How does incoherence affect inconsistency-tolerant semantics for Datalog$^{\pm}$?

Cristhian A. D. Deagustini[1,2] · **M. Vanina Martinez**[1] · **Marcelo A. Falappa**[1,2] · **Guillermo R. Simari**[1]

Abstract The concept of incoherence naturally arises in ontological settings, specially when integrating knowledge. In the Datalog$^{\pm}$ literature, however, this is an issue that is yet to be studied more deeply. The main focus of our work is to show how classical inconsistency-tolerant semantics for query answering behaves when dealing with atoms that are relevant to unsatisfiable sets of existential rules, which may hamper the quality of answers and any reasoning task based on those semantics. We also propose a notion of incoherency-tolerant semantics for query answering in Datalog$^{\pm}$, and exemplify this notion with a particular semantics based on the transformation of classic Datalog$^{\pm}$ ontologies into defeasible Datalog$^{\pm}$ ones, which use argumentation as its reasoning machinery.

Keywords Incoherence · Inconsistency-tolerant semantics · Argumentation · Datalog$^{\pm}$ ontologies

Mathematics Subject Classification (2010) 68T27 · 68T30 · 68T35 · 68T37

1 Introduction and motivation

In recent times it is usual the development of applications capable to share and reuse data, specially in environments like the Semantic Web, which provide an effective infrastructure for data exchange. At the core of the Semantic Web knowledge is represented by means

✉ Cristhian A. D. Deagustini
caddeagustini@gmail.com

[1] AI R&D Lab., Institute for Computer Science and Engineering (ICIC), Consejo Nacional de Investigaciones Científicas y Técnicas (CONICET), Universidad Nacional del Sur, Alem 1253, (B8000CPB) Bahía Blanca, Argentina

[2] Agents and Intelligent Systems Area, Fac. of Management Sciences, Universidad Nacional de Entre Ríos, (E3202BHR) Concordia, Entre Ríos, Argentina

of ontological languages. These are powerful knowledge representation tools given their decidability and tractability properties which make them attractive for handling practical applications.

In such scenarios a preponderant problem is that of the inconsistencies that can arise in ontologies. There are a number of works addressing this issue in both the Semantic Web and Database Theory communities, and several methods have been developed to deal with it, e.g., [1, 4, 6, 25, 26, 28]. Among the most important semantics developed for querying inconsistent databases, *consistent answers* [1] (or AR semantics in [25] for ontological languages) is the most accepted one. Intuitively, this semantics uses a cautious approach that yields the set of atoms that can be derived despite all possible ways of repairing the inconsistency. A common assumption regarding such semantics is that the intensional knowledge (represented as a set of rules and/or constraints in Datalog$^{\pm}$ or a TBox in Description Logics) correctly models the semantics of the data and this does not change over time. In practice, such assumption have a twofold impact:

(a) First, the set of rules/constraints is always satisfiable, in the sense that their application do not *inevitably* yield a consistency problem.
(b) Second, as a result of the previous observation, it must be the case that the conflicts come from the data contained in the database instance (or ABox) and that is the part of the ontology that must be modified in order to restore consistency.

To consider the set constraints as always satisfiable may be a reasonable assumption to make. This is specially true in the case of a single ontology which in theory could be assumed to be carefully designed and/or to be curated over time. Nevertheless, as noticed by Schlobach and Cornet [36], incoherences do arise in real-world ontologies. An example of such incoherent ontologies is DICE: in such ontology the definition of the "brain" concept is incorrectly specified as both "central nervous-system" and "body-part" located in the head, which is contradictory as nervous-systems and body-parts are declared disjoint in DICE [36]. So, in this work we will consider this more general setting and consider that as knowledge evolves, both data and constraints may change and become conflicting. The problem of conflicts among constraints is known in the Description Logics community as *incoherence* [21, 32, 36]. As we will see later in this work, several of the well-known inconsistency-tolerant semantics for query answering fail at computing good quality answers in the presence of incoherence.

In this paper we focus on a particular family of ontological languages, namely Datalog$^{\pm}$[8]. We show how incoherence can arise in Datalog$^{\pm}$ ontologies, and how the reasoning technique based on the use of defeasible elements in Datalog$^{\pm}$ and an argumentative semantics introduced by Martinez et al. [28] can tolerate such issues, thus resulting in a reasoning machinery suitable of dealing with both incoherent and inconsistent knowledge.

This work integrates three different building blocks:

- First, the notion of incoherence for Datalog$^{\pm}$ ontologies [15] is recalled, which relates to the problem of satisfiability of concepts for Description Logics;
- second, we show how such notion affects most of well-known inconsistency-tolerant semantics which, since they were not designed to confront such issues, can go up to the point of not returning any useful answer;
- finally, we propose a definition for incoherency-tolerant semantics, and introduce an alternative semantics for classic Datalog$^{\pm}$ ontologies based on an argumentative reasoning process over the transformation of classic Datalog$^{\pm}$ ontologies to their correspondent defeasible Datalog$^{\pm}$ ontologies [28]. We show how this semantics behaves in a satisfactory way in the presence of incoherence, as the process can return as

answers atoms that trigger incoherency, which we show that cannot be done by classical inconsistency-tolerant semantics.

2 Preliminaries

First, we briefly recall some basics on Datalog$^{\pm}$[8]. 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 $k \geq 0$. 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 *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.

Given a relational schema $\mathcal{R}$, a *tuple-generating dependency (TGD) σ* is a first-order formula $\forall\mathbf{X}\forall\mathbf{Y}\, \Phi(\mathbf{X}, \mathbf{Y}) \rightarrow \exists\mathbf{Z}\, \Psi(\mathbf{X}, \mathbf{Z})$, 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, respectively. Satisfaction of TGDs are defined via *homomorphisms*, which are mappings $\mu\colon \Delta \cup \Delta_N \cup \mathcal{V} \rightarrow \Delta \cup \Delta_N \cup \mathcal{V}$ 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. Consider a database D for a relational schema $\mathcal{R}$, and a TGD σ on $\mathcal{R}$ of the form $\Upsilon(\mathbf{X}, \mathbf{Y}) \rightarrow \exists\mathbf{Z}\, \Psi(\mathbf{X}, \mathbf{Z})$. Then, σ is *applicable* to D if there exists a homomorphism h that maps the atoms of $\Upsilon(\mathbf{X}, \mathbf{Y})$ to atoms of D. Let σ be applicable to D, and h' be a homomorphism that extends h as follows: for each $X_i \in \mathbf{X}$, $h'(X_i) = h(X_i)$; for each $Z_j \in \mathbf{Z}$, $h'(Z_j) = z_j$, 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 $h'(\Psi(\mathbf{X}, \mathbf{Z}))$ if it is not already in D. After the application we say that σ is satisfied by D. The *Chase* for a database D and a set of TGDs Σ_T, denoted $chase(D, \Sigma_T)$, is the exhaustive application of the TGDs [9] in a breadth-first (level-saturating) fashion, which leads to a (possibly infinite) chase for D and Σ. Since TGDs can be reduced to TGDs with only single atoms in their heads, in the sequel, every TGD has without loss of generalization a single atom in its head.

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. In this work we restrict our attention to atomic queries. 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. 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 $ans(Q, D, \Sigma) \neq \emptyset$. It is important to remark that BCQs Q over D and Σ_T can be evaluated on the chase for D and Σ_T, i.e., $D \cup \Sigma_T \models Q$ is equivalent to $chase(D, \Sigma_T) \models Q$ [9].

Negative constraints (NCs) are first-order formulas of the form $\forall\mathbf{X}\Phi(\mathbf{X}) \rightarrow \bot$, where the body $\mathbf{X}$ is a conjunction of atoms (without nulls) and the head is the truth constant *false*, denoted $\bot$. Intuitively, the head of these constraints have to evaluate to false in D under a set of TGDs Σ_T. That is, an NC τ is satisfied by a database D under a set of TGDs Σ_T iff there not exists a homomorphism h that maps the atoms of $\Phi(\mathbf{X})$ to D, where D is such that every TGD in Σ_T is satisfied. As we will see through the paper, negative constraints are important to identify inconsistencies in a Datalog$^\pm$ ontology, as their violation is one of the main inconsistency sources. In this work we restrict our attention to binary negative constraints (or denial constraints), which are NCs such that their body is the conjunction of exactly two atoms, e.g., $p(X, Y) \wedge q(X, Z) \rightarrow \bot$. As we will show later, this class of constraints suffices for the formalization of the concept of conflicting atoms.

Equality-generating dependencies (EGDs) are first-order formulas of the form $\forall \mathbf{X} \Phi(\mathbf{X}) \rightarrow X_i = X_j$, where $\Phi(\mathbf{X})$ is a conjunction of atoms, and X_i and X_j are variables from $\mathbf{X}$. An EGD σ is satisfied in a database D for $\mathcal{R}$ iff, whenever there exists a homomorphism h such that $h(\Phi(\mathbf{X})) \subseteq D$, it holds that $h(X_i) = h(X_j)$. In this work we will focus on a particular class of EGDs, called *separable* [8]; intuitively, separability of EGDs *w.r.t.* a set of TGDs states that, if an EGD is violated, then atoms contained in D are the reason of the violation (and not the application of TGDs); i.e., if an EGD in Σ_E is violated when we apply the TGDs in Σ_T for a database D, then the EGD is also violated in D. Separability is an standard assumption in Datalog$^\pm$ ontology, as one of the most important features of this family of languages is the focus on decidable [10] (actually tractable) fragments of Datalog$^\pm$. In this work we will adopt this assumption as well and focus only on decidable fragments of the family. EGDs play also an important role in the matter of conflicts in Datalog$^\pm$ ontologies. Note that the restriction of using only separable EGDs makes that certain cases of conflicts are not considered in our proposal; the treatment of such cases, though interesting from a technical point of view, are outside the scope of this work since we focus on tractable fragments of Datalog$^\pm$ as the ones mentioned above. Moreover, as for the case with NCs, we restrict EGDs to binary ones; that is, those which body $\forall \mathbf{X} \Phi(\mathbf{X})$ is such that $\Phi(\mathbf{X})$ is the conjunction of exactly two atoms, e.g., $p(X, Y) \wedge q(X, Z) \rightarrow Y = Z$. In this work we often use the terms constraints and/or dependencies indistinguishable when referring to sets of TGDs, NCs, and EGDs.

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

Datalog$^\pm$ Ontologies A *Datalog$^\pm$ ontology* $KB = (D, \Sigma)$, where $\Sigma = \Sigma_T \cup \Sigma_E \cup \Sigma_{NC}$, consists of a database D, a set of TGDs Σ_T, a set of separable EGDs Σ_E, and a set of negative constraints Σ_{NC}.

Example 1 illustrates a simple Datalog$^\pm$ ontology.

Example 1 Consider the following KB.

$$\left\{\begin{array}{ll}
D: & \{a_1 : \mathit{can\_sing}(\mathit{simone}), a_2 : \mathit{rock\_singer}(\mathit{axl}), \\
& a_3 : \mathit{sing\_loud}(\mathit{ronnie}), a_4 : \mathit{has\_fans}(\mathit{ronnie}), \\
& a_5 : \mathit{manage}(\mathit{band_1}, \mathit{richard}), a_6 : \mathit{sang\_in}(\mathit{axl}, \mathit{band_2}), \\
& a_7 : \mathit{pop\_band}(\mathit{band_2}), a_8 : \mathit{invited\_guest}(\mathit{axl}, \mathit{band_2})\} \\[1ex]
\Sigma_{NC}: & \{\tau_1 : \mathit{sore\_throat}(X) \wedge \mathit{can\_sing}(X) \rightarrow \bot, \\
& \tau_2 : \mathit{unknown}(X) \wedge \mathit{famous}(X) \rightarrow \bot, \\
& \tau_3 : \mathit{pop\_singer}(X) \wedge \mathit{rock\_singer}(X) \rightarrow \bot, \\
& \tau_4 : \mathit{invited\_guest}(X, Y) \wedge \mathit{sang\_in}(X, Y) \rightarrow \bot\} \\[1ex]
\Sigma_E: & \{\nu_1 : \mathit{manage}(X, Y) \wedge \mathit{manage}(X, Z) \rightarrow Y = Z\} \\[1ex]
\Sigma_T: & \{\sigma_1 : \mathit{rock\_singer}(X) \rightarrow \mathit{sing\_loud}(X), \\
& \sigma_2 : \mathit{sing\_loud}(X) \rightarrow \mathit{sore\_throat}(X), \\
& \sigma_3 : \mathit{has\_fans}(X) \rightarrow \mathit{famous}(X), \\
& \sigma_4 : \mathit{rock\_singer}(X) \rightarrow \mathit{can\_sing}(X), \\
& \sigma_5 : \mathit{sang\_in}(X, Y) \wedge \mathit{pop\_band}(Y) \rightarrow \mathit{pop\_singer}(X)\}
\end{array}\right\}$$

Following the classical notion of consistency, we say that a consistent Datalog$^{\pm}$ ontology has a non-empty set of models.

Consistency A Datalog$^{\pm}$ ontology $KB = (D, \Sigma)$ is *consistent* iff $mods(D, \Sigma) \neq \emptyset$. We say that KB is inconsistent otherwise.

3 Incoherence in Datalog$^{\pm}$

The problem of obtaining consistent knowledge from an inconsistent knowledge base is natural in many computer science fields. As knowledge evolves, conflicts and/or contradictions are likely to appear, and these inconsistencies have to be handled in a way such that they do not affect the quality of the information obtained from the knowledge base.

In the setting of Consistent Query Answering (CQA), database repairing, and inconsistency-tolerant query answering in ontological languages [1, 25, 26], often the assumption is made that the set of constraints Σ expresses the semantics of the data in the component D, and as such there is no internal conflict on the set of constraints; furthermore, these constraints are not subject to changes over time. In this paper we will drop such assumption and focus on the relationship between the set of TGDs and the set of NCs and EGDs, as it could happen that (a subset of) the TGDs in Σ_T cannot be applied without always leading to the violation of the NCs or EGDs. This issue is related to that of *unsatisfiability problem of a concept* in an ontology and it is known in the Description Logics community as *incoherence* [21, 32, 36]. Incoherence can be particularly important when combining multiple ontologies since the constraints imposed by each one of them over the data could (potentially) represent conflicting modellings of the application at hand. Clearly, the notions of incoherence and inconsistency are highly related; in fact, Flouris et al. [21] establish a relation between incoherence and inconsistency, considering the former as a special case of the latter.

The notion of incoherence that we use here states that given a set of incoherent constraints $\Sigma_T \subseteq \Sigma$ it is not possible to find a set of atoms D such that $KB = (D, \Sigma)$ is a consistent ontology and at the same time all TGDs in Σ_T are applicable in D [15]. This means that a Datalog$^{\pm}$ ontology KB can be consistent even if the set of constraints is incoherent, as long as the database instance does not make the set of conflicting dependencies applicable. On the other hand, a Datalog$^{\pm}$ ontology KB can be inconsistent even when the set of constraints is coherent. Consider, as an example, the following $KB = (\{tall(peter), small(peter)\}, \{tall(X) \wedge small(X) \rightarrow \bot\})$, where the (empty) set of dependencies is trivially coherent; the ontology is, nevertheless, inconsistent.

In the last decades, several approaches to handling inconsistency were developed in *Artificial Intelligence* and *Database Theory* (e.g., [1, 16, 24]). Some of the best known approaches deal with inconsistency by removing from the theory atoms, or a combination of atoms and constraints or rules. A different approach is to simultaneously consider all possible ways of *repairing* the ontology by deleting or adding atoms, as in most approaches to *Consistent Query Answering* [1] (CQA for short). However, these data-driven approaches might not be adequate for an incoherent theory and may produce meaningless results. As we stated before, an incoherent set of constraints Σ renders inconsistent any ontology whose database instance is such that the incoherent (sub)set of TGDs are applicable; in particular cases, this may lead to the removal (or ignoring) of every single atom in a database instance in an attempt to restore consistency, resulting in an ontology without any valuable information, when it could be the case that it is the set of constraints that is ill defined.

We will now recall from [15] the different concepts needed for the following sections. Before formalizing the notion of *incoherence* that we use in our Datalog$^\pm$ setting we need to identify the set of atoms that are *relevant* to a given set of TGDs. Intuitively, we say that a set of atoms A is relevant to a set T of TGDs if the atoms in the set A are such that the application of T over A generates the atoms that are needed to apply all dependencies in T, i.e., A triggers the application of every TGD in T. Formally, the definition of atom relevancy is as follows:

Definition 1 (Relevant Set of Atoms for a Set of TGDs [15]) Let $\mathcal{R}$ be a relational schema, T be a set of TGDs, and A a (possibly existentially closed) non-empty set of atoms, both over $\mathcal{R}$. We say that A is *relevant* to T iff for all $\sigma \in T$ of the form $\forall \mathbf{X} \forall \mathbf{Y} \Phi(\mathbf{X}, \mathbf{Y}) \rightarrow \exists \mathbf{Z} \Psi(\mathbf{X}, \mathbf{Z})$ it holds that $chase(A, T) \models \exists \mathbf{X} \exists \mathbf{Y} \Phi(\mathbf{X}, \mathbf{Y})$.

When it is clear from the context, if a singleton set $A = \{a\}$ is relevant to $T \subseteq \Sigma_T$ we just say that atom a is relevant to T. The following example illustrates atom relevancy.

Example 2 (Relevant Set of Atoms) Consider the following constraints:

$$\begin{aligned}\Sigma_T = \{&\sigma_1 : plays(X, Y) \rightarrow musician(X),\\ &\sigma_2 : musician(X) \wedge compositor(X) \rightarrow band\_leader(X, B),\\ &\sigma_3 : band(X) \rightarrow plays\_in(X, B)\}\end{aligned}$$

First, let us consider the set $A_1 = \{plays(jordan, keyboard), compositor(jordan), band(dream)\}$. This set is a relevant set of atoms to the set of constraints $\Sigma_T = \{\sigma_1, \sigma_2, \sigma_3\}$, since σ_1 and σ_3 are directly applicable to A_1 and σ_2 becomes applicable when we apply σ_1 (i.e., the chase entails the atom $musician(jordan)$, which together with $compositor(jordan)$ triggers σ_2).

However, the set $A_2 = \{plays(jordan, keyboard), compositor(mike)\}$ is not relevant to Σ_T. Note that even though σ_1 is applicable to A_2, the TGDs σ_2 and σ_3 are never applied in $chase(A_2, \Sigma_T)$. For instance, consider the TGD $\sigma_2 \in \Sigma_T$. In the chase of Σ_T over D we create the atom $musician(jordan)$, but nevertheless we still cannot trigger σ_2 since we do not have and cannot generate the atom $compositor(jordan)$, and the atom $compositor(mike)$ that is already in A_2 does not match the constant value.

We now recall the notion of coherence for Datalog$^\pm$, which adapts others introduced previously for DLs [21, 36]. Our conception of (in)coherence is based on the notion of satisfiability of a set of TGDs *w.r.t.* a set of constraints. Intuitively, a set of dependencies is satisfiable when there is a relevant set of atoms that triggers the application of all dependencies in the set and does not produce the violation of any constraint in $\Sigma_{NC} \cup \Sigma_E$, i.e., the TGDs can be satisfied along with the NCs and EGDs in KB.

Definition 2 ((Satisfiability of a set of TGDs *w.r.t.* a set of constraints [15])) Let $\mathcal{R}$ be a relational schema, $T \subseteq \Sigma_T$ be a set of TGDs, and $N \subseteq \Sigma_{NC} \cup \Sigma_E$, both over $\mathcal{R}$. The set T is *satisfiable w.r.t.* N iff there is a set A of (possibly existentially closed) atoms over $\mathcal{R}$ such that A is relevant to T and $mods(A, T \cup N) \neq \emptyset$. We say that T is *unsatisfiable w.r.t.* N iff T is not satisfiable *w.r.t.* N. Furthermore, Σ_T is satisfiable *w.r.t.* $\Sigma_{NC} \cup \Sigma_E$ iff there is no $T \subseteq \Sigma_T$ such that T is unsatisfiable *w.r.t.* some N with $N \subseteq \Sigma_{NC} \cup \Sigma_E$.

In the rest of the paper sometimes we write that a set of TGDs is (un)satisfiable omitting the set of constraints, we do this in the context of a particular ontology where we have a fixed set of constraints $\Sigma_{NC} \cup \Sigma_E$. Also, through the paper we denote by $\mathcal{U}(KB)$ the set of minimal unsatisfiable sets of TGDs in Σ_T for KB (i.e., unsatisfiable set of TGDs such that every proper subset of it is satisfiable). The following example illustrates the concept of satisfiability of a set of TGDs in a Datalog$^{\pm}$ ontology

Example 3 (Unsatisfiable sets of dependencies) Consider the following sets of constraints.

$$\Sigma^1_{NC} = \{\tau : \mathit{drives\_car}(P) \wedge \mathit{teenager}(P) \rightarrow \bot\}$$

$$\Sigma^1_T = \{\sigma_1 : \mathit{taxist}(P) \rightarrow \mathit{drives\_car}(P), \sigma_2 : \mathit{in\_high\_school}(P) \rightarrow \mathit{teenager}(P)\}$$

The set Σ^1_T is a satisfiable set of TGDs, and even though the simultaneous application of σ_1 and σ_2 may violate $\tau \in \Sigma^1_{NC}$, that does not hold for every relevant set of atoms. Consider as an example the relevant set $D_1 = \{\mathit{taxist}(\mathit{travis}), \mathit{in\_high\_school}(\mathit{rust})\}$; D_1 is a relevant set for Σ^1_T, however, as we have that $\mathit{mods}(D_1, \Sigma^1_T \cup \Sigma^1_{NC} \cup \Sigma^1_E) \neq \emptyset$ then Σ^1_T is satisfiable.

On the other hand, as an example of unsatisfiability consider the following constraints:

$$\begin{aligned}\Sigma^2_{NC} &= \{\tau_1 : \mathit{sore\_throat}(X) \wedge \mathit{can\_sing}(X) \rightarrow \bot\}\\ \Sigma^2_T &= \{\sigma_1 : \mathit{rock\_singer}(X) \rightarrow \mathit{sing\_loud}(X),\\ &\quad\ \sigma_2 : \mathit{sing\_loud}(X) \rightarrow \mathit{sore\_throat}(X),\\ &\quad\ \sigma_3 : \mathit{rock\_singer}(X) \rightarrow \mathit{can\_sing}(X)\}\end{aligned}$$

The set Σ^2_T is an unsatisfiable set of dependencies, as the application of TGDs $\{\sigma_1, \sigma_2, \sigma_3\}$ on any relevant set of atoms will cause the violation of τ_1. For instance, consider the relevant atom $\mathit{rock\_singer}(\mathit{axl})$: we have that the application of Σ^2_T over $\{\mathit{rock\_singer}(\mathit{axl})\}$ causes the violation of τ_1 when considered together with Σ^2_T, therefore $\mathit{mods}(\{\mathit{rock\_singer}(\mathit{axl})\}, \Sigma^2_T \cup \Sigma^2_{NC} \cup \Sigma^2_E) = \emptyset$. Note that *any* set of relevant atoms will cause the violation of τ_1.

With the presented elements we can have a formal definition of coherence for a Datalog$^{\pm}$ ontology. Intuitively, an ontology is coherent if there is no subset of the set of TGDs that is unsatisfiable *w.r.t.* the constraints in the ontology.

Definition 3 (Coherence [15]) Let $KB = (D, \Sigma)$ be a Datalog$^{\pm}$ ontology defined over a relational schema $\mathcal{R}$, and $\Sigma = \Sigma_T \cup \Sigma_E \cup \Sigma_{NC}$, where Σ_T is a set of TGDs, Σ_E a set of separable EGDs and Σ_{NC} a set of negative constraints. KB is *coherent* iff Σ_T is satisfiable *w.r.t.* $\Sigma_{NC} \cup \Sigma_E$. Also, KB is said to be incoherent iff it is not coherent.

Example 4 (Coherence) Consider the sets of dependencies and constraints defined in Example 3 and an arbitrary database instance D. Clearly, the Datalog$^{\pm}$ ontology $\mathrm{KB}^1 = (D, \Sigma^1_T \cup \Sigma^1_{NC} \cup \Sigma^1_E)$ is coherent, while $\mathrm{KB}^2 = (D, \Sigma^2_T \cup \Sigma^2_{NC} \cup \Sigma^2_E)$ is incoherent.

Finally, we recall the relation between incoherence and inconsistency [15]. Looking into Definitions 2 and 3 we can infer that an incoherent KB will induce an inconsistent KB when the database instance contains any set of atoms that is relevant to the unsatisfiable sets of TGDs. This result is captured in the following proposition.

Proposition 1 ([15]) *Let $KB = (D, \Sigma)$ be a Datalog$^{\pm}$ ontology where $\Sigma = \Sigma_T \cup \Sigma_E \cup \Sigma_{NC}$. If KB is incoherent and there exists $A \subseteq D$ such that A is relevant to some unsatisfiable set $U \in \mathcal{U}(KB)$ then $KB = (D, \Sigma)$ is inconsistent.*

Example 5 (Relating Incoherence and Inconsistency) As an instance of the relationship expressed in Proposition 1, consider once again the ontology presented in Example 1. As hinted previously in Example 3, there we have the set $A \subset D = \{rock\_singer(axl)\}$ and the unsatisfiable set of TGDs $U \subset \Sigma_T = \{\sigma_1 : rock\_singer(X) \rightarrow sing\_loud(X), \sigma_2 : sing\_loud(X) \rightarrow sore\_throat(X), \sigma_4 : rock\_singer(X) \rightarrow can\_sing(X)\}$. Since A is relevant to U the conditions in Proposition 1 are fulfilled, and indeed the ontology $KB = (D, \Sigma)$ from Example 1 is inconsistent since $\tau_1 \in \Sigma_T$ is violated.

4 Incoherence influence on classic inconsistency-tolerant semantics

In the previous section we have established the relationship between incoherence and inconsistency. As explained, classic inconsistency-tolerant techniques do not account (by design) for coherence issues. Nevertheless, if we consider that both components in the ontology evolve (perhaps being collaboratively maintained by a pool of users) then certainly incoherence is prone to arise. In the following we show that it may be important for query answering semantics to consider incoherence in ontologies besides inconsistency, since if not treated appropriately an incoherent set of TGDs may lead to the trivial solution of removing/ignoring every single relevant atom in D (which in the worst case could be the entire database instance). While this may be adequate for some particular domains, it does not seem to be a desirable outcome in the general case.

Although classical query answering in Datalog$^{\pm}$ is not tolerant to inconsistency issues, a variety of inconsistency-tolerant semantics have been developed in the last decade for ontological languages, including lightweight Description Logics (DLs), such as $\mathcal{EL}$ and *DL-Lite* [5, 25], and several fragments of Datalog$^{\pm}$[26]. In this section we analyze the influence of incoherence in several inconsistency-tolerant semantics for ontological languages: AR semantics [25], CAR semantics [25], and provide some insights for sound approximations of AR and of CAR. We present the basic concepts needed to understand the different semantics for query answering on Datalog$^{\pm}$ ontologies and then show how entailment under such semantics behaves in the presence of incoherence. The notion of *repair* in relational databases is a model of the set of integrity constraints that is maximally close, i.e., *"as close as possible"* to the original database.

Depending on how repairs are obtained we can have different semantics. In the following we recall AR-semantics [25], one of the most widely accepted inconsistency-tolerant semantics, along with an alternative to AR called CAR-semantics.

AR semantics The AR semantics corresponds to the notion of *consistent answers* in relational databases [1]. Intuitively, an atom a is said to be AR-consistently entailed from a Datalog$^{\pm}$ ontology KB, denoted $KB \models_{AR} a$ iff a is classically entailed from every ontology that can be built from every possible A-box repair (a maximally consistent subset of the D component that after its application to Σ_T respects every constraint in $\Sigma_E \cup \Sigma_{NC}$).

We denote by $KB \not\models_{AR} a$ the fact that a cannot be AR-consistently inferred from KB. We extend entailment to set of atoms straightforwardly, i.e., for a set of atoms A it holds that $KB \models_{AR} A$ iff for every $a \in A$ it holds that $KB \models_{AR} a$, and $KB \not\models_{AR} A$ otherwise.

CAR semantics As noted by Lembo et al. [25], the *AR* semantics is *not independent* from the form of the knowledge base; it is easy to show that given two inconsistent knowledge bases that are logically equivalent, contrary to what one would expect, their respective repairs do not coincide. To address this, another definition of repairs was also proposed by Lembo et al. [25] that includes knowledge that comes from the closure of the database instance with respect to the set of TGDs. Since the closure of an inconsistent ontology yields the whole language, they define the *consistent closure* of an ontology $KB = (D, \Sigma_T \cup \Sigma_E \cup \Sigma_{NC})$ as the set $CCL(KB) = \{\alpha \mid \alpha \in \mathcal{H}(\mathcal{L_R})\, s.t.\ \exists S \subseteq D$ and $mods(S, \Sigma_T \cup \Sigma_E \cup \Sigma_{NC}) \neq \emptyset$ and $(S, \Sigma_T) \models \alpha\}$. A *Closed ABox repair* of a Datalog$^\pm$ ontology KB is a consistent subset D' of $CCL(KB)$ such that it maximally preserves the database instance [25]. It is said that an atom a is CAR-consistently entailed from a Datalog$^\pm$ ontology KB, denoted by $KB \models_{CAR} a$ iff a is classically entailed from every ontology built from each possible closed ABox repair. We extend entailment to set of atoms straightforwardly, i.e., for a set of atoms A it holds that $KB \models_{CAR} A$ iff for every $a \in A$ it holds that $KB \models_{CAR} a$, and $KB \not\models_{CAR} A$ otherwise.

Incoherence has great influence when calculating repairs, as can be seen in the following result: independently of the query answering semantics (i.e., AR or CAR) no atom that is relevant to an unsatisfiable set of TGDs could belong to a repair of an incoherent KB.

Lemma 1 *Let $KB = (D, \Sigma)$ be an incoherent Datalog$^\pm$ ontology where $\Sigma = \Sigma_T \cup \Sigma_E \cup \Sigma_{NC}$ and $\mathcal{R}(KB)$ be the set of (A-Box or Closed A-Box) repairs of KB. If $A \subseteq D$ is relevant to some unsatisfiable set $U \in \mathcal{U}(KB)$ then $A \nsubseteq R$ for every $R \in \mathcal{R}(KB)$.*

The proof of Lemma 1 follows from Proposition 1, since any set of atoms relevant to an unsatisfiable set of TGDs will be conflictive with $\Sigma_{NC} \cup \Sigma_E$, thus not qualifying to be part of a proper repair.

Proof Let $A \subseteq D$ be a set of atoms relevant to some unsatisfiable set $U \in \mathcal{U}(KB)$. Suppose by absurd that there exists an arbitrary $R \in \mathcal{R}(KB)$ such that $A \subseteq R$.

Since A is relevant to U then from Proposition 1 we have that $mods(D, \Sigma) = \emptyset$, and it easy to show that $mods(A, \Sigma) = \emptyset$. However, since $R \in \mathcal{R}(KB)$ then by the definition of repairs we have that $mods(R, \Sigma) \neq \emptyset$, and as $A \subseteq R$ then $mods(A, \Sigma) \neq \emptyset$.

Then, we have that $mods(A, \Sigma) = \emptyset$ and $mods(A, \Sigma) \neq \emptyset$, an absurd coming from our initial supposition that there exists an arbitrary $R \in \mathcal{R}(KB)$ such that $A \subseteq R$, and it holds that if $A \subseteq D$ is relevant to some unsatisfiable set $U \in \mathcal{U}(KB)$ then $A \nsubseteq R$ for every $R \in \mathcal{R}(KB)$. □

Example 6 Consider the atom $rock\_singer(axl)$ from the ontology presented in Example 1. As we have explained in Example 5, such atom is relevant to $U \subset \Sigma_T = \{\sigma_1 : rock\_singer(X) \rightarrow sing\_loud(X), \sigma_2 : sing\_loud(X) \rightarrow sore\_throat(X), \sigma_4 : rock\_singer(X) \rightarrow can\_sing(X)\}$.

It is easy to show that as a result of this the atom does not belong to any A-Box or Closed A-Box repair. Consider the case of A-Box repairs. We have that they are maximally *consistent* subsets of the component D. We have that $mods(rock\_singer(axl), \Sigma) = \emptyset$, as the NC $\tau_1 : sore\_throat(X) \wedge can\_sing(X) \rightarrow \bot$ is violated. Moreover, clearly this violation happens for every set $A \subseteq D$ such that $rock\_singer(axl) \in A$, and thus we have that $mods(A, \Sigma) = \emptyset$, i.e., $rock\_singer(axl)$ cannot be part of any A-Box repair for the KB.

Analogously, we can show that for any $D' \subseteq D$ such that $mods(D', \Sigma) \neq \emptyset$ it holds that $(D', \Sigma_T) \not\models rock\_singer(axl)$, and thus it holds that $rock\_singer(axl) \notin CCL(KB)$. Then, since Closed A-Box repairs are subsets of $CCL(KB)$ it cannot happen that $rock\_singer(axl)$ belongs to any of these repairs.

Then, we can extend the result in Lemma 1 and say that every atom that is relevant to an unsatisfiable set of TGDs cannot be AR-consistently (resp, CAR-consistently) entailed.

Proposition 2 *Let $KB = (D, \Sigma)$ be an incoherent Datalog$^{\pm}$ ontology where $\Sigma = \Sigma_T \cup \Sigma_E \cup \Sigma_{NC}$. If $A \subseteq D$ is relevant to some unsatisfiable set $U \subseteq \Sigma_T$ then $KB \nvDash_{AR} A$ and $KB \nvDash_{CAR} A$.*

Proof In this proof we focus on AR semantics, disregarding the proof for CAR as it is analogous.

Consider $A \subseteq D$ such that it is relevant to some unsatisfiable set $U \subseteq \Sigma_T$, and any arbitrary $R \in \mathcal{R}(KB)$. Since A is relevant to U then from Proposition 1 follows that $mods(A, \Sigma) = \emptyset$. However, since $R \in \mathcal{R}(KB)$ then by the definition of repairs we have that $mods(R, \Sigma) \neq \emptyset$. Since $mods(A, \Sigma) = \emptyset$ and $mods(R, \Sigma) \neq \emptyset$ we can conclude that $R \nvDash A$. Then, since there exists a repair that does not entail A we have that $\bigcap_{R \in \mathcal{R}(KB)} R \nvDash A$, and thus $KB \nvDash_{AR} A$. □

As a corollary, in the limit case that every atom in the database instance is relevant to some unsatisfiable subset of the TGDs in the ontology then the set of AR-answers (resp, CAR-answers) is empty.

Corollary 1 *Let $KB = (D, \Sigma)$ be an incoherent Datalog$^{\pm}$ ontology where $\Sigma = \Sigma_T \cup \Sigma_E \cup \Sigma_{NC}$, and let $\mathcal{A}_{AR}$ and $\mathcal{A}_{CAR}$ be the set of atoms AR-consistently and CAR-consistenly entailed from KB, respectively. If for every $a \in D$ there exists some $U \in \mathcal{U}(KB)$ such that a is relevant to U then $\mathcal{A}_{AR} = \emptyset$ and $\mathcal{A}_{CAR} = \emptyset$.*

Proof Since for every $a \in D$ there exists some $U \in \mathcal{U}(KB)$ such that a is relevant to U, then from Lemma 1 we have that $\mathcal{R}(KB) = \emptyset$. Thus, since the set of repairs are empty then nothing could be inferred from them and we have that $\mathcal{A}_{AR} = \emptyset$ and $\mathcal{A}_{CAR} = \emptyset$. □

Example 7 Consider the KB in Example 1, and the atom $a_2 : rock\_singer(axl)$ in D. Atom a_2 is relevant to the unsatisfiable set $U \subset \Sigma_T = \{\sigma_1 : rock\_singer(X) \rightarrow sing\_loud(X), \sigma_2 : sing\_loud(X) \rightarrow sore\_throat(X), \sigma_4 : rock\_singer(X) \rightarrow can\_sing(X)\}$, and indeed it holds that $KB \nvDash_{AR} rock\_singer(axl)$ and $KB \nvDash_{CAR} rock\_singer(axl)$. As explained in Example 6, this is because $rock\_singer(axl)$ cannot belong to any repair since its consistent application to Σ is not feasible, i.e., $mods((rock\_singer(axl), \Sigma)) = \emptyset$.

The results shown in Proposition 2 and Corollary 1 also hold for other inconsistency-tolerant semantics based on repairs, such as IAR, $ICAR$ and ICR [25]. The reason behind this is that the IAR and ICR semantics are sound approximations of the AR semantics, and the $ICAR$ semantics is a sound approximation of the CAR semantics. Then, there is no answer that could be provided by IAR, ICR, and $ICAR$ that is not given by AR or IAR. Thus, if no set of atoms relevant to unsatisfiable sets of TGDs could be entailed from the latter semantics, then certainly they could not be entailed by the former; therefore, the results shown hold also for IAR, $ICAR$, and ICR.

5 Incoherency-tolerant semantics

We have shown how incoherence affects classic inconsistency-tolerant semantics up to the point of not returning any meaningful answer (since they were not develop to consider such kind of issues). In this section we propose the notion of tolerance to incoherence for query answering semantics. Such semantics will allow to be able to obtain useful answers from incoherent ontologies. We continue this section by showing an alternative semantics for Datalog$^\pm$ based on the use of argumentative inference that is tolerant to incoherence. For the elements of argumentation we refer the reader to [3, 33].

Definition 4 (Incoherence-tolerant semantics) A query answering semantics S is said to be *tolerant to incoherence* (or incoherency-tolerant) iff for every Datalog$^\pm$ ontology $KB = (D, \Sigma)$ where $\Sigma = \Sigma_T \cup \Sigma_E \cup \Sigma_{NC}$, there exists $A \subseteq D$ and $U \in \mathcal{U}(KB)$ such that A is relevant to U and it holds that $KB \models_S A$.

Intuitively, a query answering semantics is tolerant to incoherence if it can entail atoms that trigger incoherent sets of TGDs as answers. Clearly, from Proposition 2 it follows that inconsistency-tolerant semantics based on repairs are not tolerant to incoherence.

Observation 1 *AR and CAR semantics are not incoherency-tolerant semantics.*

5.1 An incoherency-tolerant semantics via argumentative inference

We begin by recalling Defeasible Datalog$^\pm$(for the interested reader, a more complete presentation of the framework can be found in [28]), and then we move on to show the behaviour of this semantics in the presence of incoherence.

Defeasible Datalog$^\pm$[28] is a variation of Datalog$^\pm$ that enables argumentative reasoning in Datalog$^\pm$ by means of transforming the information encoded in a KB to represent statements whose acceptance can be challenged. To do this, a Datalog$^\pm$ ontology is extended with a set of *defeasible atoms* and *defeasible TGDs*; thus, a Defeasible Datalog$^\pm$ ontology contains both (classical) strict knowledge and defeasible knowledge. The set of defeasible TGDs allows to express weaker connections between pieces of information than in a classical TGDs. *Defeasible TGDs* are rules of the form $\Upsilon(\mathbf{X}, \mathbf{Y}) \succ\!\!- \exists \mathbf{Z}\, \Psi(\mathbf{X}, \mathbf{Z})$, where $\Upsilon(\mathbf{X}, \mathbf{Y})$ and $\Psi(\mathbf{X}, \mathbf{Z})$ are conjunctions of atoms. As in DeLP's defeasible rules [22], defeasible TGDs are used to represent weaker connections between the body and the head of a rule. Defeasible TGDs are written using the symbol "$\succ\!\!-$", while the classical (right) arrow "$\rightarrow$" is reserved to *strict* TGDs and NCs.

Defeasible Datalog$^\pm$ Ontologies. A *defeasible* Datalog$^\pm$ *ontology* KB consists of a finite set F of *ground atoms*, called *facts*, a finite set D of *defeasible atoms*, a finite set of TGDs Σ_T, a finite set of defeasible TGDs Σ_D, and a finite set of binary constraints $\Sigma_E \cup \Sigma_{NC}$.

The following example shows a defeasible Datalog$^\pm$ ontology that encodes the knowledge from Example 1 changing some of the facts and TGDs to defeasible ones.

Example 8 The information from the ontology presented in Example 1 can be better represented by the following defeasible Datalog$^\pm$ ontology $KB = (F, D, \Sigma_T', \Sigma_D, \Sigma_{NC})$, where $F = \{can\_sing(simone), rock\_singer(axl), sing\_loud(ronnie), has\_fans(ronnie)\}$ and $D = \{manage(band_1, richard)\}$. Note that we have changed the fact stating that *richard* manages $band_1$ to a defeasible one, since it could be the case that we heard reports that indicate that the members of $band_1$ are looking for a new manager. The sets of TGDs, and defeasible TGDs are now given by the following sets; note that we have changed some of the TGDs into defeasible TGDs to make clear that the connection between the head and body is weaker.

$$\begin{aligned}\Sigma_{T'} &= \{sing\_loud(X) \rightarrow sore\_throat(X),\\ &\qquad rock\_singer(X) \rightarrow can\_sing(X)\\ \Sigma_D &= \{rock\_singer(X) \succ sing\_loud(X),\\ &\qquad has\_fans(X) \succ famous(X)\}\end{aligned}$$

Derivations from a defeasible Datalog$^\pm$ ontology rely in the application of (strict or defeasible) TGDs. Given a defeasible Datalog$^\pm$ ontology $KB = (F, D, \Sigma_T, \Sigma_D, \Sigma_{NC})$, a (strict or defeasible) TGD σ is applicable if there exist a homomorphism mapping the atoms in the body of σ into $F \cup D$. The *application of* σ on KB generates a new atom from the head of σ if it is not already in $F \cup D$, in the same way as explained in the preliminaries of this work.

The following definitions follow similar ones first introduced by Martinez et al. [29]. Here we adapt the notions to defeasible Datalog$^\pm$ ontologies. An atom has a derivation from a KB iff there is a finite sequence of applications of (strict or defeasible) TGDs that has the atom as its last component.

Definition 5 Let $KB = (F, D, \Sigma_T, \Sigma_D, \Sigma_{NC})$ be a defeasible Datalog$^\pm$ ontology and L an atom. An *annotated derivation* ∂ of L from KB consists of a finite sequence $[R_1, R_2, \ldots, R_n]$ such that R_n is L, and each atom R_i is either: (*i*) R_i is a fact or defeasible atom, i.e., $R_i \in F \cup D$, or (*ii*) there exists a TGD $\sigma \in \Sigma_T \cup \Sigma_D$ and a homomorphism h such that $h(head(\sigma)) = R_i$ and σ is applicable to the set of all atoms and defeasible atoms that appear before R_i in the sequence. When no defeasible atoms and no defeasible TGDs are used in a derivation, we say the derivation is a *strict derivation*, otherwise it is a *defeasible derivation*.

Note that there is non-determinism in the order in which the elements in a derivation appear; TGDs (strict and defeasible) can be reordered, and facts and defeasible atoms could be added at any point in the sequence before they are needed to satisfy the body of a TGD. These syntactically distinct derivations are, however, equivalent for our purposes. It is possible to introduce a canonical form for representing them and adopt that canonical form as the representative of all of them. For instance, we might endow the elements of the program from which the derivation is produced with a total order; thus, it is possible to select one derivation from the set of all the derivations of a given literal that involve the same elements by lexicographically ordering these sequences. When no confusion is possible, we assume that a unique selection has been made.

We say that an atom a is strictly derived from KB iff there exists a strict derivation for a from KB, denoted with $KB \vdash a$, and a is defeasibly derived from KB iff there exists a defeasible derivation for a from KB and no strict derivation exists, denoted with $KB \mid\!\sim a$. A derivation ∂ for a is *minimal* if no proper sub-derivation ∂' of ∂ (every member of ∂' is a member of ∂) is also an annotated derivation of a. Considering minimal derivations in a defeasible derivation avoids the insertion of unnecessary elements that will weaken its ability to support the conclusion by possibly introducing unnecessary points of conflict. Given a derivation ∂ for a, there exists at least one minimal sub-derivation $\partial' \subseteq \partial$ for an atom a. Thus, through the paper we only consider minimal derivations [28].

Example 9 From the defeasible Datalog$^{\pm}$ ontology in Example 8, we can get the following (minimal) annotated derivation for atom $\mathit{sore\_throat}(axl)$:

$$\begin{aligned} \partial = \big[&\mathit{rock\_singer}(axl),\\ \mathit{rock\_singer}(X) \succ\!\!- &\mathit{sing\_loud}(X),\\ &\mathit{sing\_loud}(axl),\\ \mathit{sing\_loud}(X) \rightarrow &\mathit{sore\_throat}(X),\\ &\mathit{sore\_throat}(axl)\big] \end{aligned}$$

Then, we have that $KB \vdash \mathit{rock\_singer}(axl)$ and that $KB \mathrel{|\!\!\sim} \mathit{sore\_throat}(axl)$.

Classical query answering in defeasible Datalog$^{\pm}$ ontologies is equivalent to query answering in Datalog$^{\pm}$ ontologies. The next proposition shows that every atom that is defeasibly or strictly entailed by the defeasible ontology is classically entailed by a classical Datalog$^{\pm}$ ontology that contains a strict version of the database instance and constraints of the defeasible one. This proposition was previously only stated in [28], in this work we show the formal proof.

Proposition 3 ([28]) *Let L be a ground atom, $KB = (F, D, \Sigma_T, \Sigma_D, \Sigma_{NC})$ be a defeasible Datalog$^{\pm}$ ontology, $KB' = (F \cup D, \Sigma'_T \cup \Sigma_{NC})$ is a classical Datalog$^{\pm}$ ontology where $\Sigma'_T = \Sigma_T \cup \{\Upsilon(\mathbf{X}, \mathbf{Y}) \rightarrow \exists \mathbf{Z}\, \Psi(\mathbf{X}, \mathbf{Z}) \mid \Upsilon(\mathbf{X}, \mathbf{Y}) \succ\!\!- \exists \mathbf{Z}\, \Psi(\mathbf{X}, \mathbf{Z})\}$. Then, $KB' \models L$ iff $KB \vdash L$ or $KB \mathrel{|\!\!\sim} L$.*

Proof The proof relies on the fact that if we have a (strict or defeasible) derivation for a literal then the application of those TGDs over the atoms generates the literal, and viceversa.

$\Rightarrow$) Let L be such that $KB' \models L$. This means that $\mathit{chase}(D, \Sigma'_T) \models L$, and thus there exists a set $S' \subseteq \Sigma'_T$ such that the application of TGDs infers L, i.e., $S' \models L$.

When considering the defeasible Datalog$^{\pm}$ ontology KB that originated KB' we can have two different scenarios, either

(a) all TGDs in the set $S \subseteq \Sigma_T$ are classical TGDs, or
(b) at least a TGD in S is a defeasible TGD.

Let us consider these two cases separately.

(a) if all TGDs in S are classical, since we have that $S' \models L$ then we have a finite sequence $[R_1, R_2, \ldots, R_n]$ where R_i is a TGD in S or an atom (either strict or defeasible) and $R_n = L$. Then, if there is no R_i such that it is a defeasible atom then $KB \vdash L$, and otherwise $KB \mathrel{|\!\!\sim} L$.
(b) if at least one TGD in S is defeasible, since we have that $S' \models L$ then we have a finite sequence $[R_1, R_2, \ldots, R_n]$ where R_i is a strict or defeasible TGD in S or an atom and $R_n = L$. Then, we have that $KB \mathrel{|\!\!\sim} L$.

$\Leftarrow$) Now, consider KB such that $KB \vdash L$ or $KB \mathrel{|\!\!\sim} L$. Then, there exists a finite sequence $[R_1, R_2, \ldots, R_n]$ where R_i is a strict or defeasible TGD in S or an atom (either strict or defeasible) and $R_n = L$.

Let $KB' = (F \cup D, \Sigma'_T \cup \Sigma_{NC})$ be a classical Datalog$^{\pm}$ ontology where $\Sigma'_T = \Sigma_T \cup \{\Upsilon(\mathbf{X},\mathbf{Y}) \rightarrow \exists \mathbf{Z}\, \Psi(\mathbf{X},\mathbf{Z}) \mid \Upsilon(\mathbf{X},\mathbf{Y}) \succ\!\!- \exists \mathbf{Z}\, \Psi(\mathbf{X},\mathbf{Z})\}$. Then, there exist a sequence $[R'_1, R'_2, \ldots, R'_n]$ where

- every R_i in $[R_1, R_2, \ldots, R_n]$ that is a strict TGD is such that $R_i = R'_i$,
- for every defeasible TGD $R_i = \Upsilon(\mathbf{X},\mathbf{Y}) \succ\!\!- \exists \mathbf{Z}\, \Psi(\mathbf{X},\mathbf{Z})$ we have that there exists $R'_i = \Upsilon(\mathbf{X},\mathbf{Y}) \rightarrow \exists \mathbf{Z}\, \Psi(\mathbf{X},\mathbf{Z})$,
- every R_i in $[R_1, R_2, \ldots, R_n]$ that is an atom is such that $R_i = R'_i$,
- $R_n = L$.

Then, we have that $[R'_1, R'_2, \ldots, R'_n]$ is a derivation for L. Let S' be the set of every R'_i such that $R'i$ is a TGD, and D' the set of every R_i such that R_i is an atom. Clearly we have that $S' \subseteq \Sigma'_T$ and $D' \subseteq D$. Then, $chase(D, \Sigma'_T) \models L$, and it holds that $KB' \models L$. □

Proposition 3 states the equivalence between derivations from defeasible Datalog$^{\pm}$ ontologies and entailment in traditional Datalog$^{\pm}$ ontologies whose database instance corresponds to the union of facts and defeasible atoms, and the set of TGDs corresponds to the union of the TGDs and the strict version of the defeasible TGDs. As a direct consequence, all the existing work done for Datalog$^{\pm}$ directly applies to defeasible Datalog$^{\pm}$. In particular, it is easy to specify a defeasible *chase* procedure over defeasible Datalog$^{\pm}$ ontologies, based on the revised notion of application of (defeasible) TGDs, whose result is an *universal model*. Therefore, a (B)CQ Q over a defeasible Datalog$^{\pm}$ ontology can be evaluated by verifying that Q is a classical consequence of the chase obtained from the defeasible Datalog$^{\pm}$ ontology.

5.1.1 Argumentation-based reasoning in defeasible Datalog$^{\pm}$

Conflicts in defeasible Datalog$^{\pm}$ ontologies come, as in classical Datalog$^{\pm}$, from the violation of NCs or EGDs. Intuitively, two atoms are in conflict relative to a defeasible Datalog$^{\pm}$ ontology whenever they are both derived from the ontology (either strictly or defeasible) and together map to the body of a negative constraint or they violate an equality-generating dependency.

Definition 6 Given a set of NCs Σ_{NC} and a set of non-conflicting EGDs Σ_E, two ground atoms (possibly with nulls) a and b are said to be *in conflict* relative to $\Sigma_E \cup \Sigma_{NC}$ iff there exists an homomorphism h such that $h(body(\upsilon)) = a \wedge b$ for some $\upsilon \in \Sigma_{NC}$ or $h(X_i) \neq h(Y_j)$ for some $\nu \in \Sigma_E$ where $h(X_i)$ is a term in a and $h(Y_j)$ is a term in b.

In what follows, we say that a set of atoms is a *conflicting* set of atoms relative to $\Sigma_E \cup \Sigma_{NC}$ if and only if there exist at least two atoms in the set that are in conflict relative to $\Sigma_E \cup \Sigma_{NC}$, otherwise will be called *non-conflicting*. Whenever is clear from the context we omit the set of NCs and EGDs.

Example 10 Consider the NC $\{sore\_throat(X) \wedge can\_sing(X) \rightarrow \bot\}$ in Σ_{NC} from the defeasible ontology in Example 8. In this case, the set of atoms $\{sore\_throat(axl), can\_sing(axl)\}$ is a conflicting set relative to Σ_{NC}. However, this is not the case for the set $S = \{rock\_singer(axl)\}$: even when such set generates a violation when applied to the set of TGDs, it is not conflicting in itself.

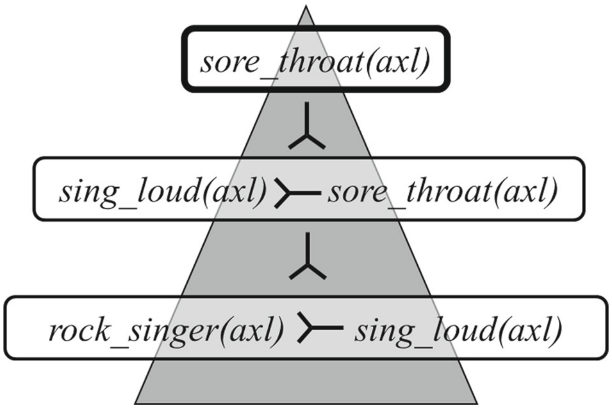
sore_throat(axl)
sing_loud(axl) ≻— sore_throat(axl)
rock_singer(axl) ≻— sing_loud(axl)

Fig. 1 An argument for *sore_throat*(*axl*)

Whenever defeasible derivations of conflicting atoms exist, we use a dialectical process to decide which information prevails, i.e., which piece of information is such that no acceptable reasons can be put forward against it. Reasons are supported by arguments; an argument is an structure that supports a claim from evidence through the use of a reasoning mechanism. We maintain the intuition that led to the classic definition of arguments by Simari and Loui [37], as shown in the following definition.

Definition 7 Let KB be a defeasible Datalog$^{\pm}$ontology and L a ground atom. A set $\mathcal{A}$ of facts, defeasible atoms, TGDs, and defeasible TGDs used in an annotated derivation ∂ of L is an *argument for L constructed from KB* iff ∂ is a $\subseteq$-minimal derivation and no conflicting atoms can be defeasible derived from $\mathcal{A} \cup \Sigma_T$. An argument $\mathcal{A}$ for L is denoted $\langle \mathcal{A}, L \rangle$, and $\mathbb{A}_{KB}$ will be the set of all arguments that can be built from KB.

Example 11 Consider the derivation ∂ in Example 9; there, we have that $\langle$*sore_throat*(*axl*), $\partial\rangle$ is an argument in $\mathbb{A}_{KB}$. Figure 1 shows the argument.

Answers to atomic queries are supported by arguments built from the ontology. However, it is possible to build arguments for conflicting atoms, and so arguments can *attack* each other. We now adopt the definitions of counter-argument and attacks for defeasible Datalog$^{\pm}$ ontologies from [22]. First, an argument $\langle \mathcal{B}, L' \rangle$ is a sub-argument of $\langle \mathcal{A}, L \rangle$ if $\mathcal{B} \subseteq \mathcal{A}$. Argument $\langle \mathcal{A}_1, L_1 \rangle$ counter-argues, rebuts, or attacks $\langle \mathcal{A}_2, L_2 \rangle$ at literal L, iff there exists a sub-argument $\langle \mathcal{A}, L \rangle$ of $\langle \mathcal{A}_2, L_2 \rangle$ such that L and L_1 conflict.

Example 12 Consider derivation ∂ from Example 9 and let $\mathcal{A}$ be the set of (defeasible) atoms and (defeasible) TGDs used in ∂. $\mathcal{A}$ is an argument for *sore_throat*(*axl*). Also, we can obtain a minimal derivation ∂' for *can_sing*(*axl*) where $\mathcal{B}$, the set of (defeasible) atoms and (defeasible) TGDs used in ∂', is such that no conflicting atoms can be defeasibly derived from $\mathcal{B} \cup \Sigma_T$. As {*sore_throat*(*axl*), *can_sing*(*axl*)} is conflicting relative to Σ_{NC}, we have that $\langle \mathcal{A}$, *sore_throat*(*axl*)$\rangle$ and $\langle \mathcal{B}$, *can_sing*(*axl*)$\rangle$ attack each other.

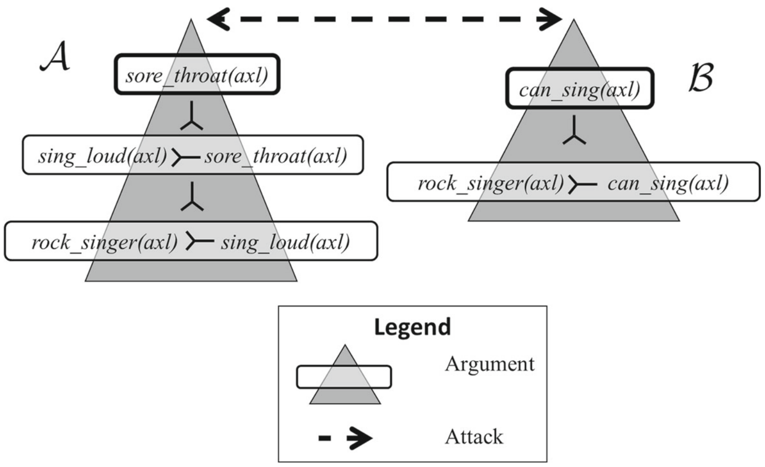
$\mathcal{A}$
sore_throat(axl)
sing_loud(axl) ≻— sore_throat(axl)
rock_singer(axl) ≻— sing_loud(axl)
$\mathcal{B}$
can_sing(axl)
rock_singer(axl) ≻— can_sing(axl)
Legend
Argument
Attack

Fig. 2 Attack between arguments

Once the attack relation is established between arguments, it is necessary to analyze whether the attack is strong enough so one of the arguments can *defeat* the other. Given an argument $\mathcal{A}$ and a counter-argument $\mathcal{B}$, a comparison criterion is used to determine if $\mathcal{B}$ is preferred to $\mathcal{A}$ and, therefore, *defeats* $\mathcal{A}$. For our defeasible Datalog$^{\pm}$ framework, unless otherwise stated, we assume an arbitrary preference criterion $\succ$ among arguments where $\mathcal{A} \succ B$ means that $\mathcal{B}$ is preferred to $\mathcal{A}$ and thus defeats it. More properly, given two arguments $\langle \mathcal{A}_1, L_1 \rangle$ and $\langle \mathcal{A}_2, L_2 \rangle$ we say that argument $\langle \mathcal{A}_1, L_1 \rangle$ is a defeater of $\langle \mathcal{A}_2, L_2 \rangle$ iff there exists a sub-argument $\langle \mathcal{A}, L \rangle$ of $\langle \mathcal{A}_2, L_2 \rangle$ such that $\langle \mathcal{A}_1, L_1 \rangle$ counter-argues $\langle \mathcal{A}, L \rangle$ at L, and either $\langle \mathcal{A}_1, L_1 \rangle \succ \langle \mathcal{A}, L \rangle$ (it is a proper defeater) or $\langle \mathcal{A}_1, L_1 \rangle \not\succ \langle \mathcal{A}, L \rangle$, and $\langle \mathcal{A}, L \rangle \not\succ \langle \mathcal{A}_1, L_1 \rangle$ (it is a blocking defeater).

Finally, the combination of arguments, attacks and comparison criteria gives raise to Datalog$^{\pm}$ argumentation frameworks.

Definition 8 Given a Defeasible Datalog$^{\pm}$ ontology KB defined over a relational schema $\mathcal{R}$, a *Datalog*$^{\pm}$ *argumentation framework* $\mathfrak{F}$ is a tuple $\langle \mathcal{L}_{\mathcal{R}}, \mathbb{A}_{KB}, \succ \rangle$, where $\succ$ specifies a preference relation defined over $\mathbb{A}_{KB}$.

To decide whether an argument $\langle \mathcal{A}_0, L_0 \rangle$ is undefeated within a Datalog$^{\pm}$ argumentation framework, all its defeaters must be considered, and there may exist defeaters for their counter-arguments as well, giving raise to *argumentation lines*. The dialectical process considers all possible admissible argumentation lines for an argument, which together form a dialectical tree. An *argument line* for $\langle \mathcal{A}_0, L_0 \rangle$ is defined as a sequence of arguments that starts at $\langle \mathcal{A}_0, L_0 \rangle$, and every element in the sequence is a defeater of its predecessor in the line [22]. Note that for defeasible Datalog$^{\pm}$ ontologies arguments in an argumentation line can contain both facts and defeasible atoms.

Different argumentation systems can be defined by setting a particular criterion for proper attack or defining the admissibility of argumentation lines. Here, we adopt the one from [22], which states that an argumentation line has to be finite, and no argument is a sub-argument of an argument used earlier in the line; furthermore, when an argument $\langle \mathcal{A}_i, L_i \rangle$

is used as a blocking defeater for $\langle \mathcal{A}_{i-1}, L_{i-1} \rangle$ during the construction of an argumentation line, only a proper defeater can be used for defeating $\langle \mathcal{A}_i, L_i \rangle$.

The dialectical process considers all possible admissible argumentation lines for an argument, which together form a dialectical tree. Dialectical trees for defeasible Datalog$^{\pm}$ ontologies are defined following [22], and we adopt the notion of coherent dialectical tree from [29], which ensures that the use of defeasible atoms is *coherent* in the sense that conflicting defeasible atoms are not used together in supporting (or attacking) a claim. We denote with $Args(\mathcal{T})$ the set of arguments in $\mathcal{T}$.

Definition 9 Let $\langle \mathcal{A}_0, L_0 \rangle$ be an argument from a Datalog$^{\pm}$ argumentation framework $\mathfrak{F}$. A dialectical tree for $\langle \mathcal{A}_0, L_0 \rangle$ from $\mathfrak{F}$, denoted $\mathcal{T}(\langle \mathcal{A}_0, L_0 \rangle)$, is defined as follows:

(1) The root of the tree is labeled with $\langle \mathcal{A}_0, L_0 \rangle$.
(2) Let N be a non-root node of the tree that is labeled $\langle \mathcal{A}_n, L_n \rangle$, and $C = [\langle \mathcal{A}_0, L_0 \rangle, \langle \mathcal{A}_1, L_1 \rangle, \ldots, \langle \mathcal{A}_n, L_n \rangle]$ be the sequence of labels of the path from the root to N. Let $\langle \mathcal{B}_1, Q_1 \rangle, \langle \mathcal{B}_2, Q_2 \rangle, \ldots, \langle \mathcal{B}_k, Q_k \rangle$ be all the defeaters for $\langle \mathcal{A}_n, L_n \rangle$. For each defeater $\langle \mathcal{B}_i, Q_i \rangle (1 \leq i \leq k)$, such that the argumentation line $C' = [\langle \mathcal{A}_0, L_0 \rangle, \langle \mathcal{A}_1, L_1 \rangle, \langle \mathcal{A}_2, L_2 \rangle, \ldots, \langle \mathcal{A}_n, L_n \rangle, \langle \mathcal{B}_i, Q_i \rangle]$ is admissible, the node N has a child N_i labeled $\langle \mathcal{B}_i, Q_i \rangle$. If there is no defeater for $\langle \mathcal{A}_n, L_n \rangle$ or there is no $\langle \mathcal{B}_i, Q_i \rangle$ such that C' is admissible, then N is a leaf.

Argument evaluation, i.e., determining whether the root node of the tree is defeated or undefeated, is done by means of a *marking* or *labelling* criterion. Each node in an argument tree is labelled as either defeated (D) or undefeated (U). We denote the dialectical tree built for the argument $\mathcal{A}$ supporting claim L as $\mathcal{T}(\langle \mathcal{A}, L \rangle)$, $Args(\mathcal{T})$ the set of arguments in $\mathcal{T}$, and the root of $\mathcal{T}(\langle \mathcal{A}, L \rangle)$ with $root(\mathcal{T}(\langle \mathcal{A}, L \rangle))$. Also, $marking(N)$, where N is a node in a dialectical tree, denotes the value of the marking for node N (either U or D). Deciding whether a node is defeated or undefeated depends on whether or not all its children are defeated: (1) if node N is a leaf then $marking(N) = U$, (2) node N is such that $marking(N) = D$ iff at least one of its children that is marked with U, and (3) node N is such that $marking(N) = U$ iff all its children are marked with D.

By means of the marking procedure we can define when an atom is *warranted* in the argumentation framework for a Defeasible Datalog$^{\pm}$ ontology.

Definition 10 (Query answering semantics in Defeasible Datalog$^{\pm}$) Let KB be a Defeasible Datalog$^{\pm}$ ontology and $\mathfrak{F}$ the corresponding Datalog$^{\pm}$ argumentation framework where $\succ \in \mathfrak{F}$ is an arbitrary argument comparison criterion. An atom L is *warranted* in $\mathfrak{F}$ (through $\mathcal{T}$) iff there exists an argument $\langle \mathcal{A}, L \rangle$ such that $marking(root(\mathcal{T}(\langle \mathcal{A}, L \rangle))) = U$. We say that L is entailed from KB (through $\mathfrak{F}$), denoted with $KB \models_{\mathfrak{F}} L$, iff it is *warranted* in $\mathfrak{F}$.

Example 13 Suppose that we have the query $Q = can\_sing(axl)$, i.e., we want to know whether or not Axl can sing. Consider the conflict between arguments $\mathcal{A}$ and $\mathcal{B}$ shown in Example 12. As we have stated, we do not define any particular criterion $\succ$ to solve attacks. Nevertheless, for the sake of example assume now that we are indeed using a criterion $\succ$ that is such that $\mathcal{B} \succ \mathcal{A}$. Under such supposition we have the labelled dialectical tree shown in Fig. 3.

As can be seen in the dialectical tree, if we assume that $\mathcal{B} \succ \mathcal{A}$ then we have reasons to think that Axl cannot sing due to its throat being sore.

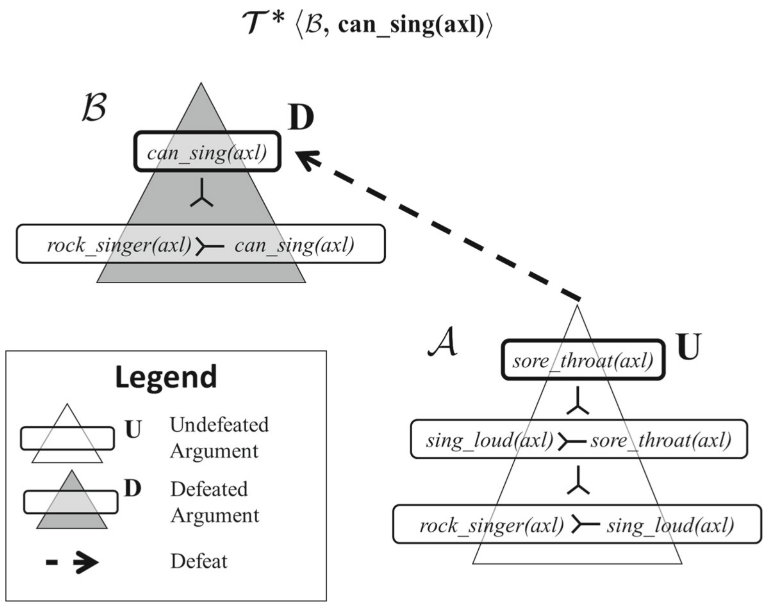
$\mathcal{T}^*$ ⟨$\mathcal{B}$, can_sing(axl)⟩
$\mathcal{B}$
D
can_sing(axl)
rock_singer(axl) ≻— can_sing(axl)
$\mathcal{A}$
sore_throat(axl)
U
sing_loud(axl) ≻— sore_throat(axl)
rock_singer(axl) ≻— sing_loud(axl)
Legend
U
Undefeated Argument
D
Defeated Argument
Defeat

Fig. 3 A labelled dialectical tree for atom *can_sing*(*axl*)

Now, we establish how a classic Datalog$^{\pm}$ ontology can be transformed to a defeasible one. Intuitively, the transformation of a classic ontology to a defeasible one involves transforming every atom and every TGD in the classic ontology to its defeasible version.

Definition 11 (Transformation between ontologies) Let $KB = (D, \Sigma_T \cup \Sigma_E \cup \Sigma_{NC})$ be a classic Datalog$^{\pm}$ ontology. Then, its transformation to a defeasible Datalog$^{\pm}$ ontology, denoted $\mathcal{D}(KB)$, is a defeasible ontology $KB' = (F, D', \Sigma'_T, \Sigma_D, \Sigma_E \cup \Sigma_{NC})$ where $F = \emptyset$, $D' = D$, $\Sigma'_T = \emptyset$ and $\Sigma_D = \{\Upsilon(\mathbf{X}, \mathbf{Y}) \succ\!\!- \exists \mathbf{Z}\, \Psi(\mathbf{X}, \mathbf{Z}) \mid \Upsilon(\mathbf{X}, \mathbf{Y}) \rightarrow \exists \mathbf{Z}\, \Psi(\mathbf{X}, \mathbf{Z})\}$.

Finally, we define a query answering semantics for classical Datalog$^{\pm}$ ontologies for atomic queries. The reason behind this choice is that we are relying on query answering semantics as in logic programming and defeasible logic programming [22] as it seems to be the more natural, simpler choice for defeasible Datalog$^{\pm}$; of course, it will also be quite interesting to explore different, more complex alternatives to expand the capabilities of the system to answer queries more precisely, a line of work that we will leave for the future. From now on we denote such semantics as $\mathbf{D}^2$ (**D**efeasible **D**atalog$^{\pm}$). Intuitively, a literal is an answer for a classical Datalog$^{\pm}$ ontology KB under the $\mathbf{D}^2$ semantics iff it is warranted in the transformation of KB to a defeasible one.

Definition 12 (Query answering in classical Datalog$^{\pm}$ under $\mathbf{D}^2$) Let $KB = (D, \Sigma_T \cup \Sigma_E \cup \Sigma_{NC})$ be a classic Datalog$^{\pm}$ ontology, $KB' = \mathcal{D}(KB)$ its defeasible transformation, Q a query and $\succ$ a comparison criterion. Then, a ground atom L is an *answer for Q from KB under* $\mathbf{D}^2$, denoted $KB \models_{\mathbf{D}^2_{\succ}} L$, iff there exists a homomorphism h such that $h(Q) \subseteq D$, $h(Q) = L$ and $KB' \models_{\mathfrak{F}} L$ where $\mathfrak{F} = \langle \mathcal{L}_{\mathcal{R}}, \mathbb{A}_{KB'}, \succ \rangle$.

Note that the semantics is parametrized by the comparison criterion $\succ$, which helps to solve conflicts when they arise.

5.2 Influence of incoherence in defeasible Datalog$^{\pm}$

Now, we focus on the behaviour of Defeasible Datalog$^{\pm}$ regarding atoms relevant to unsatisfiable sets of TGDs. It can be shown that the argumentation framework $\mathfrak{F} = \langle \mathcal{L}_{\mathcal{R}}, \mathbb{A}_{\mathcal{D}(KB)}, \succ \rangle$ is such that one relevant atom L to an unsatisfiable set is warranted (and thus an answer), provided that the comparison criterion $\succ$ is such that $marking(root(\mathcal{T}_{\mathfrak{F}}(\langle \mathcal{A}, L\rangle))) = U$ for some dialectical tree $\mathcal{T}_{\mathfrak{F}}(\langle \mathcal{A}, L\rangle)$ built upon $\mathfrak{F}$. It is interesting to see that such comparison criterion can always be found: intuitively, it suffices to arbitrary establish $\mathcal{A}$ as the most preferred argument in $\mathbb{A}_{\mathcal{D}(KB)}$ (note however that other criteria can have the exact same result).

Proposition 4 *Let KB be a Datalog$^{\pm}$ ontology defined over a relational schema $\mathcal{R}$, and KB' be a Defeasible Datalog$^{\pm}$ ontology such that $\mathcal{D}(KB) = KB'$. Finally, let $L \in D$ and $U \in \mathcal{U}(KB)$ such that L is relevant to U. Then, it holds that there exists $\succ$ such that $KB \vDash_{\mathbf{D}^2_{\succ}} L$.*

Proof Let $L \in D$ and $U \in \mathcal{U}(KB)$ such that L is relevant to U. Since L is a literal then there exists a derivation $\partial = L$, i.e., the literal derives itself. Consider the argument $\mathcal{A} = \langle \partial, L\rangle$. Now, let $\succ$ be an argument criterion defined as follows: for any argument $\langle \mathcal{B}, L'\rangle$ such that L and L' are conflicting it holds that $\langle \mathcal{B}, L'\rangle \succ \langle \mathcal{A}, L\rangle$. Then, we have that $\mathcal{T}(\langle \mathcal{A}, L\rangle)$ where $Args(\mathcal{T}) = \{\mathcal{A}\}$ is a proper dialectical tree, since no defeater can be found for $\mathcal{A}$. Then, the root of $\mathcal{T}(\langle \mathcal{A}, L\rangle)$ is marked as undefeated, and then $KB \vDash_{\mathbf{D}^2_{\succ}} L$. Thus, it holds that there exists $\succ$ such that $KB \vDash_{\mathbf{D}^2_{\succ}} L$. □

Corollary 2 (Corollary from Proposition 4) *Given a Datalog$^{\pm}$ontology KB there exists $\succ$ such that $\mathbf{D}^2_{\succ}$ applied to KB is tolerant to incoherence.*

Proof Straightforwardly follows from Proposition 4. □

As an example of the above corollary, consider again the running example.

Example 14 Let $KB' = \mathcal{D}(KB)$ be the defeasible transformation of KB in Example 1, where the sets Σ_E and Σ_{NC} are the same, $F = \emptyset$, $D = \{can\_sing(simone)$, $rock\_singer(axl)$, $sing\_loud(ronnie)$, $has\_fans(ronnie)$, $manage(band_1, richard)$, $sang\_in(axl, band_2)$, $pop\_band(band_2)$, $invited\_guest(axl, band_2)\}$, and

$$\begin{aligned}\Sigma_D = \{&rock\_singer(X) \succ\!\!- sing\_loud(X),\\ &sing\_loud(X) \succ\!\!- sore\_throat(X),\\ &has\_fans(X) \succ\!\!- famous(X),\\ &rock\_singer(X) \succ\!\!- can\_sing(X),\\ &sang\_in(X, Y) \wedge pop\_band(Y) \succ\!\!- pop\_singer(X)\}\end{aligned}$$

Now, for the sake of example suppose that we define our comparison criterion to model the fact that we trust the information that says that Axl had singed with the pop band Band$_2$

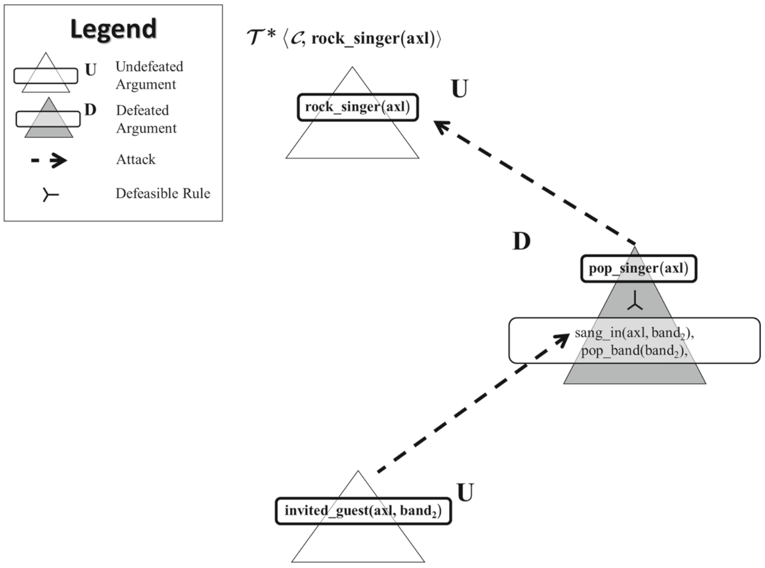

Fig. 4 A labelled dialectical tree for atom *rock_singer*(*axl*)

over the mere fact that he is a rock singer, and that Axl was only invited to sing with them at a certain time rather than he being the actual singer of Band$_2$. Then, we have the dialectical tree shown in Fig. 4, where the argument $\langle [rock\_singer(axl)], rock\_singer(axl)\rangle$ is its undefeated root.

Then, clearly $KB' \models_{\mathfrak{F}} rock\_singer(axl)$, and thus $KB \vDash_{\mathbf{D}^2_{\succ}} rock\_singer(axl)$.

6 Related work

In Artificial Intelligence, many efforts for dealing with potentially inconsistent information have been developed in the last four decades. Frameworks such as default logic [34] 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. Finally, argumentation methods [18, 30, 37] have been used for handling uncertainty and inconsistency by means of reasoning about how contradictory arguments defeat each other. In the database community, the field of *database repairing* and *consistent query answering* (CQA) has gained attention since [1], which provided a model-theoretic construct of a database *repair*. The work of [12] addresses the basic concepts and results of the area of CQA. Recently, Ontology-based Data Access approach to data integration, has led to a resurgence of interest in this area as well, specially focusing on the development of efficient inconsistency-tolerant reasoning and query answering semantics in DLs and other ontological languages. Lately, several works have focused on inconsistency handling for different classes of DLs, adapting and specializing general techniques previously considered for traditional logics [23, 27, 31]. In [25], the adaptation of CQA for *DL-Lite* ontologies and several sound and complete approximation are studied. The data and combined complexity of the semantics were studied in [35] for a wide spectrum of DLs. Computing consistent answers is an inherently hard problem, [25] shows co-NP completeness even for ground atomic queries in *DL-Lite*, though some works identify cases for very simple ontologies and restricted queries (within the *DL-Lite* family) for which tractable results can be obtained [4]. In [35], the complexity for query answering under inconsistency-tolerant semantics is provided for a wide spectrum of DLs, ranging from tractable ones ($\mathcal{EL}$) to very expressive ones (*SHIQ*). In [26], an alternative semantics called k-lazy is proposed, which relaxes the notion of repairs by adopting a compromise between quality of answers and tractability for fragments of Datalog$^{\pm}$. The work of [2] contains a survey of existing approaches for handling inconsistencies in DL-based ontologies.

This work, however, focuses on the study of the effects of incoherence in query answering inconsistency-tolerant semantics. We show that the inconsistency-tolerant semantics proposed in [4, 25, 26, 35] are not incoherent-tolerant semantics and their use in such scenarios may lead to useless answers. In general, all CQA-based proposals do not address the issues of incoherence and inconsistency together by design. Instead most of the approaches assume that the set of integrity constraints correctly defines the semantics of the database instance, so there is no room for incoherence. In our case, both the information contained in the database instance and whatever can be obtained from the application of TGDs are defeasible. Though not analyzed in this work, the proposed framework is flexible enough to implement different semantics depending on the particular transformation we apply to a Datalog$^{\pm}$ ontology. For instance, one could assume that the information in the database instance must be always considered correct, in which case that information is translated into a set of facts instead of into a set of defeasible atoms as stated in Definition 11. Also, if only the database instance is translated into defeasible atoms, then we obtain the ICR semantics from [4] (or some approximation to it depending on the preference criterion used for the argument comparison).

As mentioned in the introduction, incoherence has been studied recently in the Description Logics area [21, 32, 36]. In particular the notion of incoherence to which we attend in this work is inspired by that analyzed in [21, 36]. In an ontological settings, incoherence refers to a set of ontological rules that cannot be applied without leading to violations of the constraints imposed on the knowledge, making them unsatisfiable. In [36] that paper the authors focus on the definition of processes capable of detect unsatisfiabilities and incoherences in DLs ontologies, introducing complete algorithms along with an empirical analysis of the approach. The authors do not cover the issue of how to reason with a potential incoherent knowledgebase and what type of information can be obtained from one. On the other hand, in [21] the authors introduce a general framework that aims towards a formalization of dynamic processes in ontologies, where such dynamics is captured by Belief Revision (BR) approaches. Since classical BR approaches rely on an assumption that the underlying logic is capable of expressing negations (which ontologies like DL cannot), one of the main contributions by Flouris et al. is introducing different ways of achieving negations in such ontologies. A particularly interesting introduced negation for the purpose of our work is that of *coherence negation*, which states that two atoms are the (coherence) negation of each other if when considered together we have an unsatisfiable concept. In this way Flouris et al. successfully shows how incoherence is of utter importance in ontologies.

An alternative way of looking into incoherences is provided by Qi and Hunter in [32]. Instead of going the classical direction of finding incoherences and solving them in some way, in that work the authors propose different ways of measuring the level of incoherence. They do this by defining two classes of measures for incoherent ontologies. The first measure is tailored towards for unsatisfiable concepts, whereas the second one is focused on measuring incoherence for terminologies. The first class of measures gives us information on comparing unsatisfiable concept names. The second class of measures gives us information on comparing terminology axioms and comparing ontologies. Moreover, the authors show some empirical results based on an implementation of the approaches. The approaches proposed by Qi and Hunter can be very benefitial for argumentation-based incoherent-tolerant semantics such as $\mathbf{D}^2$. Notice that, as explained before, argumentation-based semantics need ways of defining which argument prevails when an attack between arguments arise. In our work this is captured by using a general argument comparison criterion $\succ$. As Qi and Hunter states, the proposed measures of incoherence can provide important information for dealing with incoherence and evaluating ontologies. In our framework this could translate to instantiations of $\succ$ accounting for such measures. In particular, the measure of incoherence of unsatisfiable concepts can be used to define an ordering among different concepts regarding how much they influence incoherence (by using Shapley Values, as stated by Qi and Hunter). Clearly, this ordering can in turn be used to define a relation $\succ$ in which arguments using concepts with a lower incoherence value are the preferred ones (for instance by using weakest link). This way is clear that the results shown in [32] can be exploited to further refine the behaviour of argument-based incoherence tolerant such as the one presented here.

An interesting, highly related work with our is [13]. In that work the authors aim to find out the relation between two different classes of inconsistency-tolerant semantics, namely argumentation-based semantics and inconsistent ontological KB query answering. The authors focus on some really interesting research questions such as whether or not the two classes of semantics can obtain the same results, and under which conditions this holds. So, in a sense the strategy used in that work resembles ours: they compare the answers that can be obtained by the different semantics to find out inclusions or equivalences. The results shown by Croitoru and Vesic indicates that indeed there are several equivalences between inconsistent ontological KB query answering and different instantiations of Dung's argumentation frameworks [18]. For instance, they show that AR semantics corresponds to universal acceptance under stable / preferred argumentation semantics. In this work we have taken a similar path, but considering a particular formalism (Defeasible Datalog$^{\pm}$with $\mathbf{D}^2$ semantics) which has a semantics that is instead closer to the grounded semantics [18]. Our results, however, indicates that there exists answers that can be obtained through argumentation but cannot be obtained by repair-based semantics. This apparent contradiction has a reason, and that is incoherence. As noticed, under an incoherence-free assumption the semantics are equivalent, but once we drop the assumption the situation changes, because of the difference in obtaining answers based on repairs or arguments. That is, as said before repair based semantics such as AR are not tolerant to incoherence, whereas $\mathbf{D}^2$ is, and thus the semantics are not equivalent for incoherent application domains.

In the field of Logic Programming an interesting work regarding incoherence is [17]. In that work the case of unsatisfiable programs is studied, similar to the way we consider incoherence leaded by unsatisfiable sets of TGDs. Two different approaches for merging a set of logic programs are studied. The first one follows an arbitration approach, selecting the models of a program that differs the least *w.r.t.* the models of the other programs. The strategy to solve unsatisfiability is simply leaving the unsatisfiable program out of consideration for the merging, instead of trying to solve the conflict somehow. We can argue that this approach is not technically tolerant to incoherence as the incoherent program is merely discarded from consideration and thus query answering is no longer a problem in the merged program. The second approach is based on the selection of the models of a special program

P_0, which can be thought as the constraints guiding the merging process, that has the least variations *w.r.t.* the programs for the merging. This last approach though it is able to handle incoherent information without getting rid of it completely also does not technically fit the definition of incoherent-tolerant query answering semantics as it proposes changes in the knowledge base(s); however, the rationale behind that particular incoherence/inconsistency resolution method could in theory be exploited to develop new interesting query answering semantics tolerant to incoherence. This particular study is left for future work.

Quite related to our work, [6] proposes an approach that is capable of using information coming from several DL ontologies in order to answer queries, taking care in the process of both incoherence and inconsistency. Their approach is based on agents with argumentative capabilities, each one with a personal knowledge base in the form of a DL ontology. These agents use dialogue games to interchange arguments until they reach an agreement about the answer to a certain query. Thus, the agents can use the (possible incoherent/inconsistent) union of the ontologies without merging them, and still obtain an answer influenced by every ontology in play. Moreover, this approach has the advantage that no information is lost, as no formula is deleted from the ontologies. A difference between their work and ours is that in our work we use Defeasible Datalog$^{\pm}$[28] which considers the application of TGDs as defeasible, making the truth of every derived literal open to challenge. Nevertheless, the proposal in [6] is indeed tolerant to incoherences. Another multiagent framework using argumentation-based reasoning is presented in [38]. Much like in [6], in this work the authors enable agent to use argumentation to reach some consensus regarding a posted query. The framework consists of several agents, and an special agent called the moderator, which coordinates the argumentation process carried out by the rest of agents. Intuitively, the moderator accepts a query, consisting of a single literal, and then proceeds to ask the agents for reasons in favor or against it. Eventually, the system returns an answer to the questioner, according to the agents' knowledge. Moreover, just as our proposed semantics $\mathbf{D}^2$, the argumentation process is based on DeLP, which brings the work closer to our proposal. Opposed to [6] and our semantics semantics $\mathbf{D}^2$, the framework presented in [38] was not designed with an ontology environment in mind. Thus, the notion of incoherence (which as said is almost exclusively considered within the ontologies community) is not considered in the work. Nevertheless, since the DeLP-based reasoning process is similar to our semantics then an adaptation of the framework to an ontology setting will render an incoherence-tolerant semantics, further reinforcing our claim that our characterization is not tailored to our particular proposal and that there exists other incoherence-tolerant semantics as well.

7 Conclusions

One problem that has been increasingly receiving attention in Knowledge Representation and Reasoning is that of incoherence. Such problem is specially important for scenarios where different sources of information need to be integrated, since such integration is prone not only to errors at the database level but also at the constraints one. Nevertheless, since incoherence is a fairly recent concept most of the works in query answering for Datalog$^{\pm}$ ontologies and DLs have focused on consistency issues making the assumption that the set of constraints correctly represents the semantics of the data and therefore any conflict can only come from the data itself. Even when such assumption is reasonable in certain scenarios, clearly in more general ones the assumption does not hold: there are known ontologies such as the "Diagnoses for Intensive Care Evaluation" (DICE) ontology where incoherence arises, as noticed by Schlobach and Cornet [36].

We begin our work with a presentation of the concept of incoherence for Datalog$^{\pm}$ ontologies, which is an adaptation to such language of efforts made in the Description Logics community. In Datalog$^{\pm}$, incoherence is related to the presence of sets of TGDs such that their application inevitably yield to violations in the set of negative constraints and equality-generating dependencies. Then, these sets of TGDs are in a sense in an "idle" state: they may not be actually provoking a violation of a EGD or NC at the time, but neither can they be applied at the same time. This is a clear indication that there is something wrong with the set of TGDs as a whole (and perhaps with the knowledge the TGDs try to model).

After our presentation of the concept, we have shown how incoherence affects inconsistency-tolerant semantics. The first important matter regarding this relation is that when an ontology is such that some unsatisfiable set of TGDs is activated by the database instance in the ontology then the knowledge base is imminently inconsistent. This means that in such scenarios the knowledge base completely rule out the use of classic (i.e., non inconsistency-tolerant) semantics, and thus we necessarily need to turn out to those that can tolerate inconsistency. Nevertheless, we have shown in the paper that not every inconsistency-tolerant semantics can deal with incoherence-driven conflicts in a satisfactory way. Since they were not designed to address incoherence (or even acknowledge it for that matter), then some of the most well known inconsistency tolerant semantics are greatly affected by inconsistencies arising from the activation of unsatisfiable sets of TGDs, to the point that for some incoherent ontologies these semantics may produce no useful answer at all.

Considering all things mentioned, we have concluded our work proposing a further classification regarding query answering semantics, i.e., whether or not they can be seen as tolerant to incoherence. We have analyzed a particular semantics satisfying that property. Such semantics rely on the use of argumentation over Datalog$^{\pm}$ ontologies, and conflicting pieces of information are warranted or not (thus being answer or not under the semantics) after a dialectical process is carried out, which intuitively analyzes all reasons in favor and against the atom in the query. To do this, in the current version of the framework the semantics assume that there are no pieces of information in the ontology settled in stone, and leaves all atoms and TGDs up to debate by transforming them to their defeasible versions. Nevertheless, if it is required by the application at hand then it is certainly possible to alter this behavior so more valuable information (under some criterion) is left as strict knowledge, giving preponderance to them.

Through the years, various efforts have been carried out to establish the computational complexity of argumentation [11, 19, 20, 39]. In particular, in [11] is proven that the decision problem *"Is a set of defeasible rules an argument for a literal under a defeasible logic program?"* in DeLP [22] is **P**-complete; whilst the problem *"Does there exist an argument for a literal under a defeasible logic program?"* is proven to be in **NP**. Moreover, in [39] authors explore equivalence relations for set of arguments, showing for instance that to check such equivalence is co-**NP**-complete. Nevertheless, despite such results regarding the intrinsic complexity of argumentation, that does not imply that argumentation cannot be used for real-world applications, because such studies involve worst-case scenarios that may greatly differ from the average case. In fact, it has been shown in [14] that defeasible argumentation can be combined with relational databases as a provider of argument-supporting information, and empirical results have indicated that execution times are often improved by this when building arguments considering massive amounts of data; which was further evidenced by the development of a real-world recommender system based on such framework [7]. It will be interesting then to analyze the relation between the complexity of a task such as building an argument or establishing attack relations with the particular characteristics of Datalog$^{\pm}$, specially over the tractable fragments of the family; since it is possible that a similar effect to that of the efficiency of database management systems may arise from the use of mechanisms capable of dealing with massive information such as those in Datalog$^{\pm}$. Nevertheless, to perform a comprehensive study of such relation is beyond the scope of this paper, and left for future work.

Finally, we would like to stress out that our definition of incoherence-tolerant semantics is not tied to our particular proposal or the Datalog$^{\pm}$ language, and that there exists other frameworks that also falls under our definition, as it is the case of the work (also argumentation-based) by Black et al. [6] where dialogue games between agents are used to solve queries under Description Logics ontologies that can be incoherent.

References

1. Arenas, M., Bertossi, L.E., Chomicki, J.: Consistent query answers in inconsistent databases. In: Proceedings of PODS, pp. 68–79 (1999)
2. Bell, D.A., Qi, G., Liu, W.: Approaches to inconsistency handling in description-logic based ontologies. In: OTM Workshops, vol. 2, pp. 1303–1311 (2007)
3. Besnard, P., Hunter, A.: Elements of argumentation. MIT Press (2008)
4. Bienvenu, M.: On the complexity of consistent query answering in the presence of simple ontologies. In: Proceedings of AAAI (2012)
5. Bienvenu, M., Rosati, R.: Tractable approximations of consistent query answering for robust ontology-based data access. In: Proceedings of IJCAI (2013)
6. Black, E., Hunter, A., Pan, J.Z.: An argument-based approach to using multiple ontologies. In: SUM, pp. 68–79 (2009)
7. Briguez, C.E., Budàn, M.C., Deagustini, C.A.D., Maguitman, A.G., Capobianco, M., Simari, G.R.: Argument-based mixed recommenders and their application to movie suggestion. Expert Syst. Appl. **41**(14), 6467–6482 (2014)
8. Calì, A., Gottlob, G., Lukasiewicz, T.: A general Datalog-based framework for tractable query answering over ontologies, vol. 14, pp. 57–83 (2012a)
9. Calì, A., Gottlob, G., Lukasiewicz, T.: A general Datalog-based framework for tractable query answering over ontologies. J. Web Semant. **14**, 57–83 (2012b)
10. Calì, A., Lembo, D., Rosati, R.: On the decidability and complexity of query answering over inconsistent and incomplete databases. In: Proceedings of PODS 2003, pp. 260–271. ACM (2003)
11. Cecchi, L., Fillottrani, P., Simari, G.R.: On the Complexity of Delp through Game Semantics. In: Dix, J., Hunter, A. (eds.) Proceedings 11Th Intl. Workshop on Nonmonotonic Reasoning (NMR 2006, pp. 386–394 (2006)
12. Chomicki, J.: Consistent query answering: five easy pieces. In: Proceedings of ICDT, pp. 1–17 (2007)
13. Croitoru, M., Vesic, S.: What can argumentation do for inconsistent ontology query answering? In: Scalable uncertainty management, pp. 15–29. Springer (2013)
14. Deagustini, C.A.D., Dalibón, S.E.F., Gottifredi, S., Falappa, M.A., Chesñevar, C.I., Simari, G.R.: Relational databases as a massive information source for defeasible argumentation. Knowl.-Based Syst. **51**, 93–109 (2013)
15. Deagustini, C.A.D., Martinez, M.V., Falappa, M.A., Simari, G.R.: Datalog± ontology consolidation. J. Artif. Intell. Research (JAIR) (2016). To appear
16. Delgrande, J.P., Jin, Y.: Parallel belief revision: Revising by sets of formulas. Artif. Intell. **176**(1), 2223–2245 (2012)
17. Delgrande, J.P., Schaub, T., Tompits, H.: Stefanwoltran merging logic programs under answer set semantics. In: Hill, P., Warren, D. (eds.) ICLP. Vol. 5649 of lecture notes in computer science, pp. 160–174. Springer (2009)
18. Dung, P.M.: On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. Artif. Intell. **77**, 321–357 (1995)
19. Dunne, P., Wooldridge, M.: Argumentation in artificial intelligence, pp. 85–104. Springer, Ch. Complexity of Abstract Argumentation (2009)
20. Dvořák, W., Woltran, S.: Complexity of semi-stable and stage semantics in argumentation frameworks. Inf. Process. Lett. **110**(11), 425–430 (2010)

21. Flouris, G., Huang, Z., Pan, J.Z., Plexousakis, D., Wache, H.: Inconsistencies, negations and changes in ontologies. In: AAAI, pp. 1295–1300. AAAI Press (2006)
22. García, A.J., Simari, G.R.: Defeasible logic programming: an argumentative approach. TPLP **4**(1–2), 95–138 (2004)
23. Huang, Z., van Harmelen, F., ten Teije, A.: Reasoning with inconsistent ontologies. In: Proceedings of IJCAI, pp. 354–359 (2005)
24. Konieczny, S., Pérez, R.P.: Merging information under constraints: a logical framework. J. Log. Comput. **12**(5), 773–808 (2002)
25. Lembo, D., Lenzerini, M., Rosati, R., Ruzzi, M., Savo, D.F. In: Proceedings of RR, pp. 103–117 (2010)
26. Lukasiewicz, T., Martinez, M.V., Simari, G.I.: Inconsistency handling in Datalog+/– ontologies. In: Proceedings of ECAI, pp. 558–563 (2012)
27. Ma, Y., Hitzler, P.: Paraconsistent reasoning for OWL 2. In: Proceedings of RR. Vol. 5837 of LNCS, pp. 197–211. Springer (2009)
28. Martinez, M.V., Deagustini, C.A.D., Falappa, M.A., Simari, G.R.: Inconsistency-tolerant reasoning in Datalog± Ontologies via an argumentative semantics. In: Proceedings of IBERAMIA 2014, pp. 15–27 (2014)
29. Martinez, M.V., García, A.J., Simari, G.R.: On the use of presumptions in structured defeasible reasoning. In: Proceedings of COMMA, pp. 185–196 (2012)
30. Prakken, H., Sartor, G.: Argument-based extended logic programming with defeasible priorities. J. Appl. Non-Classical Logics **7**(1) (1997)
31. Qi, G., Du, J.: Model-based revision operators for terminologies in description logics. In: Proceedings of IJCAI, pp. 891–897 (2009)
32. Qi, G., Hunter, A.: Measuring incoherence in description logic-based ontologies. In: ISWC/ASWC, pp. 381–394 (2007)
33. Rahwan, I., Simari, G.R.: Argumentation in artificial intelligence. Springer (2009)
34. Reiter, R.: A logic for default reasoning. Artif. Intel. **13**(1–2), 81–132 (1980)
35. Rosati, R.: On the complexity of dealing with inconsistency in description logic ontologies. In: Proceedings of IJCAI, pp. 1057–1062 (2011)
36. Schlobach, S., Cornet, R.: Non-standard reasoning services for the debugging of description logic terminologies. In: Proceedings of IJCAI 2003, pp. 355–362 (2003)
37. Simari, G.R., Loui, R.P.: A mathematical treatment of defeasible reasoning and its implementation. Artif. Intell. **53**(2–3), 125–157 (1992)
38. Thimm, M.: Realizing argumentation in multi-agent systems using defeasible logic programming. In: Argumentation in multi-agent systems, pp. 175–194. Springer (2010)
39. Wooldridge, M., Dunne, P.E., Parsons, S.: On the complexity of linking deductive and abstract argument systems. In: AAAI, vol. 6, pp. 299–304 (2006)