On the Influence of Incoherence in Inconsistency-tolerant Semantics for Datalog±

C. A. Deagustini and **M. V. Martinez** and **M. A. Falappa** and **G. R. Simari**
AI R&D Lab., Dep. of Computer Science and Engineering,
Universidad Nacional del Sur, Bahía Blanca, Argentina
Consejo Nacional de Investigaciones Científicas y Técnicas (CONICET),
Avenida Rivadavia 1917, Ciudad Autónoma de Buenos Aires, Argentina

Abstract

The concept of incoherence naturally arises in ontological settings, specially when integrating knowledge. In this work we study a notion of incoherence for Datalog$^{\pm}$ ontologies based on the definition of satisfiability of a set of existential rules regarding the set of integrity constraints in a Datalog$^{\pm}$ ontology. We 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—even under inconsistency-tolerant semantics, which is expected as they were not designed to confront such issues. Finally, we propose a notion of incoherency-tolerant semantics for query answering in Datalog$^{\pm}$, and present a particular one based on the transformation of classic Datalog$^{\pm}$ ontologies into defeasible Datalog$^{\pm}$ ones, which use argumentation as its reasoning machinery.

Introduction and Motivation

The problem of inconsistency in ontologies has been widely acknowledged in both the Semantic Web and Database Theory communities, and several methods have been developed to deal with it, *e.g.,* (Arenas, Bertossi, and Chomicki 1999; Lembo et al. 2010; Lukasiewicz, Martinez, and Simari 2012; Black, Hunter, and Pan 2009; Bienvenu 2012; Martinez et al. 2014). The most widely accepted semantics for querying inconsistent databases is that of *consistent answers* (Arenas, Bertossi, and Chomicki 1999) (or AR semantics in (Lembo et al. 2010) for ontological languages), which yields the set of atoms that can be derived despite all possible ways of repairing the inconsistency. In this semantics often an assumption is made that the set of ontological knowledge Σ expresses the semantics of the data and as such there is no internal conflict on the set of constraints, which is not subject to changes over time. This means first, that the set of constraints is always satisfiable, in the sense that their application do not *inevitably* yield a consistency problem; 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 and that is the part of the ontology that must be modified in order to restore consistency.

Copyright © 2015, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

Although to consider the constraints as always satisfiable is a reasonable assumption to make, specially in the case of a single ontology, in this work we will focus on a more general setting and consider that both data and constraints can change through time and become conflicting. In this more general scenario, as knowledge evolves (and so the ontology that represents it) not only data related issues can appear, but also constraint related ones. The problem of conflicts among constraints is known in the Description Logics community as *incoherence* (Flouris et al. 2006; Qi and Hunter 2007). As they were not developed to consider this kind of issue, 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 of ontological languages, namely Datalog$^{\pm}$ (Calì, Gottlob, and Lukasiewicz 2012a). 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.* (2014) 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, we introduce the notion of incoherence for Datalog$^{\pm}$ ontologies, relating it 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, introducing an alternative semantics based on an argumentative reasoning process over the transformation of Datalog$^{\pm}$ ontologies to their correspondent defeasible Datalog$^{\pm}$ ontologies. 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.

Preliminaries

First, we briefly recall some basics on Datalog$^{\pm}$ (Calì, Gottlob, and Lukasiewicz 2012a). 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, Σ_T), is the exhaustive application of the TGDs (Calì, Gottlob, and Lukasiewicz 2012b) 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, Σ), is the set of all tuples $\mathbf{a}$ such that $\mathbf{a} \in Q(B)$ for all $B \in$ *mods*(D, Σ). 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*$(D, \Sigma_T) \models Q$ (Calì, Gottlob, and Lukasiewicz 2012b).

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* (Calì, Gottlob, and Lukasiewicz 2012a); 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 (Calì, Lembo, and Rosati 2003) (actually tractable) fragments of Datalog$^{\pm}$. 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.,* $p(X, Y) \wedge q(X, Z) \rightarrow Y = Z$.

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*.

$$\left\{\begin{array}{ll} D: & \{a_1 : can\_sing(simone) \\ & \quad a_2 : rock\_singer(axl), \\ & \quad a_3 : sing\_loud(ronnie), \\ & \quad a_4 : has\_fans(ronnie), \\ & \quad a_5 : manage(band_1, richard)\} \\ \Sigma_{NC}: & \{\tau_1 : sore\_throat(X) \wedge can\_sing(X) \rightarrow \bot, \\ & \quad \tau_2 : unknown(X) \wedge famous(X) \rightarrow \bot\} \\ \Sigma_E: & \{\nu_1 : manage(X,Y) \wedge manage(X,Z) \rightarrow Y = Z\} \\ \Sigma_T: & \{\sigma_1 : rock\_singer(X) \rightarrow sing\_loud(X), \\ & \quad \sigma_2 : sing\_loud(X) \rightarrow sore\_throat(X), \\ & \quad \sigma_3 : has\_fans(X) \rightarrow famous(X), \\ & \quad \sigma_4 : rock\_singer(X) \rightarrow can\_sing(X)\} \end{array}\right\}$$

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.

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, 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 (Arenas, Bertossi, and Chomicki 1999; Lembo et al. 2010; Lukasiewicz, Martinez, and Simari 2012), 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 and these constraints are not subject to changes over time. We argue that it is also important to identify and separate the sources of conflicts in Datalog$^{\pm}$ ontologies. In the previous section we defined inconsistency of a Datalog$^{\pm}$ ontology based on the lack of models. From an operational point of view, conflicts appear in a Datalog$^{\pm}$ ontology whenever a NC or an EGD is violated, that is, whenever the body of one such constraint can be mapped to either atoms in D or atoms that can be obtained from D by the application of the TGDs in $\Sigma_T \subseteq \Sigma$. Besides these conflicts, we will also 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. Note that in this case clearly the data in the database instance is not the problem, as any database in which these TGDs are applicable will inevitable produce an inconsistent ontology. 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* (Flouris et al. 2006; Qi and Hunter 2007). Incoherence can be particularly important when combining multiple ontologies since the constraints imposed by each one of them over the data could (possibly) represent conflicting modellings of the application at hand. Clearly, the notions of incoherence and inconsistency are highly related; in fact, Flouris *et al.* (2006) establish a relation between incoherence and inconsistency, considering the former as a particular form of the latter.

Our proposed notion of incoherence states that given a set of incoherent constraints Σ 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 $\Sigma_T \subseteq \Sigma$ are applicable in D. 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 those 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.,* (Konieczny and Pérez 2002; Delgrande and Jin 2012; Arenas, Bertossi, and Chomicki 1999)). 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* (Arenas, Bertossi, and Chomicki 1999) (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 Σ renders inconsistent any ontology whose database instance is such that the TGDs are applicable; in particular cases this may lead to the removal 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.

Before formalizing the notion of *incoherence* that we use in our Datalog$^{\pm}$ setting we need to identify the set of atoms 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) *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:

$$\Sigma_{_T} = \{\sigma_1 : supervises(X,Y) \rightarrow supervisor(X),\\ \sigma_2 : supervisor(X) \wedge take\_decisions(X) \rightarrow \\ leads\_department(X,D),\\ \sigma_3 : employee(X) \rightarrow works\_in(X,D)\}$$

First, let us consider the set $A_1 = \{supervises(walter, jesse), take\_decisions(walter), employee(jesse)\}$. 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 $supervisor(walter)$, which together with $take\_decisions(walter)$ triggers σ_2).

However, the set $A_2 = \{supervises(walter, jesse), take\_decisions(gus)\}$ is not relevant to $\Sigma_{_T}$. Note that even though σ_1 is applicable to A_2, the TGDs σ_2 and σ_3 are never applied in $chase(A_2, \Sigma_{_T})$, since the atoms in their bodies are never generated in $chase(A_2, \Sigma_{_T})$. For instance, consider the TGD $\sigma_2 \in \Sigma_{_T}$. In the chase of $\Sigma_{_T}$ over D we create the atom $supervisor(walter)$, but nevertheless we still cannot trigger σ_2 since we do not have and cannot generate the atom $take\_decisions(walter)$, and the atom $take\_decisions(gus)$ that is already in A_2 does not match the constant value.

We now present the notion of coherence for Datalog$^{\pm}$, which adapts the one introduced by Flouris *et al.* for DLs (Flouris et al. 2006). 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)*** *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, $\Sigma_{_T}$ 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 unsatiasfiable sets of TGDs in $\Sigma_{_T}$ 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.*

$$\Sigma^1_{_{NC}} = \{\tau : risky\_job(P) \wedge unstable(P) \rightarrow \bot\}$$

$$\Sigma^1_{_T} = \{\sigma_1 : dangerous\_work(W) \wedge works\_in(W,P) \rightarrow risky\_job(P),\\ \sigma_2 : in\_therapy(P) \rightarrow unstable(P)\}$$

The set $\Sigma^1_{_T}$ is a satisfiable set of TGDs, and even though the simultaneous application of σ_1 and σ_2 may violate some formula in $\Sigma^1_{_{NC}} \cup \Sigma^1_{_E}$, that does not hold for every relevant set of atoms. Consider as an example the relevant set $D_1 = \{dangerous\_work(police), works\_in(police, marty), in\_therapy(rust)\}$; D_1 is a relevant set for $\Sigma^1_{_T}$, however, as we have that $mods(D_1, \Sigma^1_{_T} \cup \Sigma^1_{_{NC}} \cup \Sigma^1_{_E}) \neq \emptyset$ then $\Sigma^1_{_T}$ is satisfiable.

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

$$\Sigma^2_{_{NC}} = \{\tau_1 : sore\_throat(X) \wedge can\_sing(X) \rightarrow \bot\}$$

$$\Sigma^2_{_T} = \{\sigma_1 : rock\_singer(X) \rightarrow sing\_loud(X),\\ \sigma_2 : sing\_loud(X) \rightarrow sore\_throat(X),\\ \sigma_3 : rock\_singer(X) \rightarrow can\_sing(X)\}$$

The set $\Sigma^2_{_T}$ 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 $rock\_singer(axl)$: we have that the application of $\Sigma^2_{_T}$ over $\{rock\_singer(axl)\}$ causes the violation of τ_1 when considered together with $\Sigma^2_{_T}$, therefore $mods(\{rock\_singer(axl)\}, \Sigma^2_{_T} \cup \Sigma^2_{_{NC}} \cup \Sigma^2_{_E}) = \emptyset$. Note that any *set of relevant atoms will cause the violation of τ_1.*

We are now ready to formally define coherence for a Datalog$^{\pm}$ ontology. Intuitively, an ontology is coherent if there is no subset of their TGDs that is unsatisfiable *w.r.t.* the constraints in the ontology.

Definition 3 (Coherence) *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 $\Sigma_{_T}$ is a set of TGDs, $\Sigma_{_E}$ a set of separable EGDs and $\Sigma_{_{NC}}$ a set of negative constraints. KB is* coherent *iff $\Sigma_{_T}$ 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 $\text{KB}_1 = (D, \Sigma^1_{_T} \cup \Sigma^1_{_{NC}} \cup \Sigma^1_{_E})$ is coherent, while $\text{KB}_2 = (D, \Sigma^2_{_T} \cup \Sigma^2_{_{NC}} \cup \Sigma^2_{_E})$ is incoherent.*

Finally, we look deeper into the relation between incoherence and inconsistency. 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 *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.

Incoherence influence on classic inconsistency-tolerant semantics

We have established the relation between incoherence and inconsistency. As explained, classic inconsistency-tolerant techniques do not account for coherence issues since they assume that such kind of problems will not appear. 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 inconsistency-tolerant techniques to consider incoherence in ontologies as well, since if not treated appropriately an incoherent set of TGDs may lead to the trivial solution of removing every single relevant atom in D (which in the worst case could be the entire database instance). This may be adequate for some particular domains, but 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* (Lembo et al. 2010; Bienvenu and Rosati 2013), and several fragments of Datalog$^{\pm}$ (Lukasiewicz, Martinez, and Simari 2012). In this section we analyze how incoherence influence in several inconsistency-tolerant semantics for ontological languages: AR semantics (Lembo et al. 2010), CAR semantics (Lembo et al. 2010), adn 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 (Lembo et al. 2010), 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 (Arenas, Bertossi, and Chomicki 1999). 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.* (2010), 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.* (2010) 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 $CCL(KB) = \{\alpha \mid \alpha \in \mathcal{H}(\mathcal{L}_\mathcal{R}) \, s.t. \, \exists S \subseteq D \text{ and } mods(S, \Sigma_T \cup \Sigma_E \cup \Sigma_{NC}) \neq \emptyset \text{ and } (S, \Sigma_T) \models \alpha\}$. 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 (Lembo et al. 2010). 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 semantics (*i.e.,* AR or CAR) no atom that is relevant to an unsatisfiable set of TGDs belongs 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.

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.*

In an analogous way 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, from Lemma 1 follows 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* $\not\models_{AR} A$ *and KB* $\not\models_{CAR} A$.

The proof follows from Lemma 1: since a relevant set of atoms does not belong to any repair then it cannot be part of the answers of the AR and CAR semantics. 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* $A \subseteq D$ *such that* $a \in A$ *and* A *is a minimal set of TGDs relevant to some* $U \in \mathcal{U}$(*KB*) *then* $\mathcal{A}_{AR} = \emptyset$ *and* $\mathcal{A}_{CAR} = \emptyset$.

Since they follow from Proposition 1, both Proposition 2 and Corollary 1 can be straightforwardly extended to other repair based inconsistency-tolerant semantics such as ICAR and ICR (Lembo et al. 2010).

Example 7 *Consider once again KB in Example 1, and the atom* $a_2 : rock\_singer(axl)$ *in* D. *Such atom 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* $\not\models_{AR} rock\_singer(axl)$ *and KB* $\not\models_{CAR} rock\_singer(axl)$. *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.,* $mods((rock\_singer(axl), \Sigma)) = \emptyset$.

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 (Besnard and Hunter 2008; Rahwan and Simari 2009).

Definition 4 (Incoherence-tolerant semantics) *Let KB* $= (D, \Sigma)$ *be a Datalog*$^{\pm}$ *ontology where* $\Sigma = \Sigma_{_T} \cup \Sigma_{_E} \cup \Sigma_{_{NC}}$. *A query answering semantics* S *is said to be* tolerant to incoherence *(or incoherency-tolerant) iff 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.*

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 (Martinez et al. 2014)), and then we move on to show the behaviour of this semantics in the presence of incoherence.

Defeasible Datalog$^{\pm}$ (Martinez et al. 2014) 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 em defeasible atoms and *defeasible TGDs*; thus, a Defeasible Datalog$^{\pm}$ ontology contains both (classical) strict knowledge and defeasible knowldge. 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 (García and Simari 2004), 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 reports indicates 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.*

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

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.* (2012). 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 $KB \sim a$. . 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 (Martinez et al. 2014).

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

$$\partial = \big[rock\_singer(axl),$$
$$rock\_singer(X) \succ sing\_loud(X),$$
$$sing\_loud(axl),$$
$$sing\_loud(X) \rightarrow sore\_throat(X),$$
$$sore\_throat(axl)\big]$$

Then, we have that $KB \vdash rock\_singer(axl)$ *and that* $KB \sim sore\_throat(axl)$.

Classical query answering in defeasible Datalog$^{\pm}$ ontologies is equivalent to query answering in Datalog$^{\pm}$ ontologies.

Proposition 3 ((Martinez et al. 2014)) *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 $KB \sim L$.*

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 a *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.

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.*

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 (1992), 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. We have that* $\langle sore\_throat(axl), \partial \rangle$ *is an argument in* $\mathbb{A}_{KB}$*. Figure 1 shows the argument.*

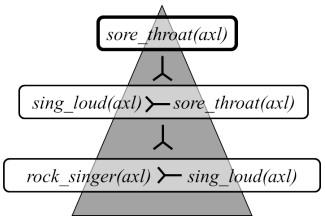

Figure 1: An argument for $sore\_throat(axl)$.

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 (García and Simari 2004). 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.*

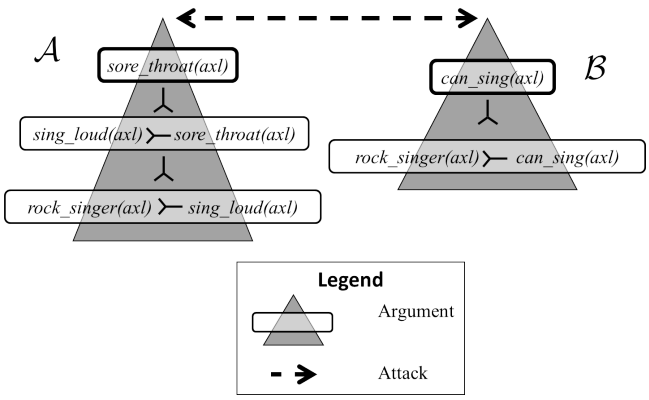

Figure 2: Attack between arguments.

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 (García and Simari 2004). 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 (García and Simari 2004), which states that an argumentation line has to be finite, and no argument is a subargument 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 (García and Simari 2004), and we adopt the notion of coherent dialectical tree from (Martinez, García, and Simari 2012), 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.

Definition 10 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 Figure 3.*

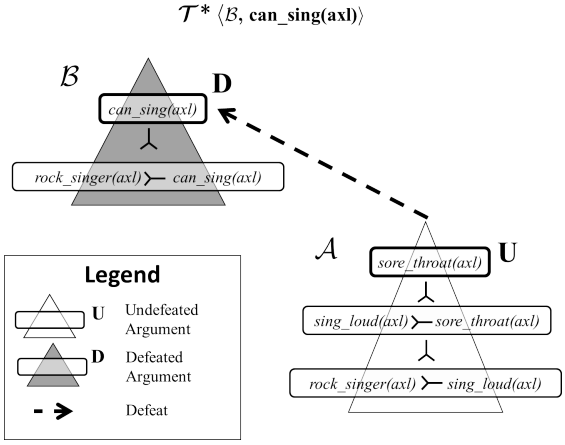

Figure 3: A labelled dialectical tree for atom $can\_sing(axl)$.

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.

In Definition 10 we specify a semantics based on the use of argumentative inference. From now on we denote such semantics as $\mathbf{D}^2$ (**D**efeasible **D**atalog$^{\pm}$). Such semantics relies on the transformation of classic Datalog$^{\pm}$ ontologies to defeasible ones and then obtaining answers from the transformed one. So, we begin by establishing 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 $\Sigma_D = \{\Upsilon(\mathbf{X}, \mathbf{Y}) \succ \exists \mathbf{Z}\, \Psi(\mathbf{X}, \mathbf{Z}) \mid \Upsilon(\mathbf{X}, \mathbf{Y}) \rightarrow \exists \mathbf{Z}\, \Psi(\mathbf{X}, \mathbf{Z})\}$.*

Next, we define the set of answers in $\mathbf{D}^2$ for an atomic query. 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 (Answers in $\mathbf{D}^2$) *Let KB be a Datalog$^{\pm}$ ontology, KB$' = \mathcal{D}$(KB) its defeasible transformation, Q a query and $\succ$ a comparison criterion. Then, an atom L is an* answer for Q from *KB* under $\mathbf{D}^2$*, denoted KB $\models^2_{\boldsymbol{D}^2_{\succ}} L$, iff 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.

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 $KB \models_{\boldsymbol{D}^2_{\succ}} L$.*

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

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, $F = \emptyset$, $D = \{can\_sing(simone), rock\_singer(axl), sing\_loud(ronnie), has\_fans(ronnie), manage(band_1, richard)\}$, and*

$$\begin{aligned}\Sigma_D = \{&rock\_singer(X) \succ sing\_loud(X),\\ &sing\_loud(X) \succ sore\_throat(X),\\ &has\_fans(X) \succ famous(X),\\ &rock\_singer(X) \succ can\_sing(X)\}\end{aligned}$$

Here, we have the dialectical tree with argument $\langle [rock\_singer(axl)], rock\_singer(axl) \rangle$ as its undefeated root, since no counterargument for it can be built (Figure 4).

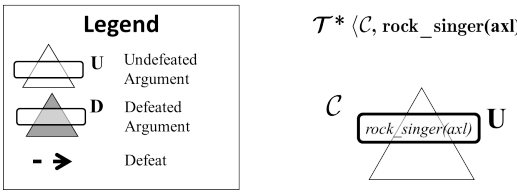

Figure 4: A labelled dialectical tree for atom $rock\_singer(axl)$.

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

Note that in Example 14 the atom $rock\_singer(axl)$ is warranted under **any** criterion comparison $\succ$, and thus we have not needed to perform any restriction on the criterion.

Conclusions

Incoherence is an important problem in knowledge representation and reasoning, specially when integrating different sources of information. Nevertheless, 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.

In this work we have introduced the concept of incoherence for Datalog$^{\pm}$ ontologies, relating it 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. We have shown how incoherence affects classic inconsistency-tolerant semantics to the point that for some incoherent ontologies these semantics may produce no useful answer. Finally, we have introduced the concept of incoherency-tolerant semantics, and shown a particular semantics satisfying that property. Nevertheless, it is important to remark that our definition of incoherency-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.* (2009) where dialogue games between agents are used to solve queries under Description Logics ontologies that can be incoherent.

References

Arenas, M.; Bertossi, L. E.; and Chomicki, J. 1999. Consistent query answers in inconsistent databases. In *Proc. of PODS*, 68–79.

Besnard, P., and Hunter, A. 2008. *Elements of Argumentation*. MIT Press.

Bienvenu, M., and Rosati, R. 2013. Tractable approximations of consistent query answering for robust ontology-based data access. In *Proc. of IJCAI*.

Bienvenu, M. 2012. On the complexity of consistent query answering in the presence of simple ontologies. In *Proc. of AAAI*.

Black, E.; Hunter, A.; and Pan, J. Z. 2009. An argument-based approach to using multiple ontologies. In *SUM*, 68–79.

Calì, A.; Gottlob, G.; and Lukasiewicz, T. 2012a. A general Datalog-based framework for tractable query answering over ontologies. In *J. Web Sem.*, volume 14, 57–83.

Calì, A.; Gottlob, G.; and Lukasiewicz, T. 2012b. A general Datalog-based framework for tractable query answering over ontologies. *J. of Web Semant.* 14:57–83.

Calì, A.; Lembo, D.; and Rosati, R. 2003. On the decidability and complexity of query answering over inconsistent and incomplete databases. In *Proc. of PODS 2003*, 260–271. ACM.

Delgrande, J. P., and Jin, Y. 2012. Parallel belief revision: Revising by sets of formulas. *Artif. Intell.* 176(1):2223–2245.

Flouris, G.; Huang, Z.; Pan, J. Z.; Plexousakis, D.; and Wache, H. 2006. Inconsistencies, negations and changes in ontologies. In *AAAI*, 1295–1300. AAAI Press.

García, A. J., and Simari, G. R. 2004. Defeasible logic programming: An argumentative approach. *TPLP* 4(1-2):95–138.

Konieczny, S., and Pérez, R. P. 2002. Merging information under constraints: A logical framework. *J. Log. Comput.* 12(5):773–808.

Lembo, D.; Lenzerini, M.; Rosati, R.; Ruzzi, M.; and Savo, D. F. 2010. Inconsistency-tolerant semantics for description logics. In *Proc. of RR*, 103–117.

Lukasiewicz, T.; Martinez, M. V.; and Simari, G. I. 2012. Inconsistency handling in Datalog+/– ontologies. In *Proc. of ECAI*, 558–563.

Martinez, M. V.; Deagustini, C. A. D.; Falappa, M. A.; and Simari, G. R. 2014. Inconsistency-tolerant reasoning in datalog$\pm$ ontologies via an argumentative semantics. In *proc. of IBERAMIA 2014*, 15–27.

Martinez, M. V.; García, A. J.; and Simari, G. R. 2012. On the use of presumptions in structured defeasible reasoning. In *Proc. of COMMA*, 185–196.

Qi, G., and Hunter, A. 2007. Measuring incoherence in description logic-based ontologies. In *ISWC/ASWC*, 381–394.

Rahwan, I., and Simari, G. R. 2009. *Argumentation in Artificial Intelligence*. Springer.

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