{NC})$ be a Datalog+/– ontology without TGDs, and Q be a BCQ. Let $\Sigma_Q \subseteq \mathcal{N}(\Sigma_{NC}, Q)$ be the set of constraints that must be enforced in Q. Then, $KB \models_{ICons} Q$ iff $(D, \Sigma_Q) \models_{ICons} Q$.*

Theorem 8 shows the correctness of Algorithm $\mathsf{Enforcement}$. It is important to note that here we are now only assuming a Datalog+/– ontology of the form $KB = (D, \Sigma)$ where Σ contains only negative constraints, i.e., no rewriting relative to TGDs is needed. Though the above results are valid for general Datalog+/– ontologies, Theorem 8 only holds for Datalog+/– ontologies that do not have TGDs.

Theorem 8 *Let $KB = (D, \Sigma_{NC})$ be a Datalog+/– ontology without TGDs, and Q be a BCQ. Then, $KB \models_{ICons} Q$ iff $D \models \mathsf{Enforcement}(Q, \mathcal{N}(\Sigma_{NC}, Q))$.*

5.2 General Case

We now concentrate on the general problem of rewriting a BCQ Q relative to a set of negative constraints and $\mathcal{S}$ TGDs $\Sigma_{NC} \cup \Sigma_T$, with $\mathcal{S} \in \mathit{FORew}$. To this end, we have to generalize the enforcement of Σ_{NC} in Q described in the previous section. The following result is used to show that to enforce Σ_{NC} in Q, it is possible to rewrite the body of the negative constraints first and then to enforce the new set of negative constraints (containing all possible rewritings of the negative constraints relative to Σ_T) in Q. It follows immediately from the soundness and completeness assumed of Algorithm $\mathsf{TGD\text{-}rewrite}_{\mathcal{S}}$.

Lemma 3 *Let $KB = (D, \Sigma)$ with $\Sigma = \Sigma_{NC} \cup \Sigma_T$ be a Datalog+/– ontology, Σ_{Rew} be the set of all negative constraints $F \rightarrow \bot$ such that $F \in \mathsf{TGD\text{-}rewrite}_{\mathcal{S}}(\mathit{body}(\upsilon), \Sigma_T)$ for some $\upsilon \in \Sigma_{NC}$. Then, $\mathit{culprits}(KB) = \mathit{culprits}(KB')$, where $KB' = (D, \Sigma_{Rew})$.*

As an immediate consequence, query answering in $\mathcal{S}$ Datalog+/– under the intersection semantics is invariant to rewriting the negative constraints relative to the TGDs. This result follows immediately from Lemma 3 and Theorem 3.

Proposition 10 *Let $KB = (D, \Sigma)$, $\Sigma = \Sigma_{NC} \cup \Sigma_T$, be an $\mathcal{S}$ Datalog+/– ontology, with $\mathcal{S} \in \mathit{FORew}$. Let Σ_{Rew} be the set of all negative constraints $F \rightarrow \bot$ such that $F \in \mathsf{TGD\text{-}rewrite}_{\mathcal{S}}(\mathit{body}(\upsilon), \Sigma_T)$ for some $\upsilon \in \Sigma_{NC}$. Then, $KB \models_{ICons} Q$ iff $(D, \Sigma_{Rew} \cup \Sigma_T) \models_{ICons} Q$.*

The following example illustrates the rewriting of the set of negative constraints in the running example relative to a corresponding set of TGDs.

Example 17 Consider $\upsilon_1 = \mathit{supervises}(X, Y) \wedge \mathit{manager}(Y) \rightarrow \bot$ from Σ_{NC} in Example 1. Then, the rewriting of $\mathit{body}(\upsilon_1)$ relative to Σ_T is given by:

$$\begin{aligned}\mathsf{TGD\text{-}rewrite}_{\mathcal{S}}(\mathit{body}(\upsilon_1), \Sigma_T) \;=\; & \{\mathit{rw}_1 : \mathit{supervises}(X, Y) \wedge \mathit{manager}(Y, D),\\ & \mathit{rw}_2 : \mathit{supervises}(X, Y) \wedge \mathit{works\_in}(Y, D) \wedge \\ & \mathit{directs}(Y, D)\}\,.\end{aligned}$$

Similarly, we have that $\mathsf{TGD\text{-}rewrite}_{\mathcal{S}}(\mathit{body}(\upsilon_2), \Sigma_T) = \{\mathit{body}(\upsilon_2)\}$, and $\mathsf{TGD\text{-}rewrite}_{\mathcal{S}}(\mathit{body}(\upsilon_3), \Sigma_T) = \{\mathit{body}(\upsilon_3)\}$. (Recall that υ_3 is treated as the negative constraint $\mathit{directs}(X, D) \wedge \mathit{directs}(X, D') \wedge D \neq D' \rightarrow \bot$.) Hence, $\Sigma_{Rew} = \{\mathit{rw}_1 \rightarrow \bot, \mathit{rw}_2 \rightarrow \bot, \upsilon_2, \upsilon_3\}$. ■

```
Algorithm rewriteICons(BCQ $Q$, set of negative constraints and $\mathcal{S}$ TGDs $\Sigma_{NC} \cup \Sigma_T$)
 1. $\Sigma_{Rew} := \{F \to \bot \mid F \in \mathsf{TGD\text{-}rewrite}_{\mathcal{S}}(body(\upsilon), \Sigma_T)$ for some $\upsilon \in \Sigma_{NC}\}$;
 2. $Q_{rw} := \mathsf{TGD\text{-}rewrite}_{\mathcal{S}}(Q, \Sigma_T)$;
 3. $out := \emptyset$;
 4. for each $Q \in Q_{rw}$ do
 5.     $out := out \cup \mathsf{Enforcement}(Q, \mathcal{N}(\Sigma_{Rew}, Q))$;
 6. return $out$.
```

Figure 7: Rewriting a BCQ Q relative to a set of negative constraints and FO-rewritable $\mathcal{S}$ TGDs Σ under the intersection semantics; see Fig. 6 for Algorithm Enforcement.

Algorithm rewriteICons in Figure 7 computes the rewriting of a BCQ Q relative to a set of negative constraints and $\mathcal{S}$ TGDs $\Sigma = \Sigma_{NC} \cup \Sigma_T$. The algorithm works as follows. First, the rewriting of the bodies of the negative constraints in Σ_{NC} are computed relative to Σ_T, using algorithm $\mathsf{TGD\text{-}rewrite}_{\mathcal{S}}$. Then, similarly, the rewriting of Q is computed relative to Σ_T. Finally, for each query in the rewriting of Q, the algorithm enforces the normalization of the rewritten set of negative constraints. The following example illustrates how Algorithm rewriteICons works.

Example 18 Consider again the BCQ $Q = \exists D\, manager(john, D)$ to the Datalog+/– ontology $\Sigma = \Sigma_{NC} \cup \Sigma_T$ from Example 1. First, algorithm rewriteICons computes Σ_{Rew}, the rewriting of Σ_{NC} relative to Σ_T, as shown in Example 17. Second, Q is rewritten relative to Σ_T, obtaining $Q_{rw} = \{directs(john, D) \wedge works\_in(john, D), manager(john, D)\}$.
We have that for $Q' = directs(john, D) \wedge works\_in(john, D), \mathcal{N}(\Sigma_{Rew}, Q') = \{$
$rw_{1,1} : supervises(X, john) \wedge manager(john, D) \wedge X \neq john \to \bot$,
$rw_{1,2} : supervises(john, Y) \wedge manager(Y, D) \wedge Y \neq john \to \bot$,
$rw_{1,3} : supervises(john, john) \wedge manager(john, D) \to \bot$,
$rew_{2,1} : supervises(john, Y) \wedge directs(Y, D) \wedge works\_in(Y, D) \wedge Y \neq john \to \bot$,
$rew_{2,2} : supervises(X, john) \wedge directs(john, D) \wedge works\_in(john, D) \wedge X \neq john \to \bot$,
$rew_{2,3} : supervises(john, john) \wedge directs(john, D) \wedge works\_in(john, D) \to \bot$,
$\upsilon_{2,1} : supervises(john, Y) \wedge directs(Y, D) \wedge works\_in(john, D) \wedge Y \neq john \to \bot$,
$\upsilon_{2,2} : supervises(X, john) \wedge directs(john, D) \wedge works\_in(X, D) \wedge X \neq john \to \bot$,
$\upsilon_{2,3} : supervises(john, john) \wedge directs(john, D) \wedge works\_in(john, D) \to \bot$,
$\upsilon_{3,1} : directs(john, D) \wedge directs(john, D') \wedge D \neq D' \to \bot\}$.

Clearly, none of the normalized instances of rw_1 need to be enforced since there is no $C \subseteq Q'$, with $C \neq \emptyset$, that unifies with $B \subseteq body(rw_{1,j})$ with $1 \leqslant j \leqslant 3$. However, all the other constraints in $\mathcal{N}(\Sigma_{Rew}, Q')$ need to be enforced. We now show how the different constraints are enforced following the *for* loop in line 2 in Algorithm Enforcement. For ease of presentation, and without loss of generality, we only consider the largest sets $C \subseteq Q'$ that unify with part of the body of some constraint.

Consider constraint $rw_{2,1}$: we have $C = directs(john, D) \wedge works\_in(john, D)$, $B = directs(Y, D) \wedge works\_in(Y, D)$, and $\gamma_{C,B} = \{[Y/john]\}$. We enforce the constraint by adding the following formula:

$$\begin{aligned}\neg\big(\; & Y = john \wedge supervises(john, john) \wedge directs(john, D) \wedge \\ & works\_in(john, D) \wedge Y \neq john\big)\end{aligned} \tag{2}$$

For constraint $rw_{2,2}$, we have $C = directs(john, D) \wedge works\_in(john, D)$ and $B = directs(john, D) \wedge$

$\mathit{works\_in}(\mathit{john}, D)$. This constraint is enforced by adding the following formula:

$$\neg\exists X \big(\mathit{supervises}(X, \mathit{john}) \wedge \mathit{directs}(\mathit{john}, D) \wedge \mathit{works\_in}(\mathit{john}, D) \wedge X \neq \mathit{john}\big) \tag{3}$$

Continuing with constraint $\mathit{rw}_{2,3}$, we have $C = \mathit{directs}(\mathit{john}, D) \wedge \mathit{works\_in}(\mathit{john}, D)$, and $B = \mathit{directs}(\mathit{john}, D) \wedge \mathit{works\_in}(\mathit{john}, D)$. The following formula enforces this constraint:

$$\neg\big(\mathit{supervises}(\mathit{john}, \mathit{john}) \wedge \mathit{directs}(\mathit{john}, D) \wedge \mathit{works\_in}(\mathit{john}, D)\big) \tag{4}$$

For constraint $\upsilon_{2,1}$, $C = \mathit{directs}(\mathit{john}, D) \wedge \mathit{works\_in}(\mathit{john}, D)$, $B = \mathit{directs}(Y, D) \wedge \mathit{works\_in}(\mathit{john}, D)$, and $\gamma_{C,B} = \{[Y/\mathit{john}]\}$. We enforce the constraint by adding the following formula:

$$\neg\big(Y = \mathit{john} \wedge \mathit{supervises}(\mathit{john}, \mathit{john}) \wedge \mathit{directs}(\mathit{john}, D) \wedge \mathit{works\_in}(\mathit{john}, D) \wedge Y \neq \mathit{john}\big) \tag{5}$$

Now consider constraint $\upsilon_{2,2}$: we have $C = \mathit{directs}(\mathit{john}, D) \wedge \mathit{works\_in}(\mathit{john}, D)$, $B = \mathit{directs}(\mathit{john}, D) \wedge \mathit{works\_in}(X, D)$. We enforce it by adding the following:

$$\neg\exists X \big(\mathit{supervises}(X, \mathit{john}) \wedge \mathit{directs}(\mathit{john}, D) \wedge \mathit{works\_in}(X, D) \wedge X \neq \mathit{john}\big) \tag{6}$$

Next, for constraint $\upsilon_{2,3}$ we have $C = \mathit{directs}(\mathit{john}, D) \wedge \mathit{works\_in}(\mathit{john}, D)$, $B = \mathit{directs}(\mathit{john}, D) \wedge \mathit{works\_in}(\mathit{john}, D)$. This is enforced by the formula:

$$\neg\big(\mathit{supervises}(\mathit{john}, \mathit{john}) \wedge \mathit{directs}(\mathit{john}, D) \wedge \mathit{works\_in}(\mathit{john}, D)\big) \tag{7}$$

Finally, consider constraint $\upsilon_{3,1}$: we have $C = \mathit{directs}(\mathit{john}, D)$, $B = \mathit{directs}(\mathit{john}, D)$. We enforce this constraint by adding the following formula:

$$\neg\exists D' \big(\mathit{directs}(\mathit{john}, D) \wedge \mathit{directs}(\mathit{john}, D') \wedge D \neq D'\big) \tag{8}$$

The enforcement of $\mathcal{N}(\Sigma_{Rew}, Q')$ to Q' is

$$\begin{array}{rl} Q_1 = & \mathit{directs}(\mathit{john}, D) \wedge \mathit{works\_in}(\mathit{john}, D) \wedge \\ \neg\exists X & \big(\mathit{supervises}(X, \mathit{john}) \wedge \mathit{directs}(\mathit{john}, D) \wedge \mathit{works\_in}(\mathit{john}, D) \wedge X \neq \mathit{john}\big) \wedge \\ \neg & \big(\mathit{supervises}(\mathit{john}, \mathit{john}) \wedge \mathit{directs}(\mathit{john}, D) \wedge \mathit{works\_in}(\mathit{john}, D)\big) \wedge \\ \neg\exists X & \big(\mathit{supervises}(X, \mathit{john}) \wedge \mathit{directs}(\mathit{john}, D) \wedge \mathit{works\_in}(X, D) \wedge X \neq \mathit{john}\big) \wedge \\ \neg\exists D' & \big(\mathit{directs}(\mathit{john}, D) \wedge \mathit{directs}(\mathit{john}, D') \wedge D \neq D'\big) \end{array}$$

Note that formulas (2) and (5) are logically equivalent to *true*; for this reason, we do not include them in the final enforcement. The enforcement of $\mathcal{N}(\Sigma_{Rew}, Q'')$ to Q'', where $Q'' = \mathit{manager}(\mathit{john}, D)$ is $Q_2 =$

$$\mathit{manager}(\mathit{john}, D) \wedge \neg\exists X \big(\mathit{supervises}(X, \mathit{john}) \wedge \mathit{manager}(\mathit{john}, D) \wedge (X \neq \mathit{john}\big) \wedge$$

$$\big(\mathit{supervises}(\mathit{john}, \mathit{john}) \wedge \mathit{manager}(\mathit{john}, D)\big)$$

Finally, the output of algorithm $\mathsf{rewriteICons}$ is $\mathit{out} = \{Q_1, Q_2\}$ ($\equiv Q_1 \vee Q_2$). Both Q_1 and Q_2 are false on the database D from Example 1, and so the consistent answer for Q under the intersection semantics is *false*, which is correct since $D_I = \bigcap_{D' \in \mathit{DRep}(KB)} D' = \{\mathit{works\_in}(\mathit{john}, d_1), \mathit{works\_in}(\mathit{tom}, d_1)\}$, and thus $KB \not\models_{\mathit{ICons}} Q$. ■

The following theorem shows the correctness of Algorithm $\mathsf{rewriteICons}$.

Theorem 9 *Let $KB = (D, \Sigma)$ with $\Sigma = \Sigma_{NC} \cup \Sigma_T$ be an $\mathcal{S}$ Datalog+/– ontology with $\mathcal{S} \in \mathit{FORew}$, and Q be a BCQ. Then, $KB \models_{\mathit{ICons}} Q$ iff $D \models \bigvee_{F \in \mathsf{rewriteICons}(Q,\Sigma)} F$.*

5.3 Concrete Classes of FO Rewritable Sets of TGDs

The most basic class of TGDs that are FO rewritable under the intersection semantics is that of *linear* TGDs (cf. Section 2). The main weakness of linear TGDs is that they do not allow joins in their bodies. *Multi-linear* TGDs relax this restriction: each atom in the body a multi-linear TGD is a guard. Ontology KB in our running Example 1 is multi-linear, this is because TGD σ_3 is a multi-linear TGD. Nevertheless, the joins that can be represented are very restrictive for multi-linear TGDs. In our running example, a multi-linear TGD cannot express that an employee that is also a manager supervises some other employee: $\mathit{emp}(X) \wedge \mathit{manager}(X, D) \rightarrow \exists Y \mathit{supervises}(X, Y)$ is not multi-linear. The following, more expressive class of TGDs, allows to represent this kind of joins without losing FO rewritability.

The class of *sticky* sets of TGDs is defined in [15] by by a syntactic criterion that is easily testable, which is as follows. For every database D, assume that during the chase of D regarding a set Σ of TGDs, we apply a TGD $\sigma \in \Sigma$ which has a variable V appearing more than once in its body. Assume also that V maps (via a homomorphism) on the symbol z, and that by virtue of this application the atom a is generated. In this case, for each atom $b \in \mathit{body}(\sigma)$ we say that a is derived from b. Then, z appears in a, and in all atoms resulting from some chase derivation sequence starting from a, "sticking" to them (hence the name "sticky" sets of TGDs). The definition of sticky sets of TGDs is based heavily on a variable-marking procedure called *SMarking*. This procedure accepts as input a set of TGDs Σ, and marks the variables that occur in the body of the TGDs of Σ. Formally, *SMarking*(Σ) works as follows. First, we do the following: for each TGD $\sigma \in \Sigma$, and for each variable V in $\mathit{body}(\sigma)$, if there exists an atom a in $\mathit{head}(\sigma)$ such that V does not appear in a, then we mark each occurrence of V in $\mathit{body}(\sigma)$. Then, we apply exhaustively (i.e., until a fixpoint is reached) the propagation step: if a marked variable in $\mathit{body}(\sigma)$ appears at position ϕ, then for every TGD $\sigma' \in \Sigma$ (including the case $\sigma = \sigma'$), we mark each occurrence of the variables in $\mathit{body}(\sigma')$ that appear in $\mathit{head}(\sigma')$ at the same position ϕ.

For each pair of TGDs $\langle \sigma, \sigma' \rangle \in \Sigma \times \Sigma$ (including the case $\sigma = \sigma'$), if a universally quantified variable V occurs in $\mathit{head}(\sigma)$ at positions $\phi_1, \ldots, \phi_m$ for $m \geqslant 1$, and there exists an atom $a \in \mathit{body}(\sigma')$ such that at each position $\phi_1, \ldots, \phi_m$ a marked variable occurs, then we mark each occurrence of V in $\mathit{body}(\sigma)$. The formal definition of sticky sets of TGDs follows.

Definition 15 *A set Σ of TGDs is sticky if there is no TGD $\sigma \in$ SMarking(Σ) such that a marked variable occurs in $\mathit{body}(\sigma)$ more than once.*

The class os sticky set set of TGDs considered together with negative constraints and functional dependencies (special case of EGDs) is strictly more expressive than *DL-Lite*$_\mathcal{A}$, *DL-Lite*$_\mathcal{R}$, and *DL-Lite*$_\mathcal{F}$ [15]. Joins are restricted so the interaction with the TGDs does not make query answering undecidable; however, the set is still quite expressive, for instance many realistic sets of multi-valued dependencies are sticky.

Example 19 *Consider Datalog+/– ontology KB from Example 1 and let:*

$$\sigma_4 : \mathit{temp\_emp}(X) \wedge \mathit{works\_in}(Y, D) \wedge \mathit{directs}(X, D) \rightarrow \mathit{intern}(Y, X, D)$$

belong to Σ; now, KB is a sticky Datalog+/– ontology. TGD σ_4 expresses that a temporal employee working in a department is an intern that responds to whoever directs that department. ■

Finally, an even more expressive FO rewritable class of sets of TGDs are those so-called Sticky-join [16]. Sticky-join sets of TGDs extend both sticky sets of TGDs and linear TGDs. Similarly to sticky sets of

TGDs, sticky-join sets are defined by a testable condition based on variable marking. However, the variable-marking for this class is more sophisticated, i.e., checking whether a set of TGDs is sticky-join, is PSPACE-hard [16]. Notice that the identification problem under (multi-)linear TGDs and sticky sets of TGDs is feasible in PTIME [15].

6 Related Work

In the database community, the field of *database repairing* and *consistent query answering* (CQA) has gained much attention since the work of [17], which provided a model-theoretic construct of a database *repair*. The most widely accepted semantics for querying a possibly inconsistent database is that of *consistent answers*, which yields the set of tuples (atoms) that appear in the answer to the query over *every* possible repair. CQA enforces consistency at query time as an alternative to enforcing it at the instance level, as conventional data cleaning techniques do. This allows to focus on a smaller portion of the database for which repairs can be computed more easily. Furthermore, techniques have been developed, so that it is not necessary to materialize every possible repair. The work of [18] addresses the basic concepts and results of the area of CQA.

More recently, several works have focused on inconsistency handling for several classes of DLs, adapting and specializing general techniques previously considered for traditional logics [19, 20, 21, 22, 23]. In [24], a comparison between four different approaches to handling inconsistency in DL-based ontologies is presented: *consistent ontology evolution*, *repairing inconsistencies*, *reasoning with inconsistent ontologies*, and *ontology versioning*. In [25] a survey (up to 2007) of existing approaches for handling inconsistencies in DL-based ontologies is presented, the authors analyze different works from the literature and their corresponding usability on practical problems. The work in [9] studies the adaptation of CQA for *DL-Lite* ontologies. In that work, the *intersection* semantics is presented as a sound approximation of consistent answers, which for the *DL-Lite* family is easier to compute, as well as the *closed ABox* version for both of them, which considers the closure of the set of assertional axioms (ABox, or extensional database) by the terminological axioms (TBox, or intensional database). The data and combined complexity of the semantics were studied in [26] for a wider spectrum of DLs. The tractability results obtained in this paper for consistent answers and intersection semantics for linear Datalog+/– ontologies generalize the previous work for the *DL-Lite* family. In [26], intractability results for query answering were found for $\mathcal{EL}_\perp$ under the intersection semantics, and a non-recursive segment of that language was proved to be computable in polynomial time. These languages are incomparable with linear Datalog+/– in the sense that neither subsumes the other. More recently, in [27], another family of approximations to CQA are proposed, also for the *DL-Lite* family. The k-support semantics allows to (soundly) approximate the set of queries entailed under the CQA semantics, based on k subsets of the database that consistently entail q; on the other hand, the k-defeater semantics approximates complete approximations seeking sets that contradict the supporters for q. Both semantics are FO-rewritable for any ontological language for which standard CQ answering is FO-rewritable as well, and can be used in conjunction to over- and under-approximate consistent answers. In contraposition to these approaches, the lazy semantics is novel in the sense that it does not seek an approximation to consistent answers, aiming instead to produce answers that do not violate the set of negative constraints using the value k to restore consistency with the smallest possible changes.

Along the line of shifting focus away from consistent answers, the work in [28] proposes, instead of identifying the consistent answers to a given query, to warn the user about the presence of suspect elements in the query result. This is similar to the idea in artificial intelligence to associate with a conclusion the information that it is paraconsistent or defeasible, i.e., the opposite conclusion could be supported as well.

Moreover, note that a set of answers that does not contain any suspicious ones may still be an inconsistent set of information since functional dependencies with respect to attributes not involved in the query may be violated in this set, a CQA approach would return such answers but no extra information about them. This approach differs from CQA in the sense that it provides more information about the answers and which can be specially valuable in the case where there are no (or very few) consistent answers, and it is, in principle, also much less costly since it avoids the complexity associated with searching for minimal repairs.

Arenas et al. [17] propose a method to compute consistent query answers based on query rewriting that applies to FO queries without disjunction or quantification, and databases with only binary universal integrity constraints without considering TGDs. The rewriting applies to and produces FO queries. When a literal in the query can be resolved with an integrity constraint, the resolvent forms a residue. All such residues are then conjoined with the literal to form its expanded version. If a literal that has been expanded appears in a residue, the residue has to be further expanded until no more changes occur. A rewriting method for a larger subset of conjunctive queries but that is limited to primary key FDs (a special case of EGDs) was proposed in [29]. The work [18] summarizes some of the main results of the area of CQA. Most recently, in [14], the authors explore FO-rewritability of *DL-Lite* ontologies under the intersection and closed ABox semantics; both semantics were first introduced in [9]. The rewritability results obtained in this paper for consistent answers under the intersection semantics for linear Datalog+/– ontologies significantly generalize the previous work for *DL-Lite*$_A$. In [30], the author studies and formulates general conditions to prove that a FO-rewriting exists or not for *DL-Lite*$_{core}$ for consistent instance checking. In the paper, the key to show existence of a rewriting lies in the identification of specific types of conflict chains (the way in which conflicts interact with each other relative to the query) that can appear in a given ontology. Following up that line of work, the recent work in [31], provides a fine-grained complexity analysis that aims to understand what makes consistent answers so hard to compute even for simple ontologies consisting of class subsumption and class disjointness. The author identifies that the number of quantified variables in the conjunctive query has a dramatic effect on determining the complexity of query answering under the consistent answers semantics. The results in the paper show that (1) consistent query answering is always first-order rewritable for conjunctive queries of at most two quantified variables, (2) the problem has polynomial data complexity when there are two quantified variables, and (3) a necessary and sufficient condition for first-order rewritability is identified for queries with at most two quantified variables. Furthermore, a novel inconsistency tolerant semantics that is a sound approximation to consistent answers is presented. This semantics which is define as the answers that can be derived in the intersection the *closure* of the repairs, is a finer approximation than the intersection semantics and query answering for *DL-Lite* ontologies is first-order rewritable for arbitrary conjunctive queries. Finally, the work in [32] develops a query rewriting algorithm for *DL-Lite*$_A$ with identification (key constraints) and denial assertions (negative constraints) under the intersection semantics. The results in that paper apply to a less expressive language than the one studied here, therefore the technique developed here are no simple generalizations of the one from [32].

A very important area of research related to inconsistency management is that of *belief change* theory [33, 34]. Belief change theory focuses on modeling the changes to belief sets (sets of formulas closed under consequence) and belief bases (sets of formulas not necessarily closed under consequence). In any practical database application, only belief bases are relevant, and hence, in this paper, we borrow the notions of incision functions and kernels from [11]. Belief contraction is the process of removing beliefs from a knowledge base (and/or the input). In [11], a formal model of contraction of a sentence α is presented based on the selection among elements that contribute to make the knowledge base KB imply α. The minimal subsets of KB that imply α are called α-kernels. Incision functions are defined to formally capture the elements of each α-kernel that are to be discarded. The approach of CQA is then, in part, similar to contraction by *falsum*, i.e., remove from the knowledge base sentences so $\bot$ (inconsistency) is not implied any longer. For this reason, we make use of *culprits* that correspond exactly to $\bot$-kernels in Datalog+/– ontologies. The main difference between belief contraction and CQA is, however, that CQA does not seek to get rid of the inconsistent information but to provide an inconsistency-tolerant method of reasoning, or in our case, of query answering. No sentences are removed from the knowledge base; instead, a model of the consistent, certain, information is created and used in the process of answering queries despite the contradictions that may exist. Belief revision and contraction techniques have also been developed for DLs-based ontologies. In [35], the authors focus on the problem of ontology evolution, defined as the timely adaptation of an ontology to changes and the consistent management of these changes. A generalization of the AGM postulates is proposed for revision to DLs. Furthermore, two revision operators are described, one for which whenever an concept assertion or a *general concept inclusion* is involved in a conflict, the axioms are weakened, by either removing the assertion or removing the individuals in the that cause the problems in the concept inclusion. Finally, an alternative revision operator to refine the weakening: the weakening of a concept assertion corresponds to the weakening of its atomic concepts, i.e., dropping some individuals which are related to the individual that is in conflict; this refinement allows, in some cases, to maintain more information in the knowledge base through out the revision process.

Another important and also well-studied area is that of paraconsistent reasoning. Paraconsistent methods allow contradictory information to be derived and introduced without trivialization, i.e., the principle of explosion does not hold. Paraconsistent logics were introduced in the 60s, and logics of inconsistency were later developed [36, 37, 38, 39, 40]. The work of [38] introduced a four valued logic that was used for handling inconsistency in logic programming [41] and extended to the case of bilattices [42]. More recently, [43] proposes a 4-valued logic for P-Datalog (logic programs with negations in the bodies, but not in the heads, of the rules) for querying databases obtained from multiple sources that can contain inconsistent information. The information stored in the knowledge base is marked as true or inconsistent (this is done through a merging process). The framework is based on the paraconsistent (3-valued) logic **LFI1** [44] and allows to infer new facts and the facts in the answers are classified as true, false, inconsistent, or unknown, according to the proposed well-founded semantics. The main difference with Belnap's 4-valued logic is that this work considers a total order on the logic's values instead of partial orders. The work in [45] defines a 3-valued paraconsistent semantics for the well-known description logic $\mathcal{SHIQ}$ [46]. The paraconsistent semantics of $\mathcal{SHIQ}$ is based on Kleene's 3-valued logics [47, 48]. The main difference of the approaches based on these logics is that they identify inconsistency with the lack of knowledge, i.e, the value *inconsistent* characterizes undefined information, whereas in other approaches, such as the one in [43], it stands for overdefined (inconsistent) information (actually, a fourth value is added in [43] to characterize undefined information). The authors in [45] claim that the choice of 3-valued logic over 4-valued ones makes more sense in a multi-agent setting and the Semantic Web, also allows to simplify the underlying formalism. In fact, a faithful translation of the formalism into a 2-valued description logic is provided, which allows to adapt and use existing tools and reasoners $\mathcal{SHIQ}$ [46]. A four-valued semantics to $\mathcal{SROIQ}$ (a DL that underlies OWL 2) is proposed, as well its impact for several tractable description logics around OWL 2. The approach has the advantage of being reducible to reasoning under classical semantics and the transformation required is linear in the size of the knowledge base. Interestingly, the authors show that the four-valued semantics can be adapted for DL-Lite without losing the property mentioned above, preserving the tractability of reasoning in such DL. The main difference between these paraconsistent approaches and the work presented here is in the expressiveness of the underlying language; we focus on a more lightweight class of ontologies which allows, in principle, for more efficient query evaluation. Furthermore, CQA is only interested in *certain* answers, i.e., data that is known to be true irrespectively of how the conflicts are resolved in the knowledge base. In this sense, the work in the area of paraconsistent reasoning is more general than ours since they further refine the classifications of answers that are not guaranteed to be true under every possible model/repair.

Within Artificial Intelligence, many other efforts trying to deal with potentially inconsistent information have been developed in the last four decades. Frameworks such as default logic [49] can be used to represent a database DB with integrity constraints IC as a default logic theory where the background theory consists of the IC and the facts in D constitutes the defaults rules, i.e., a fact in D is assumed to be true if it can be assumed to be true. Maximal consistent subsets [50], inheritance networks [51], and others were used to generate multiple plausible consistent scenarios (or "extensions"), and methods to draw inferences were developed that examined truth in all (or some) extensions. [52] extended annotated logics of inconsistency developed in [41] to handle a full first order case, while [53] developed similar extensions to handle inheritance networks. Syntactic approaches have also been adopted such as the one discussed in [54]; finally, argumentation methods [55, 56, 57] have been used for handling uncertainty and inconsistency by means of reasoning about how certain contradictory arguments defeat each other. Other important contributions include [58]. There is also increasingly interest on measuring inconsistency. The work in [59] proposes a semantics of *weighted maximal consistent subsets* as a way of reasoning in inconsistent systems. Though inconsistent, the system still contains semantic information that provides evidence of the truth of a sentence. A weight on maximal consistent subsets is then computed as combination of that evidence, the quantity of semantic information of the system, and the set of models of its maximal consistent subsets. In [60], the authors review the measures of information and contradiction in the literature and to study the potential practical use of such theories in inconsistency tolerant reasoning systems. In [61], a measure to quantify the inconsistency of a full first order knowledge base is presented with applications to analyzing ontological and temporal knowledge. The measure is based on the models of tolerant logic, which is a first order four-valued logic; these models are based on bistructure, i.e., a pair of classical interpretations: One interpretations for the satisfaction of atoms, and the other one for the satisfaction of negative literals. Given a bistructure, a simple measure of inconsistency gives the proportion of the tuples in the bistructure that are in conflict. The amount of conflict in a bistructure is the number of tuples that are both true and false. This is normalized by the total number of tuples that are possible in the interpretations, which is a function of the size of the domain. This measure of inconsistency is then generalized to sets of bistructures and models. The representation of degree of inconsistency in the form of a rational function provides a summary of the nature of the inconsistency for any domain size and it provides a direct way of comparing knowledge bases in terms of their respective rational functions. As pointed out in [62], the approaches to measure inconsistency either take into account the number of formulas that are required to produce an inconsistency, or the proportion of the language that is affected by the inconsistency, this is, the more propositional variables affected, the more inconsistent the base. In [62], Hunter et. al. proposes an alternative measure takes into account both the number of variables affected by the inconsistency and the distribution of the inconsistency among the formulae of the base. They use existing inconsistency measures in order to define a game in coalitional form, and then to use the Shapley value to obtain an inconsistency measure that indicates the contribution of each formula to the overall inconsistency in the knowledge base. The work in [10] proposes an axiomatic definition of a dirtiness measure. The work draws the attention to the notion of a cluster and explains the importance of such unit in measuring inconsistency in a database. Furthermore, the axioms proposed allow to describe measures based on the variation of values within a data set, making it possible to use of well known statistical measures. More recently, [63] shows that the notion of inconsistency is a multi-dimensional concept where different measures provide different insights. The authors also explore relationships between measures of inconsistency and measures of information in terms of the trade-offs they identify when using for inconsistency resolution or management.

7 Summary and Outlook

We have developed a general framework based on incision functions that allows to define different semantics for query answering in Datalog+/– ontologies. Within this framework, we proposed a novel semantics, called the *k-lazy answers*, for inconsistency-tolerant query answering that relaxes the notion of (data) repairs. Among the benefits of this semantics, we distinguish: (i) for any given k, it yields a superset of the answers yielded by the intersection semantics; (ii) for any k, the union of i-lazy answers, with $0 \leqslant i \leqslant k$, is guaranteed to be consistent—we call these the union-k-lazy answers; and (iii) both k-lazy and its derived union-k-lazy semantics can be computed in polynomial time in the data complexity under fixed parameter assumptions for linear Datalog+/–. These results show that it is possible to go beyond the restricted possibilities of classical consistent query answering and obtain useful answers while maintaining tractability.

We also show that our framework captures previously studied inconsistency-tolerant semantics (extended in this work for Datalog+/– ontologies). Furthermore, we studied the complexity of query answering for both guarded and linear Datalog+/– under the different semantics. Finally, we show that query answering in linear Datalog+/– is first-order rewritable under the intersection semantics, and therefore very efficiently computable in the data complexity.

Future work will focus on the implementation and empirical evaluation of our framework over synthetic and real-world data. Furthermore, we will continue to investigate possible generalizations of lazy answers as well as more expressive languages, in pursuit of possible ways in which soundness and completeness may be approached without losing scalability.

References

[1] T. Lukasiewicz, M. V. Martinez, G. I. Simari, Inconsistency handling in Datalog+/– ontologies, in: Proc. of ECAI, 2012, pp. 558–563.

[2] T. Lukasiewicz, M. V. Martinez, G. I. Simari, Inconsistency-tolerant query rewriting for linear Datalog+/–, in: Proc. of Datalog, 2012, pp. 123–134.

[3] A. Calì, G. Gottlob, T. Lukasiewicz, A general Datalog-based framework for tractable query answering over ontologies, J. Web Sem. 14 (2012) 57–83.

[4] C. Beeri, M. Y. Vardi, The implication problem for data dependencies, in: Proc. of ICALP, Vol. 115 of LNCS, 1981, pp. 73–85.

[5] A. Calì, G. Gottlob, M. Kifer, Taming the infinite chase: Query answering under expressive relational constraints, in: Proc. of Description Logics, 2008.

[6] R. Fagin, P. G. Kolaitis, R. J. Miller, L. Popa, Data exchange: Semantics and query answering, Theor. Comput. Sci. 336 (1) (2005) 89–124.

[7] A. Deutsch, A. Nash, J. B. Remmel, The chase revisited, in: Proc. of PODS, 2008, pp. 149–158.

[8] S. O. Hansson, Semi-revision, J. Appl. Non-Classical Logic (7) (1997) 151–175.

[9] D. Lembo, M. Lenzerini, R. Rosati, M. Ruzzi, D. F. Savo, Inconsistency-tolerant semantics for description logics, in: Proc. of RR, 2010, pp. 103–117.

[10] M. V. Martinez, A. Pugliese, G. I. Simari, V. S. Subrahmanian, H. Prade, How dirty is your relational database? An axiomatic approach, in: Proc. of ECSQARU, 2007, pp. 103–114.

[11] S. O. Hansson, Kernel contraction, The Journal of Symbolic Logic 59 (3) (1994) 845–859.

[12] A. Calì, G. Gottlob, T. Lukasiewicz, A general Datalog-based framework for tractable query answering over ontologies, in: Proc. of PODS, 2009, pp. 77–86.

[13] A. Calì, G. Gottlob, A. Pieris, Query rewriting under non-guarded rules, in: Proc. of AMW, 2010.

[14] D. Lembo, M. Lenzerini, R. Rosati, M. Ruzzi, D. F. Savo, Query rewriting for inconsistent DL-Lite ontologies, in: Proc. of RR, 2011, pp. 155–169.

[15] A. Calì, G. Gottlob, A. Pieris, Advanced processing for ontological queries, Proc. of VLDB 3 (1–2) (2010) 554–565.

[16] A. Calì, G. Gottlob, A. Pieris, Query answering under non-guarded rules in Datalog+/–, in: Proc. of RR, 2010, pp. 1–17.

[17] M. Arenas, L. E. Bertossi, J. Chomicki, Consistent query answers in inconsistent databases, in: Proc. of PODS, 1999, pp. 68–79.

[18] J. Chomicki, Consistent query answering: Five easy pieces, in: Proc. of ICDT, 2007, pp. 1–17.

[19] S. Schlobach, R. Cornet, Non-standard reasoning services for the debugging of description logic terminologies, in: Proc. of IJCAI, 2003, pp. 355–362.

[20] B. Parsia, E. Sirin, A. Kalyanpur, Debugging OWL ontologies, in: Proc. of WWW, 2005, pp. 633–640.

[21] Z. Huang, F. V. Harmelen, A. T. Teije, Reasoning with inconsistent ontologies, in: Proc. of IJCAI, 2005, pp. 349–350.

[22] G. Qi, J. Du, Model-based revision operators for terminologies in description logics, in: Proc. of IJCAI, 2009, pp. 891–897.

[23] Y. Ma, P. Hitzler, Paraconsistent reasoning for OWL 2, in: Proc. of RR, 2009, pp. 197–211.

[24] P. Haase, F. V. Harmelen, Z. Huang, H. Stuckenschmidt, Y. Sure, A framework for handling inconsistency in changing ontologies, Springer, 2005, pp. 353–367.

[25] D. A. Bell, G. Qi, W. Liu, Approaches to inconsistency handling in description-logic based ontologies, in: Proc. of OTM Workshops, 2007, pp. 1303–1311.

[26] R. Rosati, On the complexity of dealing with inconsistency in description logic ontologies, in: Proc. of IJCAI, 2011, pp. 1057–1062.

[27] M. Bienvenu, R. Rosati, Tractable approximations of consistent query answering for robust ontology-based data access, in: Proc. of IJCAI, 2013, pp. 775–781.

[28] O. Pivert, H. Prade, Detecting suspect answers in the presence of inconsistent information, in: Proc. of FoIKS, 2012, pp. 278–297.

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

[30] M. Bienvenu, First-order expressibility results for queries over inconsistent DL-Lite knowledge bases, in: Proc. of Description Logics, 2011.

[31] M. Bienvenu, Inconsistency-tolerant conjunctive query answering for simple ontologies., in: Y. Kazakov, D. Lembo, F. Wolter (Eds.), Proc. of DL, Vol. 846, CEUR-WS.org, 2012.

[32] D. Lembo, M. Lenzerini, R. Rosati, M. Ruzzi, D. F. Savo, Inconsistency-tolerant first-order rewritability of DL-Lite with identification and denial assertions, in: Proc. of Description Logics, 2012.

[33] C. E. Alchourron, P. Gardenfors, D. Makinson, On the logic of theory change: Partial meet contraction and revision functions, The Journal of Symbolic Logic 50 (2) (1985) 510–530.

[34] P. Gardenfors, Knowledge in flux: modeling the dynamics of epistemic states, MIT Press, Cambridge, Mass., 1988.

[35] G. Qi, W. Liu, D. A. Bell, A revision-based approach to handling inconsistency in description logics, Artif. Intell. Rev. 26 (1-2) (2006) 115–128.

[36] N. da Costa, On the theory of inconsistent formal systems, N. Dame J. of Formal Logic 15 (4) (1974) 497–510.

[37] P. Besnard, T. Schaub, Signed systems for paraconsistent reasoning, J. Autom. Reas. 20 (1) (1998) 191–213.

[38] N. Belnap, A useful four valued logic, Modern Uses of Many Valued Logic (1977) 8–37.

[39] J. Grant, Classifications for inconsistent theories, N. Dame J. of Formal Logic 19 (3) (1978) 435–444.

[40] L. Cholvy, A modal logic for reasoning with contradictory beliefs which takes into account the number and the reliability of the sources, in: Proc. of ECSQARU, 2005, pp. 390–401.

[41] H. A. Blair, V. S. Subrahmanian, Paraconsistent logic programming, Theor. Comp. Sci. 68 (2) (1989) 135–154.

[42] M. Fitting, Bilattices and the semantics of logic programming, J. Log. Program. 11 (1–2) (1991) 91–116.

[43] S. de Amo, M. S. Pais, A paraconsistent logic programming approach for querying inconsistent databases, Int. J. Approx. Reasoning 46 (2) (2007) 366–386.

[44] S. d. Amo, W. A. Carnielli, J. a. Marcos, A logical framework for integrating inconsistent information in multiple databases, in: Proc. of FoIKS, Springer-Verlag, London, UK, 2002, pp. 67–84.

[45] L. A. Nguyen, A. Szalas, Three-valued paraconsistent reasoning for semantic web agents, in: Proc. of KES-AMSTA, Springer-Verlag, 2010, pp. 152–162.

[46] I. Horrocks, U. Sattler, S. Tobies, Reasoning with individuals for the description logic $\mathcal{SHIQ}$, in: Proc. of ICAD, Springer-Verlag, London, UK, 2000, pp. 482–496.

[47] S. C. Kleene, Introduction to metamathematics, Bibl. Matematica, North-Holland, Amsterdam, 1952.

[48] A. Bloesch, A tableau style proof system for two paraconsistent logics., N. Dame J. of Formal Logic 34 (2) (1993) 295–301.

[49] R. Reiter, A logic for default reasoning, Artif. Intel. 13 (1-2) (1980) 81–132.

[50] C. Baral, S. Kraus, J. Minker, Combining multiple knowledge bases, TKDE 3 (2) (1991) 208–220.

[51] D. Touretzky, The mathematics of inheritance systems, Morgan Kaufmann, 1986.

[52] M. Kifer, E. L. Lozinskii, A logic for reasoning with inconsistency, J. Autom. Reasoning 9 (2) (1992) 179–215.

[53] K. Thirunarayan, M. Kifer, A theory of nonmonotonic inheritance based on annotated logic, Artif. Intell. 60 (1) (1993) 23–50.

[54] S. Benferhat, D. Dubois, H. Prade, Some syntactic approaches to the handling of inconsistent knowledge bases: A comparative study part 1: The flat case, Studia Logica 58 (1) (1997) 17–45.

[55] G. R. Simari, R. P. Loui, A mathematical treatment of defeasible reasoning and its implementation, Artif. Intell. 53 (2-3) (1992) 125–157.

[56] P. M. Dung, On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games, Artif. Intell. 77 (1995) pp. 321–357.

[57] H. Prakken, G. Sartor, Argument-based extended logic programming with defeasible priorities, J. Appl. Non-Classical Logic 7 (1).

[58] S. Benferhat, D. Dubois, J. Lang, H. Prade, A. Saffiotti, P. Smets, A general approach for inconsistency handling and merging information in prioritized knowledge bases, in: Proc. of KR, 1998, pp. 466–477.

[59] E. L. Lozinskii, Resolving contradictions: A plausible semantics for inconsistent systems, J. Autom. Reasoning 12 (1) (1994) 1–31.

[60] A. Hunter, S. Konieczny, Approaches to measuring inconsistent information, in: Inconsistency Tolerance, 2005, pp. 191–236.

[61] J. Grant, A. Hunter, Analysing inconsistent first-order knowledgebases, Artif. Intell. 172 (8-9) (2008) 1064–1093.

[62] A. Hunter, S. Konieczny, On the measure of conflicts: Shapley inconsistency values, Artif. Intell. 174 (14) (2010) 1007–1026.

[63] J. Grant, A. Hunter, Measuring the good and the bad in inconsistent information, in: Proc. of IJCAI, AAAI Press, 2011, pp. 2632–2637.

Appendix: Proofs

Proof of Theorem 1. $(\Rightarrow)$ Suppose $D' \in \mathit{DRep}(KB)$. For each cluster $i \in \mathit{clusters}(KB)$, let $c_i = i - D'$ and $\overline{c}_i = i - c_i = i \cap D'$. Note that c_i contains the atoms in i that were removed from D when computing D', while $\overline{c}_i$ is the complement of c_i relative to i, i.e., it contains the atoms in i that remain in D'. Clearly, $\overline{c}_i \subseteq D'$, and since D' is consistent relative to Σ, so is $\overline{c}_i$. Suppose some $c'_i \subset c_i$ exists such that $\overline{c}'_i = i - c'_i$ is consistent relative to Σ (i.e., it would have been possible to remove a subset of c_i, and the result could have also been consistent). If this is the case, then $\overline{c}_i \subset \overline{c}'_i$, and so there exists at least one atom $\beta \in \overline{c}'_i - \overline{c}_i$ such that $\beta \notin D'$. This means that $D' \cup \{\beta\}$ is consistent relative to Σ, and therefore D' is not a data repair, leading to a contradiction. As a result, there is no $c'_i \subset c_i$ such that $\overline{c}'_i = i - c'_i$ is consistent relative to Σ.

Let $C = \bigcup_{i \in \mathit{clusters}(KB)} c_i$, with c_i as above, and so $D' = D - C$. It remains to show that C is an optimal incision. Clearly, $C \subseteq \bigcup_{i \in \mathit{clusters}(KB)} i$ by construction of the c_i's. Also, since $D' = D - C$, and D' is a data repair for KB, it holds that $mods(D - C, \Sigma) \neq \emptyset$. Therefore, C is an incision function over $\mathit{clusters}(KB)$. Furthermore, suppose that C is not optimal. Then, some $B \subset C$ exists such that $mods(D - B, \Sigma) \neq \emptyset$. Hence, $B = \bigcup_{i \in \mathit{clusters}(KB)} c'_i$, and $c'_i \subset c_i$ and $mods(i - c'_i, \Sigma) \neq \emptyset$ for some $i \in \mathit{clusters}(KB)$. But this is not possible, as argued above. Therefore, C is optimal.

$(\Leftarrow)$ Suppose there is an optimal incision $\chi_{\mathit{opt}}(\mathit{clusters}(KB))$ over KB such that $D' = D - \chi_{\mathit{opt}}(\mathit{clusters}(KB))$. We now prove that

(i) $D' \subseteq D$,

(ii) D' is consistent relative to Σ, i.e., $mods(D', \Sigma) \neq \emptyset$, and

(iii) no $D'' \subseteq D$ exists such that $D' \subset D''$ and $mods(D'', \Sigma) \neq \emptyset$.

(i) holds trivially. (ii) also holds, since χ_{opt} is an incision function, and thus $mods(D', \Sigma) \neq \emptyset$, by the definition of incision functions. To prove (ii), suppose some $D'' \subseteq D$ exists such that $D' \subset D''$ and $mods(D'', \Sigma) \neq \emptyset$. Then, some $B \subset \chi_{\mathit{opt}}(\mathit{clusters}(KB))$ exists such that $mods(D - B, \Sigma) \neq \emptyset$, but this contradicts the fact that $\chi_{\mathit{opt}}(\mathit{clusters}(KB))$ is an optimal incision over KB. Thus, D' is a data repair. $\Box$

Proof of Theorem 2. As for membership in co-NP, deciding whether $KB \not\models_{\mathit{Cons}} Q$ can be done by guessing and verifying some $D' \in \mathit{DRep}(KB)$ such that $(D', \Sigma) \not\models Q$. This can be done in polynomial time in the data complexity in the guarded and the linear case. In particular, verifying that $D' \in \mathit{DRep}(KB)$ is possible in polynomial time by checking that (D', Σ) is consistent and that every (D'', Σ) with larger $D'' \subseteq D$, containing one atom more, is inconsistent.

Hardness for co-NP follows from the result that deciding ABox assertions from *DL-Lite*$_{\mathit{core}}$ knowledge bases under the consistent answer semantics is co-NP-hard in the data complexity [9], since guarded Datalog+/– generalizes *DL-Lite*$_{\mathit{core}}$ [3]. $\Box$

We next prove the following two lemmas, which will be used in the proofs of Theorem 3 and Propositions 1 and 2.

Lemma 4 *Let $KB = (D, \Sigma)$ be a Datalog+/– ontology. For every $\alpha \in D$, we have that $\alpha \notin cl$ for every $cl \in \mathit{clusters}(KB)$ iff $\alpha \in D'$ for every $D' \in \mathit{DRep}(KB)$.*

Proof. $(\Rightarrow)$ Suppose $\alpha \in D$ and $\alpha \notin cl$ for every $cl \in \mathit{clusters}(KB)$. Towards a contradiction, suppose $\alpha \notin D'$ for some $D' \in \mathit{DRep}(KB)$. By Theorem 1, there is an optimal incision function χ_{opt} such that

$D' = D - \chi_{opt}(\mathit{clusters}(KB))$. But this contradicts $\alpha \notin cl$ for every cluster cl. Thus, we have that $\alpha \in D'$ for all $D' \in \mathit{DRep}(KB)$.

($\Leftarrow$) Conversely, suppose $\alpha \in D'$ for every $D' \in \mathit{DRep}(KB)$. Towards a contradiction, suppose $\alpha \in cl$ for some $cl \in \mathit{clusters}(KB)$. Hence, there is at least one culprit $c = \{q_1, \ldots, q_k, \alpha\} \subseteq cl$. Note that any data repair can only have a strict maximal subset of c that is consistent relative to Σ, otherwise the whole set would be inconsistent, and each of these sets is obtained by removing exactly one element from c. Let D' be an arbitrary data repair. Then, $A = (D' - cl) \cup \{q_1, \ldots, q_k\}$ is consistent relative to Σ, since any set of elements that could conflict with $\{q_1, \ldots, q_k\}$ belongs to cl and so was removed. Let $D'' = A \cup r$, where $r \subset cl$ is the largest strict subset of cl such that $A \cup r$ is consistent relative to Σ. We now show that D'' is a repair. Clearly, $D'' \subseteq D$ and, by construction, D'' is consistent relative to Σ. All other elements in $D - D''$ are elements from cl in conflict with q_i, so there is no strict superset of D'' that is consistent relative to Σ. Thus, D'' is a repair and $\alpha \notin D''$, which is a contradiction. Hence, $\alpha \notin cl$ for every $cl \in \mathit{clusters}(KB)$. $\Box$

Lemma 5 *Let $KB = (D, \Sigma)$ be a Datalog+/– ontology. Then, every $\alpha \in D$ belongs to at most one cluster $cl \in \mathit{clusters}(KB)$.*

Proof. Towards a contradiction, suppose $\alpha \in D$ belongs to two different $cl_1, cl_2 \in \mathit{clusters}(KB)$. Hence, $\alpha \in c_1 \subseteq cl_1$ and $\alpha \in c_2 \subseteq cl_2$ for two culprits c_1 and c_2, which implies $cl_1 = cl_2$, a contradiction. Hence, every $\alpha \in D$ belongs to at most one $cl \in \mathit{clusters}(KB)$. $\Box$

We are now ready to prove Theorem 3 and Propositions 1 and 2.

Proof of Theorem 3. We prove that $D_I = D_{all}$, where:

$$\begin{aligned} D_I &= \textstyle\bigcap_{D' \in \mathit{DRep}(KB)} D' \text{ and} \\ D_{all} &= D - \chi_{all}(\mathit{clusters}(KB)) = D - \textstyle\bigcup_{cl \in \mathit{clusters}(KB)} cl\,. \end{aligned}$$

Let $\alpha \in D_I$. That is, $\alpha \in D'$ for all $D' \in \mathit{DRep}(KB)$. By Lemma 4, the latter is equivalent to $\alpha \notin cl$ for all $cl \in \mathit{clusters}(KB)$. That is, $\alpha \notin \bigcup_{cl \in \mathit{clusters}(KB)} cl$, which is in turn equivalent to $\alpha \in D_{all}$. $\Box$

Proof of Theorem 4. As for membership in co-NP, deciding whether $KB \not\models_{ICons} Q$ can be done by guessing and verifying some $D^\star \subseteq D$ such that $(D^\star, \Sigma) \not\models Q$, and, for every $\alpha \in D - D^\star$, some $D'_\alpha \in \mathit{DRep}(KB)$ such that $\alpha \notin D'_\alpha$. Note that the latter implies that $D^\star$ is the intersection of a collection of data repairs, which in turn implies $D_I \subseteq D^\star$ for the intersection D_I of *all* data repairs. Hence, $(D^\star, \Sigma) \not\models Q$ implies $(D_I, \Sigma) \not\models Q$, since "$\models$" is monotonic. The above guessing and verifying can be done in polynomial time in the data complexity in the guarded and the linear case.

Hardness for co-NP follows from the result that deciding ABox assertions from $\mathcal{EL}_\perp$ knowledge bases under the consistent answer semantics is co-NP-hard in the data complexity [26], since guarded Datalog+/– generalizes $\mathcal{EL}$ [3] and (as easily seen) also $\mathcal{EL}_\perp$. $\Box$

Proof of Proposition 1. To prove that $\chi_{lazy}(k, \mathit{clusters}(KB))$ is indeed an incision function, we have to show that:

(i) $\chi_{lazy}(k, \mathit{clusters}(KB)) \subseteq \bigcup_{cl \in \mathit{clusters}(KB)} cl$ and

(ii) $\mathit{mods}(D - \chi_{lazy}(k, \mathit{clusters}(KB)), \Sigma) \neq \emptyset$.

(i) By definition, $\chi_{lazy}(k, \mathit{clusters}(KB)) = \bigcup_{cl \in \mathit{clusters}(KB)} c_{cl}$, where $c_{cl} \in \chi_{k\text{-}cut}(cl)$ is a subset of cl of size at most k.

(ii) Suppose $mods(D - \chi_{lazy}(k, \textit{clusters}(KB)), \Sigma) = \emptyset$. Recall that $\chi_{lazy}(k, \textit{clusters}(KB)) = \bigcup_{cl \in \textit{clusters}(KB)} c_{cl}$, where $c_{cl} \in \chi_{\textit{k-cut}}(cl)$. Then, there exists a minimal set $\textit{culp} = \{\alpha_1, ..., \alpha_n\} \subseteq D - \chi_{lazy}(k, \textit{clusters}(KB))$ such that $mods(\textit{culp}, \Sigma) = \emptyset$. This is only possible (since, by Lemma 5, clusters are disjoint sets of atoms) if for the cluster cl containing *culp*, the set c_i selected from $\chi_{\textit{k-cut}}(cl)$ is such that it does not contain *culp*. If this is the case, then $mods(cl - c_i, \Sigma) = \emptyset$, but this is not possible by the definition of $\chi_{\textit{k-cut}}(cl)$. As a result, $mods(D - \chi_{lazy}(k, \textit{clusters}(KB)), \Sigma) \neq \emptyset$. □

Proof of Proposition 2. (a) Given that $D' = D - \chi_{lazy}(k, \textit{clusters}(KB))$, where $\chi_{lazy}(k, \textit{clusters}(KB))$ is a k-lazy function for KB, and, since by Proposition 1, $\chi_{lazy}(k, \textit{clusters}(KB))$ is an incision function over $\textit{clusters}(KB)$, the statement follows directly from Definition 5.

(b) If $\beta \in D$ and $\beta \notin D'$, then $\beta \in \chi_{lazy}(k, \textit{clusters}(KB))$. Thus, by Lemma 5, there exists a unique cluster cl such that $\beta \in cl$. Let $B = cl - \chi_{lazy}(k, \textit{clusters}(KB))$. Clearly, $B \subseteq D$, $\beta \notin B$, and B contains the atoms in cl that still remain in D'. As argued in (a), $mods(D', \Sigma) \neq \emptyset$, and so $mods(B, \Sigma) \neq \emptyset$. By the definition of $\chi_{lazy}(k, \textit{clusters}(KB)))$, if $\beta \in \chi_{lazy}(k, \textit{clusters}(KB))$, then there exists some $c_i \in \chi_{\textit{k-cut}}(cl)$ with $\beta \in c_i$. By Equation 1, we have that there exists no $c'_i \subset c_i$ such that $mods(cl - c'_i, \Sigma) \neq \emptyset$. It thus follows that $mods(cl - (c_i - \{\beta\}), \Sigma) = mods(B \cup \{\beta\}, \Sigma) = \emptyset$. □

The following lemma will be used in the proof of Proposition 4.

Lemma 6 *Let $KB = (D, \Sigma)$ be a Datalog+/– ontology, and $\alpha \in D$. Then, $KB \models_{\textit{ICons}} \alpha$ iff $\alpha \notin cl$ for every $cl \in \textit{clusters}(KB)$.*

Proof. ($\Rightarrow$) Suppose $KB \models_{\textit{ICons}} \alpha$. Since $\alpha \in D$, and data repairs are maximal subsets of D, it thus follows that α belongs to every data repair. By Lemma 4, $\alpha \notin cl$ for every $cl \in \textit{clusters}(KB)$.

($\Leftarrow$) Suppose $\alpha \notin cl$ for every $cl \in \textit{clusters}(KB)$. By Lemma 4, $\alpha \in D'$ for every $D' \in \textit{DRep}(KB)$. Hence, α belongs to the intersection of all data repairs, which in turn implies $KB \models_{\textit{ICons}} \alpha$. □

We are now ready to prove Proposition 4.

Proof of Proposition 4. (a) Let $KB \models_{\textit{ICons}} Q$. That is, $(D_I, \Sigma) \models Q$, where $D_I = \bigcap_{D' \in \textit{DRep}(D,\Sigma)} D'$. By Lemma 6, each $\alpha_i \in D_I$ is such that $\alpha_i \notin cl$ for every $cl \in \textit{clusters}(KB)$. Thus, for any $k \geqslant 0$, it holds that $\alpha_i \notin c_j$ for any $c_j \in \chi_{\textit{k-cut}}(cl)$ and cluster cl. Hence, for every $\alpha_i \in D_I$, we have $\alpha_i \in \textit{lrep}$, for every $\textit{lrep} \in \textit{LRep}(k, KB)$, and so $KB \models_{\textit{k-LCons}} \alpha_i$. Thus, $KB \models_{\textit{k-LCons}} Q$.

(b) Since $\chi_{\textit{0-cut}}(cl) = cl$ for every $cl \in \textit{clusters}(KB)$, the only lazy repair for $k = 0$ is given by:

$$\textit{lrep} = D - \chi_{lazy}(0, \textit{clusters}(KB)) = D - \bigcup_{cl \in \textit{clusters}(KB)} cl = D - \bigcup_{cl \in \textit{clusters}(KB)} \chi_{all}(cl).$$

By Theorem 3, $KB \models_{\textit{ICons}} Q$ is equivalent to $KB \models_{\textit{0-LCons}} Q$. □

Proof of Proposition 5. By Theorem 1, the consistent answers for query Q can be computed using $\chi_{opt}(\textit{clusters}(KB))$. The difference between Definitions 8 and 10 is that in the latter, $\chi_{\textit{k-cut}}(cl)$ makes an additional constraint bounding the size of the incisions by k. Clearly, $\chi_{lazy}(k, \textit{clusters}(KB))$ may not be optimal (i.e., the minimal cut needed to make the cluster consistent might be greater than k in which case the whole cluster is removed). However, given $i \in \textit{clusters}(KB)$, there exists $k_i \geqslant 0$ such that it is the minimum natural number for which $\chi_{k_i\textit{-cut}}(i) = \chi_{(k_i+n)\textit{-cut}}(i)$ for all $n \geqslant 1$. Let k be the maximum of those k_i across all $i \in \textit{clusters}(KB)$. Then, $\chi_{lazy}(k, \textit{clusters}(KB)) = \chi_{opt}(\textit{clusters}(KB))$. It thus follows that $KB \models_{\textit{Cons}} Q$ iff $KB \models_{\textit{k-LCons}} Q$, for the above k. □

Proof of Theorem 6.

We prove co-NP-hardness by a reduction from the co-NP-complete problem of deciding the validity of propositional formulas in 3-DNF. Let $\Phi = C_1 \vee \cdots \vee C_m$ be an instance of 3-DNF validity, where the C_i's are conjunctions containing exactly three literals, and X is the set of n propositional variables occurring in Φ. We now construct an instance of k-lazy BCQ evaluation.

Let $\Delta = X$, and we have the predicates p_Q and p_{null} of arity 0, the predicates T and F of arity 1, and for each combination B among $FFF, FFT, \ldots, TTT$ the predicate p_B of arity 3.

The database D comprises (i) p_{null}, (ii) all $T(V)$ and $F(V)$ with $V \in X$, and (iii) all $p_{B_1B_2B_3}(V_1, V_2, V_3)$ such that Φ contains the clause $C_i = L_1 \wedge L_2 \wedge L_3$, where $B_i = T$ (resp., $B_i = F$) iff $L_i = V_i$ (resp., $L_i = \neg V_i$). The set of guarded TGDs Σ_T contains (i) for each combination $B = B_1B_2B_3$ among $FFF, FFT, \ldots,$ TTT, the guarded TGD $p_{B_1B_2B_3}(V_1, V_2, V_3) \wedge B_1(V_1) \wedge B_2(V_2) \wedge B_3(V_3) \rightarrow p_Q$, and (ii) for each variable $V \in X$, the guarded TGD $T(V) \wedge F(V) \rightarrow p_{null}$. The set of negative constraints Σ_{NC} contains (i) for each $V \in X$, a constraint $T(V) \wedge F(V) \wedge p_{null} \rightarrow \bot$, and (ii) the constraint $p_{null} \rightarrow \bot$. Finally, the BCQ is given by $Q = p_Q$. Clearly, this instance can be constructed in polynomial time.

We now prove that Φ is valid iff *Yes* is an $(n+1)$-lazy answer for Q to $KB = (D, \Sigma_T \cup \Sigma_{NC})$. Note that $clusters(KB) = \{D\}$, and any way of choosing one of $T(V_j)$ or $F(V_j)$ for $j \in \{1, \ldots, n\}$ (i.e., any truth value assignment over X), along with p_{null} to remove from D (an incision of size $n+1$) produces an $(n+1)$-lazy repair.

($\Rightarrow$) Let Φ be valid. Thus, at least one of the TGDs with head p_Q has its body satisfied in each of the lazy repairs, and therefore the query Q is entailed from KB under the lazy semantics.

($\Leftarrow$) If *Yes* is an $(n+1)$-lazy answer for Q to KB, then Q is entailed from every $(n+1)$-lazy repair. Thus, the body of at least one of the TGDs with head p_Q is satisfied in every lazy repair. So, for every truth assignment, some clause evaluates to true. That is, Φ is valid. $\Box$

The following Lemma will be useful in proving Propositon 6.

Lemma 7 *Let $KB = (D, \Sigma)$ be a guarded Datalog+/– ontology and $G = (V, E)$ be the chase graph for KB for a conjunction of ground atoms α. Algorithm* findKernels(G, α) *computes the set of minimal subsets c of D such that $(c, \Sigma_T) \models \alpha$.*

Proof. We want to prove that $c \in \mathit{findKernels}(G, \alpha)$ iff $c \subseteq D$, $(c, \Sigma_T) \models \alpha$, and there is no $c' \subset c$ such that $(c', \Sigma_T) \models \alpha$.

Let α be a conjunction of ground atoms of the form $\alpha = a_1 \wedge \ldots \wedge a_n$ W.l.o.g. suppose that $KB \models \alpha$, since, otherwise, we can easily verify that the algorithm returns the empty set as desired.

($\Rightarrow$) First we show that if $c \in \mathit{findKernels}(G, \alpha)$ then $c \subseteq D$, $(c, \Sigma_T) \models \alpha$, and there is no $c' \subset c$ such that $(c', \Sigma_T) \models \alpha$.

We will prove this result by induction on k, the total number of TGDs applied in G in $\mathit{chase}(KB)$. The application of a TGD σ corresponds to the existence of a homomorphism h that maps $\mathit{body}(v)$ to a set of atoms in the chase computed so far. Therefore, if a single TGD is applied under different homomorphisms in the chase procedure, then each of them counts as a different application.

Base case: If $k = 0$, no TGD was applied, then it must be the case that $\alpha \subseteq D$ and that $\mathit{findKernels}(G, \alpha) = \{\{\alpha\}\}$. Clearly, $(\mathit{findKernels}(G, \alpha), \Sigma_T) \models \alpha$ and it is trivially a minimal set in that sense.

Suppose that for $k \geqslant 0$, for each $c \in \mathit{findKernels}(G, \alpha)$, $c \subseteq D$, $(c, \Sigma_T) \models \alpha$, and there is no $c' \subset c$ such that $(c', \Sigma_T) \models \alpha$.

Let's prove that for $k+1$, for each $c \in \mathit{findKernels}(G, \alpha)$, $c \subseteq D$, $(c, \Sigma_T) \models \alpha$ and there is no $c' \subset c$ such that $(c', \Sigma_T) \models \alpha$.

Let c be an arbitrary element in *findKernels*(G, α). From line 10 in the algorithm we can see that c has the form $c = c_1 \cup \ldots \cup c_n$, where $c_i \in$ *kernels*$[a_i]$, with $1 \leqslant i \leqslant n$. If $c_i = \{a_i\}$ then, by the base case, c_i is a minimal subset of D such that $(c_i, \Sigma_T) \models a_i$. Otherwise, by construction of the mapping *kernels*, there exists σ and a homomorphism h such that $h(head(\sigma))$ maps to a_i with $body(\sigma) = b_1 \wedge \ldots \wedge b_m$. Let $c'_1, \ldots, c'_m$ be such that $c'_j \in$ *kernels*$[h(b_j)]$ with $1 \leqslant j \leqslant m$. By the inductive hypothesis, each c'_j is a minimal subset of D such that $(c'_j, \Sigma_T) \models h(b_j)$, since the number of TGDs needed to derive any of the atoms in the body of σ must be smaller than $k + 1$ by construction of the chase graph. Furthermore, since each c'_j is a minimal subset of D needed to derive $h(b_j)$, then $(\bigcup_{1 \leqslant j \leqslant m} c'_j, \Sigma_T) \models h(body(\sigma))$ and $(\bigcup_{1 \leqslant j \leqslant m} c'_j, \Sigma_T) \models \alpha$. Therefore, it must be the case that for some $c_i \in$ *kernels*(a_i), $c_i = \bigcup_{1 \leqslant j \leqslant m} c'_j$, so c_i is a minimal subset of D such that $(c_i, \Sigma_T) \models a_i$. Then, every c_i is a minimal subset of D necessary to derive a_i and, by the checks performed in line 10 of the algorithm, we have that $\bigcup_{1 \leqslant i \leqslant n} c_i$ is a minimal subset of D such that $(\bigcup_{1 \leqslant i \leqslant n} c_i, \Sigma_T) \models a_1 \wedge \ldots \wedge a_n = \alpha$.

($\Leftarrow$) Second, we show that for every $c \subseteq D$ such that $(c, \Sigma_T) \models \alpha$ and there is no $c' \subset c$ such that $(c', \Sigma_T) \models \alpha$, we have that $c \in$ *findKernels*(G, α). As before we assume that $KB \models \alpha$ and prove this result by induction on k, the total number of TGDs applied in G.
Base case: If $k = 0$ then $c = \alpha \subset D$ and trivially *findKernels*$(G, \alpha) = \{\{\alpha\}\}$.
Suppose that for $k \geqslant 0$, for each $c \subset D$ such that $(c, \Sigma_T) \models \alpha$ and there is no $c' \subset c$ such that $(c', \Sigma_T) \models \alpha$, then $c \in$ *findKernels*(G, α).
Let's prove that for $k + 1$, for each $c \subset D$ such that $(c, \Sigma_T) \models \alpha$ and there is no $c' \subset c$ such that $(c', \Sigma_T) \models \alpha$, then $c \in$ *findKernels*(G, α).
Let c be an arbitrary set such that $c \subset D$, $(c, \Sigma_T) \models \alpha$, and there is no $c' \subset c$ such that $(c', \Sigma_T) \models \alpha$. Let b be an arbitrary atom in c. Since c is minimal then either $b = a_i$ for some $a_i \in \alpha$, or b is necessary to derive a_i. If the first case is true, then the algorithm inserts the set $\{b\}$ into *kernels*$[a_i]$, and thus b belongs to some set $c' \in$ *findKernels*(G, α).

If $b \neq a_i$ for every $a_i \in \alpha$, then it must be the case that there exists a TGD σ and a homomorphism h such that $h(head(\sigma))$ maps to some $a_i \in \alpha$, and b belongs to a β-kernel where $\beta = h(body(\sigma))$. As we can see in line 6, it must be the case that $b \in c_b$ with $c_b \in$ *findKernels*$(G, h(body(\sigma)))$. By construction of the algorithm, we have then that $b \in ker_{a_i}$ with $ker_{a_i} \in$ *kernels*$[a_i]$. Furthermore, since every $b \in c$ is necessary to derive some $a_i \in \alpha$ (and the fact that c is minimal) then, it must be the case that $c = \bigcup_{a_i \in \alpha} ker_{a_i}$, with $ker_{a_i} \in$ *kernels*$[a_i]$. So, we have that $c \in$ *findKernels*(G, α). □

Proof of Proposition 6.

Lemma 7 shows that given a ground conjunction of atoms α, algorithm findKernels (KB, α) computes the set of all minimal subsets c of D such that $(c, \Sigma_T) \models \alpha$. Algorithm findClusters$(KB)$ computes then for each ground negative constraint υ, the set of all minimal subsets c_υ such that $(c_\upsilon, \Sigma_T) \models body(\upsilon)$. Take an arbitrary υ and c_υ, it is clear then that $mods(c_\upsilon, \Sigma_T \cup \{\upsilon\})) = \emptyset$, then findKernels $(KB, body(\upsilon)) =$ *culprits*$(KB, \{\upsilon\})$. We want to prove that before line 7 *PCul* $=$ *culprits*(KB), i.e., $c \in$ *PCul* iff $c \in$ *culprits*(KB).

($\Rightarrow$) If $c \in$ *PCul* then we want to show that $c \in$ *culprits*(KB). If $c \in$ *PCul* then, by Lemma 7, $c \subseteq D$, $(D, \Sigma_T) \models body(\upsilon)$ for some $\upsilon \in \Sigma_{NC}$. Furthermore, by the checks made in lines 8 and 10, there is no other element in $c' \in$ *PCul* such that $c' \subseteq c$, then $c \in$ *culprit*(KB).

($\Leftarrow$) If $c \in$ *culprits*(KB), then we want to show that $c \in$ *PCul*. Suppose that $c \in$ *culprits*(KB) but $c \notin$ *PCul*. If $c \in$ *culprits*(KB) then $mods(c, \Sigma) = \emptyset$, but then it must be the case that $(c, \Sigma_T) \models body(\upsilon)$ for some $\upsilon \in \Sigma_{NC}$. Then, at some point the algorithm computed PC_υ and $c \in PC_\upsilon$. Then, either (1) c was not added to *PCul* or (2) it was added but a strict subset of c replaced it further along in the computation of *PCul*. If (1) is the case, then there was already an element c' in *PCul* such that $c' \subset c$. But then $(c', \Sigma_T) \models \mathit{body}(v')$, so $\mathit{mods}(c', \Sigma) = \emptyset$, which is not possible because c is a culprit in KB. If (2) is the case then further along in the computation of *PCul*, a set c' replaced c because $c' \subset c$, but again, if this is the case then $(c', \Sigma_T) \models \mathit{body}(v')$ and $\mathit{mods}(c', \Sigma) = \emptyset$, which is impossible. Then, by contradiction, it must the case that $c \in$ *PCul*.

To finalize the proof that Algorithm findClusters(KB) correctly computes the set $\mathit{clusters}(KB)$, it only rests to say that line 12 computes the transitive closure of the overlapping relation Θ over *PCul*, i.e., over the set of *culprits*(KB). □

Proof of Theorem 7. The algorithm first computes the set of clusters of KB. Lines 5–8 test each subset A of cardinality at most k in each cluster: if A can be removed and consistency is restored, and there is no strict subset satisfying these conditions, then A is added as a possible incision for that cluster. Lines 10 and 11 directly apply Definition 11 with the results obtained above: for each possible way of choosing a subset to make an incision, a k-lazy repair is built by removing this set from D. Finally, the algorithm directly applies Definition 12 to compute the set of k-lazy answers. □

The following Lemma helps in proving Proposition 7.

Lemma 8 *Let $KB = (D, \Sigma)$ be a linear Datalog+/– ontology and α be an atom. Then the set of all minimal subsets c of D such that $(c, \Sigma_T) \models \alpha$ is polynomial in $|D|$.*

Proof. Without loss of generality suppose that $KB \models \alpha$. We will prove this result by induction on k, the number of TGDs applied in G to obtain α, where $G = (V, E)$ is the chase graph for KB. Let $\mathit{kernels}(\alpha)$ be the set of all minimal subsets c of D such that $(c, \Sigma_T) \models \alpha$.

Base case: If $k = 0$ then $\alpha \in D$ and then $\mathit{kernels}(\alpha) = \{\{a\}\}$.

Suppose that for $k \geqslant 0$, it holds that $|\mathit{kernels}(\alpha)| \in O(\mathit{poly}(|D|))$, where $\mathit{poly}(|D|)$ is a polynomial over $|D|$.

Let's prove that for $k + 1$, the size of $\mathit{kernels}(a)$ is polynomial in the size of D. Let $\Omega \subseteq \Sigma_T$ be the set of TGDs that are applicable to α, then $\mathit{kernels}(\alpha) = \bigcup_{\sigma \in \Omega} \mathit{kernels}(\mathit{body}(\sigma))$, Clearly, $|\mathit{kernels}(\alpha)| \leqslant |\Omega| \times \mathit{max}_{\sigma \in \Omega}\{|\mathit{kernels}(\mathit{body}(\sigma))|\}$. But, for any $\sigma \in \Omega$, $\mathit{body}(\sigma)$ needs exactly one application of a TGD less than what is needed for α, and then by the inductive hypothesis we have that $|\mathit{kernels}\ (\mathit{body}(\sigma))| \in O(\mathit{poly}(|D|))$.

Therefore, we have (with a slight abuse of notation) $|\mathit{kernels}(\alpha)| \leqslant |\Omega| \times O(\mathit{poly}(|D|))$, and thus $|\mathit{kernels}(\alpha)|$ is polynomial in data complexity. □

Proof of Proposition 7.

The grounding of the constraints in Σ_{NC} can be done in polynomial time since we are assuming that both Σ_{NC} and Σ_T are fixed. Furthermore for a given ground constraint υ of the form $b_1 \wedge \ldots \wedge b_n \rightarrow \bot$, there is a polynomial number of ways of computing each b_i from KB, by Lemma 8, which ensures that for linear Datalog+/– ontologies algorithm findKernels(KB, $\mathit{body}(\upsilon)$) runs in polynomial time for an arbitrary ground constraint υ. The checks for minimality in lines 8 and 10 involve going through a polynomially sized set of potential culprits a polynomial number of times. Finally, the merge of overlapping culprits (since there is a polynomial number of them) can also be done in polynomial time over $|D|$. □

Proof of Lemma 2

($\subseteq$) First, we show that if an atom α is such that $\alpha \in \bigcup_{s_i \in S} \mathit{chase}(s_i, \Sigma)$ then $\alpha \in \mathit{chase}(\bigcup_{s_i \in S} s_i, \Sigma)$.

If $\alpha \in \bigcup_{s_i \in S} \mathit{chase}(s_i, \Sigma)$ then $\alpha \in \mathit{chase}(s_i, \Sigma)$ for some $s_i \in S$, and therefore there exists $\mathit{ker} \subseteq s_i$ such that $\mathit{chase}(\mathit{ker}, \Sigma) \models \alpha$ and there is no $\mathit{ker}' \subset \mathit{ker}$ such that $\mathit{chase}(\mathit{ker}, \Sigma) \models \alpha$. By the monotonicity of the chase procedure, we have that $\mathit{ker} \subseteq \mathit{chase}(s_i, \Sigma) \subseteq \mathit{chase}(\bigcup_{s_i \in S} s_i, \Sigma)$, and therefore $\alpha \in \mathit{chase}(\bigcup_{s_i \in S} s_i, \Sigma)$.

($\supseteq$) Second, we show that if an atom α is such that $\alpha \in \mathit{chase}(\bigcup_{s_i \in S} s_i, \Sigma)$ then $\alpha \in \bigcup_{s_i \in S} \mathit{chase}(s_i, \Sigma)$. Suppose that $\alpha \in \mathit{chase}(\bigcup_{s_i \in S} s_i, \Sigma)$ and $\alpha \notin \bigcup_{s_i \in S} \mathit{chase}(s_i, \Sigma)$. Since *KB* is a linear Datalog+/– ontology, it must be the case that every minimal $\mathit{ker} \subseteq \bigcup_{s_i \in S} s_i$ such that $(\mathit{ker}, \Sigma) \models \alpha$, is a singleton. Therefore, every such *ker* belongs to some $s_i \in S$ and, by monotonicity of the chase procedure, $\mathit{ker} \in \mathit{chase}(s_i, \Sigma)$, which is a contradiction since this implies that $\alpha \in \bigcup_{s_i \in S} \mathit{chase}(s_i, \Sigma)$. □

Proof of Proposition 8.

We have to show that the output of Algorithm lazyConsistentLinear(KB, Q, k) is correct, i.e., given KB and a BCQ Q, *Yes* is the output of the algorithm iff $KB \models_{\mathit{k\text{-}LCons}} Q$.

($\Rightarrow$) First, we show that if the algorithm outputs *Yes* then $KB \models_{\mathit{LCons}} Q$.
If *Yes* is the output of the algorithm then either (1) $(D_I, \Sigma) \models Q$ or (2) $\mathit{cons} \models Q$. If (1) is true, then by Theorem 3 we have that $KB \models_{\mathit{k\text{-}LCons}} Q$. Otherwise, if $\mathit{cons} \models Q$, we have that $\bigcup_{cl \in \mathit{clusters}(KB)} \mathit{chaseInt}(KB, D_I, cl, \mathit{incisions}(cl)) \models Q$. Suppose by contradiction that $KB \not\models_{\mathit{k\text{-}LCons}} Q$; then there exists a lazy repair $\mathit{lrep} \in \mathit{LRep}(k, KB)$ such that $(\mathit{lrep}, \Sigma) \not\models Q$, which means that there is at least one atom $\alpha \in Q$ such that $(\mathit{lrep}, \Sigma) \not\models \alpha$. Since KB is linear, then each α-kernel is a singleton for KB. Therefore, for every $\beta \subseteq D$ such that $(\beta, \Sigma) \models \alpha$, $\beta \in cl$ for some $cl \in \mathit{clusters}(KB)$. Let cl be an arbitrary cluster; since $\mathit{lrep} \not\models \alpha$, by Definition 11, it must the case that there exists $c_{cl} \in \chi_{\mathit{k\text{-}cut}}(cl)$ such that c_{cl} contains every $\beta \in cl$ such that $(\beta, \Sigma) \models \alpha$. Then, we have that $\mathit{chaseInt}(KB, D_I, cl, \mathit{incisions}(cl)) \not\models \alpha$ and this is true for every cluster $cl \in \mathit{clusters}(KB)$. Therefore, we have that $\bigcup_{cl \in \mathit{clusters}(KB)} \mathit{chaseInt}(KB, D_I, cl, \mathit{incisions}(cl)) \not\models Q$, which is a contradiction.

($\Leftarrow$) Second, we show that if $KB \models_{\mathit{LCons}} Q$ then *Yes* is the output of the algorithm.
Suppose by contradiction that the algorithm returns *No*, then for some $\alpha \in Q$, neither $(D_I, \Sigma) \models \alpha$ nor $\bigcup_{cl \in \mathit{clusters}(KB)} \mathit{chaseInt}(KB, D_I, cl, \mathit{incisions}(cl)) \models \alpha$. Clearly, $(D_I, \Sigma) \models \alpha$ then the algorithm would answer *Yes* in line 4, leading to a contradiction. Now, since it holds that $\bigcup_{cl \in \mathit{clusters}(KB)} \mathit{chaseInt}(KB, D_I, cl, \mathit{incisions}(cl)) \not\models \alpha$ then, by Lemma 2 it cannot be the case that α is derived from elements across multiple clusters, and then it must be the case that $\mathit{chaseInt}(KB, D_I, cl, \mathit{incisions}(cl)) \not\models \alpha$ for every cluster cl. Suppose that $\mathit{chase}(cl, \Sigma) \models \alpha$, then $\mathit{chaseInt}(KB, D_I, cl, \mathit{incisions}(cl)) \not\models \alpha$ iff $\mathit{chase}(cl - c_i, \Sigma) \not\models \alpha$ for some $c_i \in \mathit{incisions}(cl) = \chi_{\mathit{k\text{-}cut}}(cl)$. Then, at least one k-cut performed over cl prohibits us from deriving α. This is a contradiction since this would mean that there exists some $\mathit{lrep} \in \mathit{LRep}(k, KB)$ such that for every cluster the corresponding k-cut used prohibits from deriving α, and then $KB \not\models_{\mathit{LCons}} \alpha$ and thus $KB \not\models_{\mathit{LCons}} Q$.

Finally, as shown in Proposition 7, the clusters in a linear Datalog+/– ontology KB can be computed in polynomial time in the data complexity. Furthermore, the incisions for every cluster, the individual chase procedures performed, and the *chaseInt* can also be computed in polynomial time in the size of $|D|$. □

Proof of Lemma 1. It is sufficient to show that the set of all instances of $\mathcal{N}(\Sigma_{NC}, Q)$ relative to $\Delta \cup \Delta_N$ coincides with the set of all instances of Σ_{NC} relative to $\Delta \cup \Delta_N$. Each $\upsilon \in \mathcal{N}(\Sigma_{NC}, Q)$ is obtained from some $\upsilon_o \in \Sigma_{NC}$ by applying a substitution and adding inequalities to the body of υ_o. Each instance υ' of υ with satisfied inequalities is thus also an instance of υ_o. Conversely, each instance υ'_o of $\upsilon_o \in \Sigma_{NC}$ substitutes the variables of υ_o by elements of $\Delta \cup \Delta_N$. Consider then all such inserted elements different from constants in $\Sigma_{NC} \cup \Sigma_T$ and Q as new variables, distinct from each other and the previous constants. This defines some $\upsilon \in \mathcal{N}(\Sigma_{NC}, Q)$, which has υ'_o as an instance. □

Proof of Proposition 9. By Theorem 3, $KB = (D, \Sigma_{NC}) \models_{ICons} Q$ is equivalent to $(D - \bigcup_{c \in culprits(KB)} c) \models Q$. By Lemma 1, the latter is in turn equivalent to $(D - \bigcup_{c \in culprits(KB')} c) \models Q$ with $KB' = (D, \mathcal{N}(\Sigma_{NC}, Q))$. That is, there exists a ground substitution σ such that $\sigma(Q) \subseteq D$ and, for every $c \in culprits(KB')$, it holds that $c \cap \sigma(Q) = \emptyset$. Equivalently, there exists a ground substitution σ such that $\sigma(Q) \subseteq D$ and, for every $\upsilon \in \mathcal{N}(\Sigma_{NC}, Q)$ and ground substitution σ' satisfying all inequalities of υ, if $\sigma'(body(\upsilon))$ is minimal and $\sigma'(body(\upsilon)) \cap \sigma(Q) \neq \emptyset$, then $\sigma'(body(\upsilon)) \not\subseteq D$. Equivalently, $(\star)$ there exists a ground substitution σ such that $\sigma(Q) \subseteq D$ and, for every $\upsilon \in \mathcal{N}(\Sigma_{NC}, Q)$ and ground substitution σ'' satisfying all inequalities of υ, if $\sigma''(\gamma_{C,B}(body(\upsilon)))$ is minimal (under set inclusion) and $\sigma''(\gamma_{C,B}(body(\upsilon))) \cap \sigma(Q) \neq \emptyset$, then $\sigma''(\gamma_{C,B}(body(\upsilon))) \not\subseteq D$, where $\gamma_{C,B}$ is an mgu for some $C \subseteq Q$, $C \neq \emptyset$, and some $B \subseteq body(\upsilon)$. Observe now that the existence of some $\upsilon' \in \mathcal{N}(\Sigma_{NC}, Q)$ such that $body(\upsilon')$ maps isomorphically to some $B' \subset body(\upsilon)$ implies that no ground instance of υ (where all inequalities are satisfied) is minimal. Conversely, if $\sigma''(\gamma_{C,B}(body(\upsilon)))$ is minimal, then no $\upsilon' \in \mathcal{N}(\Sigma_{NC}, Q)$ exists such that $body(\upsilon')$ maps isomorphically via some ι to some $B' \subset body(\upsilon)$, as otherwise it would be the case that $\sigma''(\gamma_{C,B}(\iota(body(\upsilon')))) \subset \sigma''(\gamma_{C,B}(body(\upsilon)))$, and thus $\sigma''(\gamma_{C,B}(body(\upsilon)))$ would not be minimal. Hence, $(\star)$ is equivalent to the existence of a ground substitution σ such that $\sigma(Q) \subseteq D$ and, for every $\upsilon \in \Sigma_Q$ and ground substitution σ'' satisfying all inequalities of υ, if ($\sigma''(\gamma_{C,B}(body(\upsilon)))$ is minimal and) $\sigma''(\gamma_{C,B}(body(\upsilon))) \cap \sigma(Q) \neq \emptyset$, then we have $\sigma''(\gamma_{C,B}(body(\upsilon))) \not\subseteq D$, where $\gamma_{C,B}$ is an mgu for some $C \subseteq Q$, $C \neq \emptyset$, and some $B \subseteq body(\upsilon)$. Equivalently, there exists a ground substitution σ such that $\sigma(Q) \subseteq D$ and, for every $c \in culprits(KB_Q)$ with $KB_Q = (D, \Sigma_Q)$, it holds that $c \cap \sigma(Q) = \emptyset$. The latter is equivalent to $(D - \bigcup_{c \in culprits(KB_Q)} c) \models Q$. By Theorem 3, equivalently, $(D, \Sigma_Q) \models_{ICons} Q$. □

Proof of Theorem 8. By the proof of Proposition 9, $KB = (D, \Sigma_{NC}) \models_{ICons} Q$ iff there exists a ground substitution σ such that $\sigma(Q) \subseteq D$ and, for each $\upsilon \in \Sigma_Q$ and ground substitution σ'' satisfying all inequalities of υ, if $\sigma''(\gamma_{C,B}(body(\upsilon))) \cap \sigma(Q) \neq \emptyset$, then $\sigma''(\gamma_{C,B}(body(\upsilon))) \not\subseteq D$, where $\gamma_{C,B}$ is an mgu for some $C \subseteq Q$, $C \neq \emptyset$, and some $B \subseteq body(\upsilon)$. The latter is equivalent to the existence of a ground substitution σ such that $\sigma(Q) \subseteq D$ and, for each $\upsilon \in \Sigma_Q$ and ground substitution σ'' satisfying all inequalities of υ, if $\sigma(X) = \sigma''(\gamma_{C,B}(X))$ for all $X \in var(C)$, then $\sigma''(\gamma_{C,B}(body(\upsilon))) \not\subseteq D$, where $\gamma_{C,B}$ is an mgu for some $C \subseteq Q$, $C \neq \emptyset$, and some $B \subseteq body(\upsilon)$. This is in turn equivalent to $D \models \mathsf{Enforcement}(Q, \mathcal{N}(\Sigma_{NC}, Q))$. □

Proof of Lemma 3.

($\subseteq$) We first prove that $culprits(KB, \Sigma) \subseteq culprits(KB')$ with $KB' = (D, \Sigma_{Rew})$.

Let $c \in culprits(KB)$, then there exists $\upsilon \in \Sigma_{NC}$ such that $(c, \Sigma_T) \models body(\upsilon)$. Clearly, $\upsilon \in \Sigma_{Rew}$, so either $c \in culprits(KB')$ or there exists $c' \in culprits(KB')$ such that $c' \subset c$, c' unifies with $body(\upsilon')$, and $body(\upsilon')$ maps isomorphically to $B \subset body(\upsilon)$. If $\upsilon' \in \Sigma_{NC}$, then c would not be a culprit. Then it must be the case that $\upsilon' \in \Sigma_{Rew}$ and $\upsilon' \notin \Sigma_{NC}$. Therefore, $body(\upsilon')$ is a rewriting relative to Σ_T of a constraint $\upsilon'' \in \Sigma_{NC}$. Then, it must be the case that $(c', \Sigma_T) \models body(\upsilon'')$, but this is a contradiction since $\upsilon'' \in \Sigma_{NC}$ and therefore c cannot be a culprit for KB.

($\supseteq$) Now, we prove that $culprits(KB') \subseteq culprits(KB, \Sigma)$ with $KB' = (D, \Sigma_{Rew})$. .

Let $c \in culprits(KB')$; then there exists $\upsilon \in \Sigma_{Rew}$ such that $body(\upsilon)$ unifies with c. If $\upsilon \in \Sigma_{NC}$ then $c \in culprits(KB)$ as long as there is no $c' \in culprits(KB)$ such that $c' \subset c$. Suppose there exists such c'; then there exists $\upsilon' \in \Sigma_{NC}$ such that $(c', \Sigma_T) \models body(\upsilon')$, and $body(\upsilon')$ maps isomorphically to $B \subset body(\upsilon)$. However, such υ' must also belong to Σ_{Rew}, in which case c would not be a minimal inconsistent set in (D, Σ_{Rew}), which is a contradiction. If $\upsilon \notin \Sigma_{NC}$, then $body(\upsilon)$ is the rewriting relative to Σ_T of a $body(\upsilon'')$ where $\upsilon'' \in \Sigma_{NC}$. If this is the case, then, υ'' also belongs to Σ_{Rew}, and therefore there must exists c' with $c' \subset c$ such that c' unifies with $body(\upsilon')$. But this is a contradiction since c would not be culprit in $culprits(KB')$. □

Proof of Theorem 9.

($\Rightarrow$) Let $KB \models_{ICons} Q$. Then, by Proposition 10, $(D, \Sigma_{Rew} \cup \Sigma_T) \models_{ICons} Q$. By Theorem 3, this means that $(D - \bigcup_{c \in culprits(KB')} c, \Sigma_T) \models Q$ where $KB' = (D, \Sigma_{Rew} \cup \Sigma_T)$. By the correctness of algorithm $\mathsf{TGD\text{-}rewrite}_{\mathcal{S}}$, we have that for an arbitrary database D', it holds that $(D', \Sigma_T) \models Q$ iff $D' \models Q'$ for some $Q' \in \mathsf{TGD\text{-}rewrite}_{\mathcal{S}}(Q, \Sigma_T)$. Let $D' = D - \bigcup_{c \in culprits(KB')} c$, and we have that $(D - \bigcup_{c \in culprits(KB')} c) \models Q'$. By Lemma 3, we have that $(D - \bigcup_{c \in culprits(KB'')} c) \models Q'$ where $KB'' = (D, \Sigma_{Rew})$, and by Theorem 3, $(D, \Sigma_{Rew}) \models_{ICons} Q'$. By Theorem 8, we get $D \models \mathsf{Enforcement}(Q', \mathcal{N}(\Sigma_{Rew}, Q))$ for some Q'. Since $Q' \in \mathsf{TGD\text{-}rewrite}_{\mathcal{S}}(Q, \Sigma_T)$, it thus follows $D \models \bigvee_{F \in \mathsf{rewriteICons}(Q,\Sigma)} F$.

($\Leftarrow$) Let $D \models \bigvee_{F \in \mathsf{rewriteICons}(Q,\Sigma)} F$. Then, $D \models \mathsf{Enforcement}(Q', \mathcal{N}(\Sigma_{Rew}, Q))$ for some $Q' \in \mathsf{TGD\text{-}rewrite}_{\mathcal{S}}(Q, \Sigma_T)$. Then, by Theorem 8, we have that $(D, \Sigma_{Rew}) \models_{ICons} Q'$, and thus $(D, \Sigma_{Rew} \cup \Sigma_T) \models_{ICons} Q'$. Also, by Proposition 10, we have that $(D, \Sigma_{NC} \cup \Sigma_T) \models_{ICons} Q'$. That is, by Theorem 3, $(D - \bigcup_{c \in culprits(KB)} c) \models Q'$. Given that Q' is a rewriting of Q relative to Σ_T, by the correctness of algorithm $\mathsf{TGD\text{-}rewrite}_{\mathcal{S}}$, we have that, for an arbitrary database D', if $D' \models Q'$ then $(D', \Sigma_T) \models Q'$. Let $D' = D - \bigcup_{c \in culprits(KB)}$. Then, we obtain $(D - \bigcup_{c \in culprits(KB)}, \Sigma_T) \models Q$. Finally, by Theorem 3, we have $KB \models_{ICons} Q$. $\square$