Proceedings of the 22nd International Conference on Principles of Knowledge Representation and Reasoning
Main Track

A Rule-Based Approach to Specifying Preferences over Conflicting Facts and Querying Inconsistent Knowledge Bases

Meghyn Bienvenu[1] , **Camille Bourgaux**[2] , **Katsumi Inoue**[3] , **Robin Jean**[1]
[1]Univ. Bordeaux, CNRS, Bordeaux INP, LaBRI, UMR 5800, Talence, France
[2]DI ENS, ENS, CNRS, PSL University & Inria, Paris, France
[3]National Institute of Informatics, Toyo, Japan
{meghyn.bienvenu, robin.jean}@u-bordeaux.fr, camille.bourgaux@ens.fr, inoue@nii.ac.jp

Abstract

Repair-based semantics have been extensively studied as a means of obtaining meaningful answers to queries posed over inconsistent knowledge bases (KBs). While several works have considered how to exploit a priority relation between facts to select optimal repairs, the question of how to specify such preferences remains largely unaddressed. This motivates us to introduce a declarative rule-based framework for specifying and computing a priority relation between conflicting facts. As the expressed preferences may contain undesirable cycles, we consider the problem of determining when a set of preference rules always yields an acyclic relation, and we also explore a pragmatic approach that extracts an acyclic relation by applying various cycle removal techniques. Towards an end-to-end system for querying inconsistent KBs, we present a preliminary implementation and experimental evaluation of the framework, which employs answer set programming to evaluate the preference rules, apply the desired cycle resolution techniques to obtain a priority relation, and answer queries under prioritized-repair semantics.

1 Introduction

Inconsistency-tolerant semantics are a well-established approach to querying data inconsistent w.r.t. some constraints, both in the relational database and ontology-mediated query answering settings (cf. recent surveys (Bertossi 2019; Bienvenu 2020)). Such semantics typically rely on *(subset) repairs*, defined as maximal subsets of the data consistent w.r.t. the constraints. The most well-known, called the *AR* semantics in the KR community and corresponding to consistent query answering in the database community, considers that a Boolean query holds true if it holds in every repair. The more cautious *IAR* semantics amounts to querying the repairs intersection, and the less cautious *brave* semantics only requires that the query holds in some repair.

Since an inconsistent dataset may have a lot of repairs, several notions of preferred repairs have been proposed in the literature, to restrict the possible worlds considered to answer queries, for example by taking into account some information about the reliability of the data (Lopatenko and Bertossi 2007; Du, Qi, and Shen 2013; Bienvenu, Bourgaux, and Goasdoué 2014; Calautti et al. 2022; Lukasiewicz, Malizia, and Molinaro 2023). In particular, since its introduction by Staworko, Chomicki, and Marcinkowski (2012), the framework of *prioritized databases*, in which a *priority relation* between conflicting facts is used to define *optimal repairs*, has attracted attention, with numerous theoretical results (Kimelfeld, Livshits, and Peterfreund 2017; Kimelfeld, Livshits, and Peterfreund 2020; Bienvenu and Bourgaux 2020; Bienvenu and Bourgaux 2023), and an implementation (Bienvenu and Bourgaux 2022). However, the crucial question of obtaining the priority relation was left unaddressed, preventing the adoption of this framework in practice. Indeed, it is not realistic to expect users to manually input a binary relation between the facts.

In our work, we tackle this challenge by developing a general rule-based approach to specifying preferences over conflicting facts. After introducing the background on optimal repair-based inconsistency-tolerant semantics in Section 2, we present in Section 3 our framework for specifying a priority relation between conflicting facts via preference rules. A distinguishing feature of our work is that we address the issue of cycles in the expressed preferences, first by investigating the problem of deciding whether a given set of preference rules is guaranteed to produce an acyclic relation, and second by providing a pragmatic approach that uses cycle removal strategies to extract an acyclic relation. Section 4 describes our implementation which employs answer set programming to evaluate the preference rules, apply the desired cycle resolution techniques to obtain a priority relation, and answer queries under optimal repair-based semantics. Finally, we present in Section 5 a preliminary experimental evaluation of the overall framework. Sections 6 and 7 discuss related and future work. Proofs and additional details on the implementation and experiments are provided in (Bienvenu et al. 2025). All materials to reproduce the experiments (e.g. datasets, logic programs) are available at https://github.com/rjean007/PreferenceRules-ASP

2 Preliminaries

We recall the framework of inconsistency-tolerant querying of prioritized knowledge bases, as considered in (Bienvenu and Bourgaux 2022; Bourgaux 2024). We assume that readers are familiar with first-order logic and consider three disjoint sets of predicates $\mathbf{P}$, constants $\mathbf{C}$ and variables $\mathbf{V}$. As usual, each predicate has an associated arity $n \geq 1$, and we shall use $\mathbf{P}_n$ for the set of the n-ary predicates in $\mathbf{P}$.

Knowledge bases A *knowledge base (KB)* $\mathcal{K} = (\mathcal{D}, \mathcal{T})$ con-

sists of a *dataset* $\mathcal{D}$ and a *logical theory* $\mathcal{T}$: $\mathcal{D}$ is a finite set of *facts* of the form $P(c_1, \ldots, c_n)$ with $P \in \mathbf{P}_n$, $c_i \in \mathbf{C}$ for $1 \leq i \leq n$, and $\mathcal{T}$ is a finite set of first-order logic (FOL) sentences built from $\mathbf{P}$, $\mathbf{C}$ and $\mathbf{V}$. We denote by $\mathsf{sig}(\mathcal{K})$ and $\mathsf{const}(\mathcal{K})$ (resp. $\mathsf{sig}(\mathcal{D})$, $\mathsf{const}(\mathcal{D})$) the set of predicates and constants that occur in $\mathcal{K}$ (resp. in $\mathcal{D}$). A KB $\mathcal{K} = (\mathcal{D}, \mathcal{T})$ is *consistent*, and $\mathcal{D}$ is called $\mathcal{T}$*-consistent*, if $\mathcal{D} \cup \mathcal{T}$ has some model. Otherwise, $\mathcal{K}$ is *inconsistent*, denoted $\mathcal{K} \models \bot$.

Typically, $\mathcal{T}$ will be either an *ontology* (formulated in some description logic or decidable class of existential rules) or a set of *database constraints*. In particular, we consider:

- *Description logics of the DL-Lite family* (Calvanese et al. 2007), such as DL-Lite$_{\mathsf{core}}$, whose axioms take the form $\mathsf{B}_1 \sqsubseteq (\neg)\mathsf{B}_2$, with $\mathsf{B}_i \in \mathbf{P}_1 \cup \{\exists \mathsf{R}, \exists \mathsf{R}^- \mid \mathsf{R} \in \mathbf{P}_2\}$.
- *Denial constraints* of the form $\alpha_1 \wedge \ldots \wedge \alpha_n \rightarrow \bot$, where each α_i is a relational or inequality atom, which includes in particular *functional dependencies* (FDs) such as $P(x, y, z) \wedge P(x, y, z') \wedge z \neq z' \rightarrow \bot$.

Queries A *conjunctive query* (CQ) is a conjunction of relational atoms $P(t_1, \ldots, t_n)$ ($P \in \mathbf{P}_n$, $t_i \in \mathbf{C} \cup \mathbf{V}$), where some variables may be existentially quantified. Given a query $q(\vec{x})$, with free variables $\vec{x}$, and a tuple of constants $\vec{a}$ such that $|\vec{a}| = |\vec{x}|$, $q(\vec{a})$ denotes the first-order sentence obtained by replacing each variable in $\vec{x}$ by the corresponding constant in $\vec{a}$. A *(certain) answer* to $q(\vec{x})$ over $\mathcal{K}$ is a tuple $\vec{a} \in \mathsf{const}(\mathcal{K})^{|\vec{x}|}$ such that $q(\vec{a})$ holds in every model of $\mathcal{K}$, denoted $\mathcal{K} \models q(\vec{a})$. A *cause* for $q(\vec{a})$ w.r.t. $\mathcal{K} = (\mathcal{D}, \mathcal{T})$ is an inclusion-minimal $\mathcal{T}$-consistent subset $\mathcal{C} \subseteq \mathcal{D}$ such that $(\mathcal{C}, \mathcal{T}) \models q(\vec{a})$. The set of causes for $q(\vec{a})$ w.r.t. $\mathcal{K}$ is denoted by $\mathit{Causes}(q(\vec{a}), \mathcal{K})$.

Conflicts and repairs A *conflict* of $\mathcal{K} = (\mathcal{D}, \mathcal{T})$ is an inclusion-minimal subset $\mathcal{D}' \subseteq \mathcal{D}$ such that $(\mathcal{D}', \mathcal{T}) \models \bot$. The set of conflicts of $\mathcal{K}$ is denoted $\mathit{Conf}(\mathcal{K})$. A *(subset) repair* of $\mathcal{K}$ is an inclusion-maximal subset $\mathcal{R} \subseteq \mathcal{D}$ such that $(\mathcal{R}, \mathcal{T}) \not\models \bot$. The set of repairs of $\mathcal{K}$ is denoted $\mathit{SRep}(\mathcal{K})$.

Prioritized KBs A *priority relation* $\succ$ for a KB $\mathcal{K} = (\mathcal{D}, \mathcal{T})$ is an acyclic[1] binary relation over the facts of $\mathcal{D}$ such that if $\alpha \succ \beta$, then there exists $\mathcal{C} \in \mathit{Conf}(\mathcal{K})$ such that $\{\alpha, \beta\} \subseteq \mathcal{C}$. It is *total* if for every pair $\alpha \neq \beta$ such that $\{\alpha, \beta\} \subseteq \mathcal{C}$ for some $\mathcal{C} \in \mathit{Conf}(\mathcal{K})$, either $\alpha \succ \beta$ or $\beta \succ \alpha$. A *completion* of $\succ$ is a total priority relation $\succ' \supseteq \succ$. A *prioritized KB* $\mathcal{K}_\succ$ is a KB $\mathcal{K}$ with a priority relation $\succ$ for $\mathcal{K}$.

Priority relations are used to select optimal repairs. We recall the notions of Pareto- and completion-optimal repairs[2] from (Staworko, Chomicki, and Marcinkowski 2012):

Definition 1. *Consider a prioritized KB $\mathcal{K}_\succ$ with $\mathcal{K} = (\mathcal{D}, \mathcal{T})$, and let $\mathcal{R} \in \mathit{SRep}(\mathcal{K})$. A* Pareto improvement *of $\mathcal{R}$ is a $\mathcal{T}$-consistent $\mathcal{B} \subseteq \mathcal{D}$ such that there is $\beta \in \mathcal{B} \setminus \mathcal{R}$ with $\beta \succ \alpha$ for every $\alpha \in \mathcal{R} \setminus \mathcal{B}$. The repair $\mathcal{R}$ is a* Pareto-optimal repair *of $\mathcal{K}_\succ$ if there is no Pareto improvement of $\mathcal{R}$, and a* completion-optimal repair *of $\mathcal{K}_\succ$ if $\mathcal{R}$ is a Pareto-optimal repair of $\mathcal{K}_{\succ'}$, for some completion $\succ'$ of $\succ$. We denote by $\mathit{PRep}(\mathcal{K}_\succ)$ and $\mathit{CRep}(\mathcal{K}_\succ)$ the sets of Pareto- and completion-optimal repairs.*

The following relation between optimal repairs is known:

$$\mathit{CRep}(\mathcal{K}_\succ) \subseteq \mathit{PRep}(\mathcal{K}_\succ) \subseteq \mathit{SRep}(\mathcal{K}).$$

Repair-based semantics We next recall three prominent inconsistency-tolerant semantics (brave, AR, and IAR), parameterized by the considered type of repair:

Definition 2. *Fix $X \in \{S, P, C\}$ and consider a prioritized KB $\mathcal{K}_\succ$ with $\mathcal{K} = (\mathcal{D}, \mathcal{T})$, query $q(\vec{x})$, and tuple $\vec{a} \in \mathsf{const}(\mathcal{K})^{|\vec{x}|}$. Then $\vec{a}$ is an answer to q over $\mathcal{K}_\succ$*

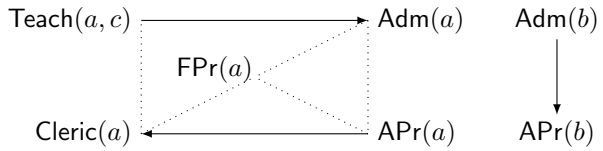
- *under* X-brave semantics*, denoted $\mathcal{K}_\succ \models^X_{\text{brave}} q(\vec{a})$, if $(\mathcal{R}, \mathcal{T}) \models q(\vec{a})$ for some $\mathcal{R} \in \mathit{XRep}(\mathcal{K}_\succ)$*
- *under* X-AR semantics*, denoted $\mathcal{K}_\succ \models^X_{\text{AR}} q(\vec{a})$, if $(\mathcal{R}, \mathcal{T}) \models q(\vec{a})$ for every $\mathcal{R} \in \mathit{XRep}(\mathcal{K}_\succ)$*
- *under* X-IAR semantics*, denoted $\mathcal{K}_\succ \models^X_{\text{IAR}} q(\vec{a})$, if $(\mathcal{B}, \mathcal{T}) \models q(\vec{a})$ where $\mathcal{B} = \bigcap_{\mathcal{R} \in \mathit{XRep}(\mathcal{K}_\succ)} \mathcal{R}$*

It is known that $\mathcal{K}_\succ \models^X_{\text{IAR}} q \Rightarrow \mathcal{K}_\succ \models^X_{\text{AR}} q \Rightarrow \mathcal{K}_\succ \models^X_{\text{brave}} q$.

Example 1. *Our running example considers a DL knowledge base $\mathcal{K}_{\mathsf{ex}} = (\mathcal{D}_{\mathsf{ex}}, \mathcal{T}_{\mathsf{ex}})$ about a university. The ontology expresses that associate and full professors (*APr *and* FPr*) are faculty members (*Fac*) and clerical staff workers (*Cleric*) are administrative staff workers (*Adm*). Moreover, one cannot be both an associate and a full professor, or an administrative staff worker and a faculty member.*

$$\begin{aligned}\mathcal{T}_{\mathsf{ex}} = \{&\mathsf{APr} \sqsubseteq \mathsf{Fac}, \mathsf{FPr} \sqsubseteq \mathsf{Fac}, \mathsf{APr} \sqsubseteq \neg\mathsf{FPr},\\ &\exists\mathsf{Teach} \sqsubseteq \mathsf{Fac}, \mathsf{Cleric} \sqsubseteq \mathsf{Adm}, \mathsf{Adm} \sqsubseteq \neg\mathsf{Fac}\}\\ \mathcal{D}_{\mathsf{ex}} = \{&\mathsf{APr}(a), \mathsf{FPr}(a), \mathsf{Cleric}(a), \mathsf{Adm}(a), \mathsf{Teach}(a, c),\\ &\mathsf{Adm}(b), \mathsf{APr}(b)\}\end{aligned}$$

The picture below represents the conflicts of $\mathcal{K}$ and a priority relation $\succ$: an arrow from α to β indicates that $\alpha \succ \beta$ and a dotted line indicates that $\{\alpha, \beta\} \in \mathit{Conf}(\mathcal{K})$ without priority between α and β.



There are six repairs:

$$\begin{aligned}\mathcal{R}_1 &= \{\mathsf{APr}(a), \mathsf{Teach}(a, c), \mathsf{Adm}(b)\},\\ \mathcal{R}_2 &= \{\mathsf{FPr}(a), \mathsf{Teach}(a, c), \mathsf{Adm}(b)\},\\ \mathcal{R}_3 &= \{\mathsf{Cleric}(a), \mathsf{Adm}(a), \mathsf{Adm}(b)\},\\ \mathcal{R}_4 &= \{\mathsf{APr}(a), \mathsf{Teach}(a, c), \mathsf{APr}(b)\},\\ \mathcal{R}_5 &= \{\mathsf{FPr}(a), \mathsf{Teach}(a, c), \mathsf{APr}(b)\},\\ \mathcal{R}_6 &= \{\mathsf{Cleric}(a), \mathsf{Adm}(a), \mathsf{APr}(b)\},\end{aligned}$$

and one can check that $\mathit{PRep}(\mathcal{K}) = \{\mathcal{R}_1, \mathcal{R}_2, \mathcal{R}_3\}$ while $\mathit{CRep}(\mathcal{K}) = \{\mathcal{R}_1, \mathcal{R}_2\}$, so, e.g., $\mathcal{K} \models^P_{\text{IAR}} \mathsf{Adm}(b)$, $\mathcal{K} \not\models^P_{\text{brave}} \mathsf{APr}(b)$, $\mathcal{K} \not\models^P_{\text{AR}} \mathsf{FPr}(a)$, $\mathcal{K} \models^C_{\text{IAR}} \mathsf{Fac}(a)$, ...

Data complexity When considering the complexity of tasks involving an input dataset $\mathcal{D}$, we always use *data complexity*,

[1] In line with existing work on prioritized repairs, we do not require priority relations to be transitive.

[2] Staworko et al. (2012) further introduces a third notion of globally-optimal repair, see Section 7 for discussion.

where the sizes of the logical theory $\mathcal{T}$ and query $q(\vec{x})$ are assumed to be fixed. Theorems 1 and 2 summarize known upper and lower bounds in the database and ontology settings (Staworko, Chomicki, and Marcinkowski 2012; Bienvenu and Bourgaux 2020; Bienvenu and Bourgaux 2022; Rosati 2011; Bienvenu and Rosati 2013).

Theorem 1. *Let $\mathcal{L}$ be an FOL fragment for which KB consistency and query entailment are in* PTIME*. Query entailment for $\mathcal{L}$ KBs and $X \in \{S, P, C\}$ is in* NP *under X-brave semantics, and in co*NP *under X-AR and X-IAR semantics.*

Theorem 2. *Let $\mathcal{L}$ be any FOL fragment that extends DL-Lite$_{\mathsf{core}}$, $\mathcal{EL}_\bot$, or FDs. Query entailment for $\mathcal{L}$ KBs is* NP*-hard under X-brave semantics ($X \in \{P, C\}$), co*NP*-hard under X-AR semantics ($X \in \{S, P, C\}$), co*NP*-hard under X-IAR semantics ($X \in \{P, C\}$).*

3 Specifying Priority Relations via Rules

The optimal repair semantics recalled in Section 2 suppose that we have a priority relation between the facts. However, the question of how to conveniently specify the priority relation has not yet been addressed in the literature. This will be the topic of the present section, which presents a declarative rule-based framework for specifying priority relations.

3.1 Preference Rules

We propose to use *preference rules* to state that, when some conditions are satisfied, a fact should generally be preferred to another fact. The rule conditions may naturally refer to the presence (or absence) of facts in the dataset. However, typically we may also want to exploit information about the facts themselves, provided in metadata, e.g. to compare facts based upon the date they were added.

We now introduce some terminology and notation in order to be able to refer to metadata in rule conditions. First, we fix a subset $\mathbf{P_M} \subsetneq \mathbf{P}$ of metadata predicates and $\mathbf{C_{ID}} \subsetneq \mathbf{C}$ of fact identifiers, assumed distinct from the predicates and constants used in the considered dataset. We assume that each n-ary predicate $P \in \mathbf{P_M}$ has an associated set of $\mathbf{C_{ID}}$-positions $\mathsf{pos}_{\mathbf{ID}}(P) \subseteq \{1, \ldots, n\}$, indicating which positions of P contain constants from $\mathbf{C_{ID}}$. Given a KB $\mathcal{K} = (\mathcal{D}, \mathcal{T})$, a *meta-database* for $\mathcal{K}$ is a pair $\mathcal{M} = (\mathbf{id}, \mathcal{F})$, where $\mathbf{id}$ is an *injective function* that associates to each fact $\alpha \in \mathcal{D}$ an identifier $\mathbf{id}(\alpha)$ from $\mathbf{C_{ID}}$, and $\mathcal{F}$ is a dataset with $\mathsf{sig}(\mathcal{F}) \subseteq \mathbf{P_M}$ satisfying the following conditions:

- if $P(c_1, \ldots, c_n) \in \mathcal{F}$, then $c_j \in \mathbf{C_{ID}}$ iff $j \in \mathsf{pos}_{\mathbf{ID}}(P)$
- if $c \in \mathsf{const}(\mathcal{F}) \cap \mathbf{C_{ID}}$, then $c = \mathbf{id}(\alpha)$ for some $\alpha \in \mathcal{D}$

Intuitively, $\mathcal{F}$ provides information about the facts in $\mathcal{D}$ by referring to their identifiers defined by $\mathbf{id}$. Every identifier in $\mathcal{F}$ must designate a unique fact in $\mathcal{D}$, but it is not required that $\mathcal{F}$ contains information about all facts in $\mathcal{D}$.

Example 2. *We associate to the KB $\mathcal{K}_{\mathsf{ex}}$ from Example 1 the meta-database $\mathcal{M}_{\mathsf{ex}} = (\mathbf{id}_{\mathsf{ex}}, \mathcal{F}_{\mathsf{ex}})$, which records the year that facts have been added to the university database:*

$\mathbf{id}_{\mathsf{ex}}(\mathsf{APr}(a)) = 1$, $\mathbf{id}_{\mathsf{ex}}(\mathsf{FPr}(a)) = 2$, $\mathbf{id}_{\mathsf{ex}}(\mathsf{Cleric}(a)) = 3$, $\mathbf{id}_{\mathsf{ex}}(\mathsf{Adm}(a)) = 4$, $\mathbf{id}_{\mathsf{ex}}(\mathsf{Teach}(a, c)) = 5$,

$\mathbf{id}_{\mathsf{ex}}(\mathsf{Adm}(b)) = 6$, $\mathbf{id}_{\mathsf{ex}}(\mathsf{APr}(b)) = 7$

$$\mathcal{F} = \{\mathsf{Date}(1, 2014), \mathsf{Date}(2, 2025), \mathsf{Date}(3, 2013), \\ <(2013, 2014), <(2013, 2025), <(2014, 2025)\}.$$

Remark 1. *To simplify the presentation, we employ a meta-database predicate $<$ to compare years. However, such comparison facts could be avoided by extending the definition of meta-databases to allow for built-in comparison predicates for different datatypes and adding further typing constraints on predicate positions.*

We now formulate a general definition of preference rules, which are evaluated over a KB and meta-database:

Definition 3. *A* preference rule *σ over $S \subseteq \mathbf{P}$ takes the form*

$$\mathsf{Cond}(x_1, x_2) \rightarrow \mathsf{pref}(x_1, x_2)$$

where $\mathsf{pref} \notin S$ is a special predicate (assumed not to occur in KBs nor meta-databases) and $\mathsf{Cond}(x_1, x_2)$ is an expression whose predicate symbols are drawn from S and whose two distinguished free variables x_1, x_2 occur only in $\mathbf{C_{ID}}$-positions of relational atoms over $S \cap \mathbf{P_M}$ or in equality atoms of the form $x_i = \mathbf{id}(P(\vec{t}))$. We call $\mathsf{Cond}(x_1, x_2)$ the body *of σ and $\mathsf{pref}(x_1, x_2)$ its* head*. A preference rule language is a set of preferences rules (intuitively, stipulating the allowed syntax of rule bodies).*

The semantics of preference rule languages is defined using evaluation functions. An evaluation function for a preference rule language *$\mathcal{PL}$ is a function $eval$ that associates* true *or* false *to every KB $\mathcal{K} = (\mathcal{D}, \mathcal{T})$ and associated meta-database $\mathcal{M} = (\mathbf{id}, \mathcal{F})$, preference rule $\mathsf{Cond}(x_1, x_2) \rightarrow \mathsf{pref}(x_1, x_2) \in \mathcal{PL}$, and pair of constants $(id_1, id_2) \in \{(\mathbf{id}(\alpha), \mathbf{id}(\beta)) \mid \alpha, \beta \in \mathcal{D}\}$. We denote by $(\mathcal{K}, \mathcal{M}) \models \mathsf{Cond}(id_1, id_2)$ the fact that $eval(\mathcal{K}, \mathcal{M}, \mathsf{Cond}(x_1, x_2), id_1, id_2) = \mathsf{true}$ and say that $\mathsf{pref}(id_1, id_2)$ is* induced by *σ over $(\mathcal{K}, \mathcal{M})$ (w.r.t. $eval$) if σ has body $\mathsf{Cond}(x_1, x_2)$ and $(\mathcal{K}, \mathcal{M}) \models \mathsf{Cond}(id_1, id_2)$. Given a set Σ of preference rules with $\Sigma \subseteq \mathcal{PL}$ and an evaluation function $eval$ for $\mathcal{PL}$, we denote by $\Sigma(\mathcal{K}, \mathcal{M})$ the set of all $\mathsf{pref}(id_1, id_2)$ induced by some $\sigma \in \Sigma$ over $(\mathcal{K}, \mathcal{M})$.*

Observe that the restrictions on the variables x_1, x_2 serve to ensure head variables are mapped to fact identifiers. Aside from this restriction, we have the preceding definition very generic to encompass many different settings. The following example and definition illustrate how our framework can be instantiated, by giving a concrete preference rule language.

Example 3. *Let Σ_{ex} contain three preference rules for the KB $\mathcal{K}_{\mathsf{ex}}$ and meta-database $\mathcal{M}_{\mathsf{ex}}$ of our running example:*

$$\sigma_1 : \mathsf{Date}(x_1, y_1) \wedge \mathsf{Date}(x_2, y_2) \wedge <(y_2, y_1) \rightarrow \mathsf{pref}(x_1, x_2)$$
$$\sigma_2 : x_1 = \mathbf{id}(\mathsf{FPr}(y)) \wedge x_2 = \mathbf{id}(\mathsf{APr}(y)) \rightarrow \mathsf{pref}(x_1, x_2)$$
$$\sigma_3 : Y \sqsubseteq \mathsf{Adm} \wedge Z \sqsubseteq \mathsf{Fac} \wedge \neg(\exists z \mathsf{Teach}(y, z)) \wedge \\ x_1 = \mathbf{id}(Y(y)) \wedge x_2 = \mathbf{id}(Z(y)) \rightarrow \mathsf{pref}(x_1, x_2)$$

Rule σ_1 states a general preference for keeping more recently added facts. Rule σ_2 states if we have both $\mathsf{FPr}(p)$ and $\mathsf{APr}(p)$, we prefer to keep $\mathsf{FPr}(p)$, capturing the domain knowledge that associate professors are promoted into full professors. Rule σ_3 states that if a person is declared to

belong both to a subclass of Adm *and a subclass of* Fac*, but there is no* Teach*-fact for the person in the dataset, then the* Adm*-related facts are deemed more reliable. Observe that σ_3 uses ontology axioms with variables in order to simplify rule formulation (avoiding the need to write separate rules for every pair of subclasses of* Adm *and* Fac*).*

Definition 4. *The preference rule language $\mathcal{PL}_{\mathsf{DL}}$ contains rules whose bodies can be obtained from atomic statements of the following forms using $\wedge, \neg$, and $\exists$:*

- *relational atoms $P(t_1, \ldots, t_n)$, with $P \in \mathbf{P}_n$, $t_i \in \mathbf{V} \cup \mathbf{C}$*
- *$x = \mathbf{id}(P(t_1, \ldots, t_n))$, where $P \in \mathbf{P}_n \setminus \mathbf{P}_{\mathbf{M}}$, $t_i \in \mathbf{V} \cup \mathbf{C}$*
- *inequality atoms $x \neq t$, where $x \in \mathbf{V}$, $t \in \mathbf{V} \cup \mathbf{C}$*
- *ontology atoms $X \sqsubseteq P$, where $P \in \mathbf{P} \setminus \mathbf{P}_{\mathbf{M}}$, $X \in \mathbf{V}$*

The evaluation function for $\mathcal{PL}_{\mathsf{DL}}$ is defined as follows: $eval(\mathcal{K}, \mathcal{M}, \mathsf{Cond}(x_1, x_2), id_1, id_2) = \mathsf{true}$ iff there exists a function ν that maps each free variable in $\mathsf{Cond}(x_1, x_2)$ to an element of $\mathbf{C} \cup \mathbf{P}$ and is such that $\nu(x_1) = id_1$, $\nu(x_2) = id_2$ and $f(\mathcal{K}, \mathcal{M}, \nu(\mathsf{Cond}(x_1, x_2))) = \mathsf{true}$, where $\nu(\mathsf{Cond}(x_1, x_2))$ denotes the expression obtained by replacing each free variable y by $\nu(y)$ in $\mathsf{Cond}(x_1, x_2)$ and f is defined recursively as follows:

- *$f(\mathcal{K}, \mathcal{M}, \phi_1 \wedge \phi_2) = f(\mathcal{K}, \mathcal{M}, \phi_1) \wedge f(\mathcal{K}, \mathcal{M}, \phi_2)$,*
- *$f(\mathcal{K}, \mathcal{M}, \neg\phi) = \neg f(\mathcal{K}, \mathcal{M}, \phi)$,*
- *$f(\mathcal{K}, \mathcal{M}, \exists z \phi) = \mathsf{true}$ iff there exists $\nu_z : \{z\} \mapsto \mathbf{C} \cup \mathbf{P}$ such that $f(\mathcal{K}, \mathcal{M}, \nu_z(\phi)) = \mathsf{true}$,*
- *for every $P \in \mathsf{sig}(\mathcal{F})$ and tuple $\vec{c}$ of constants, $f(\mathcal{K}, \mathcal{M}, P(\vec{c})) = \mathsf{true}$ iff $\mathcal{F} \models P(\vec{c})$,*
- *for every $P \in \mathsf{sig}(\mathcal{K})$ and tuple $\vec{c}$ of constants, $f(\mathcal{K}, \mathcal{M}, P(\vec{c})) = \mathsf{true}$ iff $\mathcal{D} \models P(\vec{c})$,*
- *$f(\mathcal{K}, \mathcal{M}, id_i = \mathbf{id}(P(\vec{c}))) = \mathsf{true}$ iff $id_i = \mathbf{id}(P(\vec{c}))$,*
- *for $c, d \in \mathbf{C} \setminus \mathbf{C}_{\mathbf{ID}}$, $f(\mathcal{K}, \mathcal{M}, c \neq d) = \mathsf{true}$ iff $c \neq d$,*
- *$f(\mathcal{K}, \mathcal{M}, A \sqsubseteq B) = \mathsf{true}$ iff $\mathcal{T} \models A \sqsubseteq B$,*
- *for any atom α of another form, $f(\mathcal{K}, \mathcal{M}, \alpha) = \mathsf{false}$.*

Example 4. *By Definitions 3 and 4, $\Sigma_{\mathsf{ex}}(\mathcal{K}_{\mathsf{ex}}, \mathcal{M}_{\mathsf{ex}})$ is:*

$$\{\mathsf{pref}(2,1), \mathsf{pref}(2,3), \mathsf{pref}(1,3), \mathsf{pref}(6,7)\}$$

with $\mathsf{pref}(2,1)$ induced by both the first and second rules.

While preference rules allow users to describe in which cases one fact should be preferred to another, we cannot immediately obtain a priority relation from $\Sigma(\mathcal{K}, \mathcal{M})$. This is firstly because priority relations must satisfy the property that $\alpha \succ \beta$ implies that α and β appear together in a conflict. While one could modify the definition of preference rules to enforce this property, it would lead to much more complicated rules, as users would need to include extra conditions in rule bodies to ensure only pairs of ids of conflicting facts occur in the head. We choose not to impose such a requirement, as it is more natural, we believe, to simply interpret a preference rule $\mathsf{Cond}(x_1, x_2) \rightarrow \mathsf{pref}(x_1, x_2)$ as meaning "*if the facts with ids x_1 and x_2 are in conflict*, and $\mathsf{Cond}(x_1, x_2)$ is satisfied, then prefer fact x_1 to fact x_2". Formally, this means that instead of working with all pairs mentioned in $\Sigma(\mathcal{K}, \mathcal{M})$, we consider the binary relation $\succ_{\Sigma,\mathcal{K},\mathcal{M}}$, defined by setting $\alpha \succ_{\Sigma,\mathcal{K},\mathcal{M}} \beta$ iff $\mathsf{pref}(\mathbf{id}(\alpha), \mathbf{id}(\beta)) \in \Sigma(\mathcal{K}, \mathcal{M})$ *and* there exists $\mathcal{C} \in \mathit{Conf}(\mathcal{K})$ such that $\{\alpha, \beta\} \subseteq \mathcal{C}$.

The relation $\succ_{\Sigma,\mathcal{K},\mathcal{M}}$ may still fail to be a priority relation if it contains a cycle, as priority relations are required to be acyclic. In what follows, we explore two complementary approaches to tackling this issue: identifying preference rules which are guaranteed to yield an acyclic relation, and employing different methods to extract an acyclic sub-relation.

Finally let us note that while the definition of priority relation does not require transitivity, this is often considered a natural property for preferences. However, we argue that even in cases where transitivity is desired, one should first resolve any cycles in the 'direct' preferences given in $\succ_{\Sigma,\mathcal{K},\mathcal{M}}$, then only afterwards close under transitivity.

3.2 Checking Acyclicity of Preference Rules

It would be useful to be able to determine in advance, without knowing the dataset and meta-database, whether a given set of preference rules is guaranteed to produce an acyclic relation (for example, to alert users and allow them the option of modifying the rules if this is not the case). Let us first formalize precisely which property we aim to test:

Definition 5. *Given a logical theory $\mathcal{T}$, we say that a set Σ of preference rules is* $\mathcal{T}$-acyclic *if for every dataset $\mathcal{D}$ and every meta-database $\mathcal{M}$ for the KB $\mathcal{K} = (\mathcal{D}, \mathcal{T})$, the induced binary relation $\succ_{\Sigma,\mathcal{K},\mathcal{M}}$ is acyclic.*

The decidability and complexity of verifying $\mathcal{T}$-acyclicity naturally depends on the expressivity of the logical theory and rule bodies. For our proposed language $\mathcal{PL}_{\mathsf{DL}}$, the problem is typically undecidable, since finite satisfiability of FO-sentences can be reduced to $\mathcal{T}$-acyclicity:

Theorem 3. *Let $\mathcal{T}$ be any non-trivial theory (i.e. which can generate some conflict). Then it is undecidable to test whether a set $\Sigma \subseteq \mathcal{PL}_{\mathsf{DL}}$ is $\mathcal{T}$-acyclic.*

We now present a positive result that covers some prominent ontology and constraint languages and supports reasonably expressive rule bodies. Specifically, we consider the language $\mathcal{PL}_{\mathsf{pos}}$ obtained from $\mathcal{PL}_{\mathsf{DL}}$ by disallowing ontology atoms and negation (retaining inequality atoms $x \neq t$).

Theorem 4. *Given a theory $\mathcal{T}$ consisting of binary denial constraints and a set Σ of preference rules from $\mathcal{PL}_{\mathsf{pos}}$, it is decidable whether Σ is $\mathcal{T}$-acyclic. Moreover, the problem can be decided in co*NP *if the predicate arity is bounded.*

Corollary 1. *$\mathcal{T}$-acyclicity testing is in co*NP *if $\mathcal{T}$ is a DL-Lite ontology and the preference ruleset is in $\mathcal{PL}_{\mathsf{pos}}$.*

We expect that the preceding result can be extended to arbitrary denial constraints (and ontology languages with bounded-size non-binary conflicts), but the argument will become considerably more involved as one needs to ensure that the shortened cycle constructed in the proof only involves pairs of facts that co-occur in a conflict. We observe however that the proof of Theorem 4 already provides us with a procedure for checking acyclicity of $\Sigma(\mathcal{K}, \mathcal{M})$, which provides a sufficient condition for $\mathcal{T}$-acyclicity:

Definition 6. *We say that a set Σ of preference rules is* strongly acyclic *if for every KB $\mathcal{K} = (\mathcal{D}, \mathcal{T})$ and every meta-database $\mathcal{M}$ for $\mathcal{K}$, the binary relation $\{(\alpha, \beta) \mid \mathsf{pref}(\mathbf{id}(\alpha), \mathbf{id}(\beta)) \in \Sigma(\mathcal{K}, \mathcal{M})\}$ is acyclic.*

Theorem 5. *If Σ is strongly acyclic, then it is $\mathcal{T}$-acyclic.*

Example 5. *The ruleset $\Sigma = \{\sigma_2\}$ is strongly acyclic, as σ_2 can only induce* $\mathsf{pref}(\mathbf{id}(\alpha), \mathbf{id}(\beta))$ *if α is an* FPr*-fact and β a* APr*-fact, so no cycle can be constructed.*

It is also interesting to observe that if some metadata predicates enjoy special properties, this information could be exploited to identify additional acyclic rulesets.

Example 6. *Suppose now that $\Sigma = \{\sigma_1\}$. Naturally we expect that the meta-database contains a unique fact* $\mathsf{Date}(\mathbf{id}(\alpha), d)$ *for each fact α and that $<$ provides a total order over the values in the second argument of* Date*. If we were to adapt our acyclicity notions to only quantify over meta-databases satisfying these constraints, then we could conclude that Σ is strongly acyclic.*

We leave it as future work to develop more sophisticated ($\mathcal{T}$- or strong) acyclicity checking procedures that can take into account such additional information.

3.3 Resolving Cycles to Get a Priority Relation

Ideally the preference ruleset would satisfy the introduced acyclicity conditions, but this cannot be assumed in general. Indeed, we have seen that it may be undecidable to determine whether a given ruleset satisfies the conditions. Furthermore, cycles can naturally arise when users create rules that capture different criteria, e.g. prefer more recent facts and prefer facts from more trusted sources. To ensure acyclicity in such cases, one would need to create more complex rules whose bodies consider different combinations of the criteria, making rules much harder for users to specify and understand. We thus advocate a pragmatic approach: give users free rein to specify preferences as they see fit, then apply cycle resolution techniques to extract a suitable acyclic sub-relation should any cycles arise.

To enable a more fine-grained specification of the preferences, we allow users to partition the set Σ of preference rules into priority levels $\Sigma_1, \ldots, \Sigma_n$, so that a preference induced by a preference rule from Σ_i is considered more important than one induced by a preference rule from Σ_j with $j > i$, and will thus be preferably kept in the cycle elimination process. If no such partition is specified, then all rules are assigned to Σ_1. For every $\mathsf{pref}(\mathbf{id}(\alpha), \mathbf{id}(\beta)) \in \Sigma(\mathcal{K}, \mathcal{M})$, we denote by $\mathit{level}(\alpha, \beta)$ the *minimal* index i such that $\mathsf{pref}(\mathbf{id}(\alpha), \mathbf{id}(\beta)) \in \Sigma_i(\mathcal{K}, \mathcal{M})$. We consider several ways of removing cycles to obtain a priority relation $\succ$ from $\succ_{\Sigma,\mathcal{K},\mathcal{M}}$ (abbreviated to $\succ_\Sigma$ in what follows):

- Going up ($\succ^u$): Let $\succ^u := \emptyset$ and $i := 1$. Then while $\succ^u \cup \succ_{\Sigma_i}$ is acyclic, let $\succ^u := \succ^u \cup \succ_{\Sigma_i}$ and increment i.
- Going down ($\succ^d$): Let $\succ^d := \succ_\Sigma$ and $i := n$. Then while $\succ^d$ is cyclic, let $\succ^d := \succ^d \setminus \{(\alpha, \beta) \mid \mathit{level}(\alpha, \beta) = i, (\alpha, \beta)$ is in a cycle w.r.t. $\succ^d\}$ and decrement i.
- Refined going up ($\succ^{ru}$): Let $\succ^{ru} := \succ_{\Sigma_1}$, then remove every (α, β) that occurs in a cycle w.r.t. $\succ_{\Sigma_1}$. Then for $i = 2$ to n, add to $\succ^{ru}$ all pairs (α, β) such that $\mathit{level}(\alpha, \beta) = i$ and (α, β) does not belong to any cycle w.r.t. $\succ^{ru} \cup \succ_{\Sigma_i}$.
- Grounded ($\succ^g$): Let $\succ^g := \emptyset$. Then until a fixpoint is reached, add to $\succ^g$ all pairs (α, β) such that $\alpha \succ_\Sigma \beta$ and for every cycle c of $\succ_\Sigma$ containing (α, β), either there is $(\gamma, \delta) \in c$ such that $\mathit{level}(\alpha, \beta) < \mathit{level}(\gamma, \delta)$, or there is $(\gamma, \delta) \in c$ such that $\succ^g \cup \{(\gamma, \delta)\}$ is cyclic.

We next relate the preceding strategies to notions that have been proposed in the literature to select a single consistent set of facts from a KB whose dataset is partitioned into priority levels. Indeed, one can define the KB $\mathcal{K}^{cy} = (\mathcal{D}^{cy}, \mathcal{T}^{cy})$ with $\mathcal{D}^{cy} = \{R(\mathbf{id}(\alpha), \mathbf{id}(\beta)) \mid \alpha \succ_\Sigma \beta\}$ and $\mathcal{T}^{cy} = \{R(x, y) \wedge R(y, z) \rightarrow R(x, z),\ R(x, x) \rightarrow \bot\}$, whose conflicts correspond exactly to the minimal cycles of $\succ_\Sigma$, and further partition $\mathcal{D}^{cy}$ into $\mathcal{D}^{cy}_1, \ldots, \mathcal{D}^{cy}_n$ as follows: $R(\mathbf{id}(\alpha), \mathbf{id}(\beta)) \in \mathcal{D}^{cy}_i$ iff $\mathit{level}(\alpha, \beta) = i$. For such a KB $\mathcal{K}$ whose dataset is partitioned into priority levels $\mathcal{D}_1, \ldots, \mathcal{D}_n$, Benferhat, Bouraoui, and Tabia (2015) defined the *possibilistic repair* $\mathsf{Poss}(\mathcal{K}) = \mathcal{D}_1 \cup \cdots \cup \mathcal{D}_{inc(\mathcal{K})-1}$ where $inc(\mathcal{K})$ is the inconsistency degree of $\mathcal{K}$, i.e., the minimal i such that $\mathcal{D}_1 \cup \cdots \cup \mathcal{D}_i$ is inconsistent; the *non-defeated repair* $\mathsf{NonDef}(\mathcal{K})$, defined as the union of the intersections of the (subset) repairs of $\mathcal{D}_1$, $\mathcal{D}_1 \cup \mathcal{D}_2$, …, $\mathcal{D}_1 \cup \cdots \cup \mathcal{D}_n$; and the *prioritized inclusion-based non-defeated repair* $\mathsf{Prio}(\mathcal{K})$, defined similarly to $\mathsf{NonDef}(\mathcal{K})$ but considering optimal repairs instead of subset repairs. Indeed, when a priority relation is induced from priority levels (called score-structured in the literature), the three notions of optimal repairs coincide, and can be defined directly from the priority levels (Bourgaux 2016; Livshits and Kimelfeld 2017). Finally, Bienvenu and Bourgaux (2020) defined the preference-based set-based argumentation framework associated with a prioritized KB $\mathcal{K}$, whose arguments are the KB facts and attacks are defined from the KB conflicts, and considered its *grounded extension* $\mathsf{Grd}(\mathcal{K})$.

Theorem 6. *It holds that:*

- $\alpha \succ^u \beta$ *iff* $R(\mathbf{id}(\alpha), \mathbf{id}(\beta)) \in \mathsf{Poss}(\mathcal{K}^{cy})$,
- $\alpha \succ^d \beta$ *iff* $R(\mathbf{id}(\alpha), \mathbf{id}(\beta)) \in \mathsf{NonDef}(\mathcal{K}^{cy})$,
- $\alpha \succ^g \beta$ *iff* $R(\mathbf{id}(\alpha), \mathbf{id}(\beta)) \in \mathsf{Grd}(\mathcal{K}^{cy})$.

It has been shown that $\mathsf{Poss}(\mathcal{K}) \subseteq \mathsf{NonDef}(\mathcal{K}) \subseteq \mathsf{Grd}(\mathcal{K}) \subseteq \mathsf{Prio}(\mathcal{K})$ and that all these sets of facts can be computed in polynomial time except for $\mathsf{Prio}(\mathcal{K})$ (Benferhat, Bouraoui, and Tabia 2015; Bienvenu and Bourgaux 2020). Combined with Theorem 6, these results can help us show the following theorems:

Theorem 7. $\succ^u \subseteq \succ^d \subseteq \succ^g$ *and* $\succ^u \subseteq \succ^d \subseteq \succ^{ru}$.

Theorem 8. *Each of the relations $\succ^u, \succ^d, \succ^g, \succ^{ru}$ can be computed in polynomial time from the relations $\succ_{\Sigma_i}$.*

Examples 7 and 8 show that $\succ^g$ and $\succ^{ru}$ are incomparable and that it may be the case that $\alpha \succ^{ru} \beta$ while $R(\mathbf{id}(\alpha), \mathbf{id}(\beta)) \notin \mathsf{Prio}(\mathcal{K}^{cy})$.

Example 7. *Assume that* $\succ_{\Sigma_1} = \{(\alpha, \beta), (\beta, \gamma)\}$*, and that* $\succ_{\Sigma_2} = \{(\alpha, \gamma), (\gamma, \alpha))\}$*. Then* $\succ^{ru} = \{(\alpha, \beta), (\beta, \gamma)\}$ *while* $\succ^g = \{(\alpha, \beta), (\beta, \gamma), (\alpha, \gamma)\}$*, so* $\succ^{ru} \subsetneq \succ^g$.

Example 8. *Assume that* $\succ_{\Sigma_1} = \{(\alpha, \beta), (\gamma, \delta)\}$, $\succ_{\Sigma_2} = \{(\beta, \gamma), (\delta, \alpha))\}$, *and* $\succ_{\Sigma_3} = \{(\gamma, \beta)\}$. *Then* $\succ^{ru} = \{(\alpha, \beta), (\gamma, \delta), (\gamma, \beta)\}$ *while* $\succ^g = \{(\alpha, \beta), (\gamma, \delta)\}$, *so* $\succ^g \subsetneq \succ^{ru}$. *Note that* $R(\mathbf{id}(\gamma), \mathbf{id}(\beta)) \notin \mathsf{Prio}(\mathcal{K}^{cy})$ *since* $\{R(\mathbf{id}(\alpha), \mathbf{id}(\beta)), R(\mathbf{id}(\gamma), \mathbf{id}(\delta)), R(\mathbf{id}(\beta), \mathbf{id}(\gamma))\}$ *is an optimal repair of* $\mathcal{K}^{cy}$.

	program facts and rules	input encoded
$\Pi_{\mathcal{D}}$	$\texttt{data}(i).$ $P(i, c_1, \ldots, c_n).$	$P(c_1, \ldots, c_n) \in \mathcal{D},$ $\mathbf{id}(P(c_1, \ldots, c_n)) = i$
$\Pi_{\mathcal{F}}$	$Q(c_1, \ldots, c_n).$	$Q(c_1, \ldots, c_n) \in \mathcal{F}$
Π_C	$\texttt{conf\_init}((\texttt{Id0}, \ldots, \texttt{Idk})) \text{ :- } P_0(\texttt{Id0}, t_1^0, \ldots, t_{n_0}^0), \ldots, P_k(\texttt{Idk}, t_1^k, \ldots, t_{n_k}^k).$ $\texttt{inConf\_init}((\texttt{Id0}, \ldots, \texttt{Idk}), \texttt{Idj}) \text{ :- } P_0(\texttt{Id0}, t_1^0, \ldots, t_{n_0}^0), \ldots, P_k(\texttt{Idk}, t_1^k, \ldots, t_{n_k}^k).$	$\bigwedge_{i=0}^{k} P_i(t_1^i, \ldots, t_{n_i}^i) \rightarrow \bot \in Inc(\mathcal{T})$
Π_Q	$\texttt{cause}((x_0, \ldots, x_m), (\texttt{Id0}, \ldots, \texttt{Idk})) \text{ :- } P_0(\texttt{Id0}, t_1^0, \ldots, t_{n_0}^0), \ldots, P_k(\texttt{Idk}, t_1^k, \ldots, t_{n_k}^k).$ $\texttt{inCause}((\texttt{Id0}, \ldots, \texttt{Idk}), \texttt{Idj}) \text{ :- } P_0(\texttt{Id0}, t_1^0, \ldots, t_{n_0}^0), \ldots, P_k(\texttt{Idk}, t_1^k, \ldots, t_{n_k}^k).$	$\exists \vec{y} \bigwedge_{i=0}^{k} P_i(t_1^i, \ldots, t_{n_i}^i)$ $\rightarrow q(x_0, \ldots, x_m) \in Rew(q, \mathcal{T})$
Π_P	$\texttt{pref\_init}(x_1, x_2, i) \text{ :- } \texttt{inConf}(\texttt{C}, x_1), \texttt{inConf}(\texttt{C}, x_2),$ $P_0(\texttt{X0}, t_1^0, \ldots, t_{n_0}^0), \ldots, P_k(\texttt{Xk}, t_1^k, \ldots, t_{n_k}^k),$ $\texttt{not } P_0'(\texttt{Y0}, t_1'^0, \ldots, t_{n_0'}'^0), \ldots, \texttt{not } P_{k'}'(\texttt{Yk}', t_1'^{k'}, \ldots, t_{n_{k'}'}'^{k'}),$ $Q_0(l_1^0, \ldots, l_{p_0}^0), \ldots, Q_m(l_1^m, \ldots, l_{p_m}^m),$ $\texttt{not } Q_0'(l_1'^0, \ldots, l_{p_0'}'^0), \ldots, \texttt{not } Q_{m'}'(l_1'^{m'}, \ldots, l_{p_{m'}'}'^{m'}),$ $f_1^0 \bowtie f_2^0, \ldots, f_1^r \bowtie f_2^r,$ $P_1''(t_1, t_1''^1, \ldots, t_{n_1''}''^1), \ldots, P_q''(t_q, t_1''^q, \ldots, t_{n_q''}''^q).$ $\texttt{level}(i).$	$\texttt{Cond}(x_1, x_2) \rightarrow \texttt{pref}(x_1, x_2) \in \Sigma_i$ $\texttt{Cond}(x_1, x_2) = \exists \vec{y} \bigwedge_{i=0}^{k} P_i(t_1^i, \ldots, t_{n_i}^i) \wedge$ $\bigwedge_{i=0}^{k'} \neg P_i'(t_1'^i, \ldots, t_{n_i'}'^i) \wedge$ $\bigwedge_{i=0}^{m} Q_i(l_1^i, \ldots, l_{p_i}^i) \wedge$ $\bigwedge_{i=0}^{m'} \neg Q_i'(l_1'^i, \ldots, l_{p_i'}'^i) \wedge$ $\bigwedge_{\ell=0}^{r} f_1^\ell \bowtie f_2^\ell \wedge$ $\bigwedge_{i=1}^{q} t_i = \mathbf{id}(P_i''(t_1''^i, \ldots, t_{n_i''}''^i))$

Table 1: Logic programs encoding the input. $P, P_i, P_i', P_i'' \in \mathsf{sig}(\mathcal{D})$, $Q, Q_j \in \mathsf{sig}(\mathcal{F})$, terms are in $\mathbf{C} \cup \mathbf{V}$ and $\bowtie \in \{=, \neq, >, <, \geq, \leq\}$.

4 ASP Implementation

We implement our approach using *answer set programming* (ASP) (Lifschitz 2019; Gebser et al. 2012). We consider ASP *programs* consisting of *rules* of the form

$$\gamma \text{ :- } \alpha_1, \ldots, \alpha_n, \texttt{not}\beta_1, \ldots, \texttt{not}\beta_m.$$

where $\gamma, \alpha_i, \beta_j$ are atoms built from predicates, variables, constants and comparison operators. Every variable occurring in the head γ of a rule must also occur in some positive literal of its body $\alpha_1, \ldots, \alpha_n, \texttt{not}\beta_1, \ldots, \texttt{not}\beta_m$. A rule with an empty body is a *fact*, and a rule with an empty head a *constraint*. We also use *choice rules* to select exactly or at least one atom from a set. Importantly, it is possible to use a tuple of terms as a predicate argument. We use this to define, e.g., conflict identifiers as the tuple of the identifiers of their facts. ASP is based on the *stable model* semantics.

We implement several building blocks, which provide an almost end-to-end approach to querying inconsistent KBs. Our system takes as input logic programs representing the input, and computes the query answers under the chosen semantics among X-brave, X-AR or X-IAR with $X \in \{S, P, C\}$ w.r.t. $\succ^x$ for the chosen $x \in \{u, d, ru, g\}$. All building blocks can be encoded into ASP programs that a Python program combines and passes to the ASP solver clingo[3] (Gebser et al. 2011) to check whether the resulting program has a stable model. However, we found more efficient in practice to split the computation into several steps and implement some of them in Python (see Section 4.1).

4.1 Input, Conflicts, Causes and Preferences

Our approach applies to any logical theory $\mathcal{T}$ such that:

1. there exists a set $Inc(\mathcal{T})$ of rules of the form $q \rightarrow \bot$ with q a Boolean CQ, such that for every dataset $\mathcal{D}$, $(\mathcal{D}, \mathcal{T}) \models \bot$ iff there exists $q \rightarrow \bot \in Inc(\mathcal{T})$ such that $\mathcal{D} \models q$; and
2. for every CQ $q(\vec{x})$ there exists a set $Rew(q, \mathcal{T})$ of rules of the form $q'(\vec{x}) \rightarrow q(\vec{x})$ with q' a CQ such that for every $\mathcal{D}$ s.t. $(\mathcal{D}, \mathcal{T}) \not\models \bot$ and tuple $\vec{a}$, $(\mathcal{D}, \mathcal{T}) \models q(\vec{a})$ iff there exists $q'(\vec{x}) \rightarrow q(\vec{x}) \in Rew(q, \mathcal{T})$ s.t. $(\mathcal{D}, \mathcal{T}) \models q'(\vec{a})$.

These conditions are fulfilled, e.g., when $\mathcal{T}$ is a set of denial constraints (then, $Inc(\mathcal{T}) = \mathcal{T}$ and $Rew(q, \mathcal{T}) = \{q \rightarrow q\}$), or when $\mathcal{T}$ is a DL-Lite ontology. Regarding preference rules, we handle rules whose bodies are CQs with negation and comparison operators (see Table 1 for the syntax).

We expect that the KB $\mathcal{K} = (\mathcal{D}, \mathcal{T})$, meta-database $\mathcal{M} = (\mathbf{id}, \mathcal{F})$, preference rules $\Sigma = \Sigma_1 \cup \cdots \cup \Sigma_n$, and query q have been transformed into the five ASP programs given in Table 1. Programs $\Pi_{\mathcal{D}}$ and $\Pi_{\mathcal{F}}$ represent the dataset $\mathcal{D}$ and the identifier function $\mathbf{id}$, and the meta-database respectively, and can be obtained quite straightforwardly from various data formats. Constructing Π_C and Π_Q, which encode the constraints and queries, is more demanding since it requires to compute the sets $Inc(\mathcal{T})$ and $Rew(q, \mathcal{T})$.

Proposition 1. *The program $\Pi_{\mathcal{D}} \cup \Pi_Q$ has a single stable model $\mathcal{S}$, and for every $\{\alpha_0, \ldots, \alpha_k\} \in Causes(q(\vec{a}), \mathcal{K})$, $\mathcal{S}$ contains the facts $\texttt{cause}((\vec{a}), (\mathbf{id}(\alpha_0), \ldots, \mathbf{id}(\alpha_k)))$ and $\texttt{inCause}((\mathbf{id}(\alpha_0), \ldots, \mathbf{id}(\alpha_k)), \mathbf{id}(\alpha_j))$, $0 \leq j \leq k$. Moreover, if $\texttt{cause}((\vec{a}), (\mathbf{id}(\alpha_0), \ldots, \mathbf{id}(\alpha_k))) \in \mathcal{S}$, then $(\{\alpha_0, \ldots, \alpha_k\}, \mathcal{T}) \models q(\vec{a})$.*

Essentially, $\Pi_{\mathcal{D}} \cup \Pi_Q$ computes a *superset* of $Causes(q(\vec{a}), \mathcal{K})$, such that each superfluous $\mathcal{B}$ either includes a real cause of $q(\vec{a})$ or contains a conflict. Similarly, $\Pi_{\mathcal{D}} \cup \Pi_C$ computes a *superset* of $Conf(\mathcal{K})$, such that each superfluous $\mathcal{B}$ contains an actual conflict. To obtain $Conf(\mathcal{K})$, we filter out these non-minimal $\mathcal{T}$-inconsistent subsets either via an ASP program Π_{minC} or by a Python program, which we found faster in practice. In the case where conflicts are of size at most two, we further optimize the program by relying on the fact that non-minimal $\mathcal{T}$-inconsistent subsets we compute are not conflicts only if they contain some self-inconsistent fact. We do not need to filter out the superfluous sets from the superset of

[3] https://github.com/potassco/clingo

$\Pi_{\succ^u}$	`trans_cl(X,Y,I) :- pref_init(X,Y,I), not blocked(I).`
	`trans_cl(X,Y,I) :- level(I), trans_cl(X,Y,J), J < I, not blocked(I).`
	`trans_cl(X,Y,I) :- pref_init(X,Z,J), trans_cl(Z,Y,I), J <= I, not blocked(I).`
	`cycle(I) :- trans_cl(X,X,I).`
	`blocked(I) :- level(I), cycle(J), J < I.`
		`pref(X,Y) :- pref_init(X,Y,I), not cycle(I), not blocked(I).`

$\Pi_{\succ^d}$	`trans_cl(X,Y,I) :- pref_init(X,Y,I).`
	`trans_cl(X,Y,I) :- pref_init(X,Z,I), trans_cl(Z,Y,J), J <= I.`
	`trans_cl(X,Y,I) :- pref_init(X,Z,J), trans_cl(Z,Y,I), J <= I.`
	`cycle(X,Y,I) :- pref_init(X,Y,I), trans_cl(Y,X,I).`
		`pref(X,Y) :- pref_init(X,Y,I), not cycle(X,Y,I).`

$\Pi_{\succ^{ru}}$	`trans_cl(X,Y,I) :- pref_init(X,Y,I).`
	`trans_cl(X,Y,I) :- level(I), rel(X,Y,J), J < I.`
	`trans_cl(X,Y,I) :- pref_init(X,Z,I), trans_cl(Z,Y,I).`
	`trans_cl(X,Y,I) :- trans_cl(X,Z,I), trans_cl(Z,Y,I).`
	`cycle(X,Y,I) :- pref_init(X,Y,I), trans_cl(Y,X,I).`
	`rel(X,Y,I) :- pref_init(X,Y,I), not cycle(X,Y,I).`
	`rel(X,Y,I) :- rel(X,Z,J), rel(Z,Y,I), J <= I.`
		`pref(X,Y) :- pref_init(X,Y,I), rel(X,Y,I).`

Table 2: Logic programs to compute $\succ^x$ from facts on predicates `conf`, `inConf`, `pref_init` and `level`.

$Causes(q(\vec{a}), \mathcal{K})$, and only need to ensure that they do not contain some self-inconsistent fact (cf. (Bienvenu and Bourgaux 2022, Section 4)), which we do using Python.

Proposition 2. *The program $\Pi_{\mathcal{D}} \cup \Pi_C \cup \Pi_{minC}$ has a single stable model $\mathcal{S}$, which is such that $\{\alpha_0, \ldots, \alpha_k\} \in Conf(\mathcal{K})$ iff $\mathcal{S}$ contains $\texttt{conf}((\mathbf{id}(\alpha_0), \ldots, \mathbf{id}(\alpha_k)))$ and $\texttt{inConf}((\mathbf{id}(\alpha_0), \ldots, \mathbf{id}(\alpha_k)), \mathbf{id}(\alpha_j))$, $0 \leq j \leq k$.*

Finally, Π_P encodes the preference rules with their priority levels. Note that we add in the preference rule body the condition that the two facts compared in the head belong to the same conflict to compute directly the $\succ_{\Sigma_i}$'s.

Proposition 3. *The program $\Pi_{\mathcal{D}} \cup \Pi_{\mathcal{F}} \cup \Pi_C \cup \Pi_{minC} \cup \Pi_P$ has a single stable model $\mathcal{S}$, which is such that for every $\alpha, \beta \in \mathcal{D}$, $\alpha \succ_{\Sigma_i} \beta$ iff $\texttt{pref\_init}(\mathbf{id}(\alpha), \mathbf{id}(\beta), i) \in \mathcal{S}$.*

4.2 Computing the Priority Relation

We compute $\succ^x$ for the chosen $x \in \{u, d, ru, g\}$ from the conflicts given by facts on predicates `conf`, `inConf`, and the $\succ_{\Sigma_i}$'s given by `pref_init` with $\Pi_{\succ^x}$. For $x \in \{u, d, ru\}$, $\Pi_{\succ^x}$ is given in Table 2 (for space reasons, we omit $\Pi_{\succ^g}$, which draws inspiration from the ASP encoding of the grounded extension from (Egly, Gaggl, and Woltran 2008)).

Proposition 4. *The program $\Pi_{\mathcal{D}} \cup \Pi_{\mathcal{F}} \cup \Pi_C \cup \Pi_{minC} \cup \Pi_P \cup \Pi_{\succ^x}$ has a single stable model $\mathcal{S}$ which is such that for all $\alpha, \beta \in \mathcal{D}$, $\alpha \succ^x \beta$ iff $\texttt{pref}(\mathbf{id}(\alpha), \mathbf{id}(\beta)) \in \mathcal{S}$.*

4.3 Optimal Repair-Based Semantics

After preliminary experiments, we found it more efficient to treat each potential answer separately, so we transform (using Python) the $\texttt{cause}((\vec{a}), (\mathbf{id}(\alpha_0), \ldots, \mathbf{id}(\alpha_k)))$ facts built by $\Pi_{\mathcal{D}} \cup \Pi_Q$ into a set of programs $\Pi_{\vec{a}}$ representing causes of each $\vec{a}$ with facts $\texttt{cause}((\mathbf{id}(\alpha_0), \ldots, \mathbf{id}(\alpha_k)))$ and $\texttt{inCause}((\mathbf{id}(\alpha_0), \ldots, \mathbf{id}(\alpha_k)), \mathbf{id}(\alpha_j))$. For the ease of presentation, we also denote by $\Pi_{conf_\succ}$ the logic program that contains the conflicts and priority relation. We say that a conflict $\mathcal{C}$ *attacks* a fact α, written $\mathcal{C} \rightsquigarrow \alpha$, if $\alpha \in \mathcal{C}$ and $\alpha \not\succ \beta$ for every $\beta \in \mathcal{C}$. We use a program Π_{att} to precompute the attack relation $\rightsquigarrow$ (`att`) from $\Pi_{conf_\succ}$.

For $X \in \{S, P, C\}$ and $\text{Sem} \in \{\text{brave}, \text{AR}, \text{IAR}\}$, we define $\Pi_{X\text{-Sem}}$ from building blocks inspired by the SAT encodings given by Bienvenu and Bourgaux (2022). Note, however, that the latter are implemented for *binary conflicts*, so our system is the first implementing optimal repair-based inconsistency-tolerant semantics for *conflicts of arbitrary size*. For $\text{Sem} \in \{\text{brave}, \text{AR}\}$, $\Pi_{X\text{-Sem}}$ is the union of:

- Π_{loc}, which localizes the attack relation to relevant facts (those that are reachable from the causes);
- Π_{cons}, which selects (using a choice rule) a consistent set of facts among the relevant facts by enforcing that at least one fact per relevant conflict is removed;
- Π_{brave} if $\text{Sem} = \text{brave}$, which ensures that $\Pi_{X\text{-Sem}}$ is satisfiable only if all facts of some cause are selected;
- Π_{AR} if $\text{Sem} = \text{AR}$, which ensures that $\Pi_{X\text{-Sem}}$ is satisfiable only if every cause is contradicted by the selected facts, meaning that these facts include $\mathcal{C} \setminus \{\alpha\}$ for some $\mathcal{C} \rightsquigarrow \alpha$ with α a fact of the cause;
- Π_{Pareto} if $X = P$ (resp. $\Pi_{Completion}$ if $X = C$), which ensures that $\Pi_{X\text{-Sem}}$ is satisfiable only if the selected facts can be extended into a Pareto- (resp. completion-) repair.

For $\text{Sem} = \text{IAR}$, $\Pi_{X\text{-Sem}}$ intuitively checks whether each cause can be contradicted by a consistent set of facts. It is similar to $\Pi_{X\text{-AR}}$, except that predicates in Π_{loc}, Π_{cons}, Π_{AR} and Π_{Pareto} or $\Pi_{Completion}$ are extended with an extra argument that keeps the identifier of the cause considered.

Proposition 5. *The program $\Pi_{conf_\succ} \cup \Pi_{\vec{a}} \cup \Pi_{att} \cup \Pi_{X\text{-Sem}}$ has a stable model iff*

1. $\mathcal{K}_\succ \models^X_{\text{Sem}} q(\vec{a})$ *if* $\text{Sem} = \text{brave}$*;*
2. $\mathcal{K}_\succ \not\models^X_{\text{Sem}} q(\vec{a})$ *if* $\text{Sem} \in \{\text{AR}, \text{IAR}\}$.

5 Experiments

Our main goal is to compare the different approaches to obtaining a priority relation from preferences rules, in terms of run time and size of the priority relation. We also compare our ASP implementation of the optimal repair-based semantics with ORBITS, the existing SAT-based implementation.

5.1 Experimental Setting

We use the CQAPri benchmark (Bourgaux 2016), a synthetic benchmark adapted from $\text{LUBM}^{\exists}_{20}$ (Lutz et al. 2013)

	#conf.	$\succ_\Sigma$	$\Sigma_1^a \cup \Sigma_2^a \cup \Sigma_3^a$ $\succ^u$	$\succ^d$	$\succ^g$	$\succ_\Sigma$	Σ_1^c $\succ^u$	$\succ^{d,g}$	Σ_1^d $\succ$
u1c1	2,354	7,068	3,041	3,644	3,703	5,633	0	1,510	0
u1c5	8,516	17,804	7,624	8,944	9,324	14,517	0	3,356	1
u1c10	14,301	27,927	2	14,402	14,808	22,634	0	6,082	2
u1c20	28,272	52,361	4	27,185	27,948	42,032	0	12,300	4
u1c30	45,524	82,531	6	41,300	42,601	65,142	_	16,361	6
u1c50	81,344	145,193	_	69,454	_	113,857	_	19,966	8
u5c1	12,024	23,932	10,241	13,275	13,339	18,570	0	7,821	0
u5c5	53,438	96,307	_	52,084	52,820	76,045	0	28,017	1
u5c10	109,493	194,306	6	103,094	_	154,271	_	50,673	6
u5c20	231,811	_	_	_	_	319,549	_	87,006	14
u20c1	73,252	131,103	2	73,157	73,583	103,260	0	74,909	2
u20c50	3,130,417	_	_	_	_	_	_	_	159

Table 3: Number of conflicts, `pref_init` facts ($\succ_\Sigma$), and `pref` facts computed for $\succ^u$, $\succ^d$ and $\succ^g$, for scenarios (a), (c) and (d) (which directly yields an acyclic relation). Empty cells indicate that clingo overflows or reaches a 30 min time-out. We fail to compute priority relations on omitted datasets in all scenarios but (d).

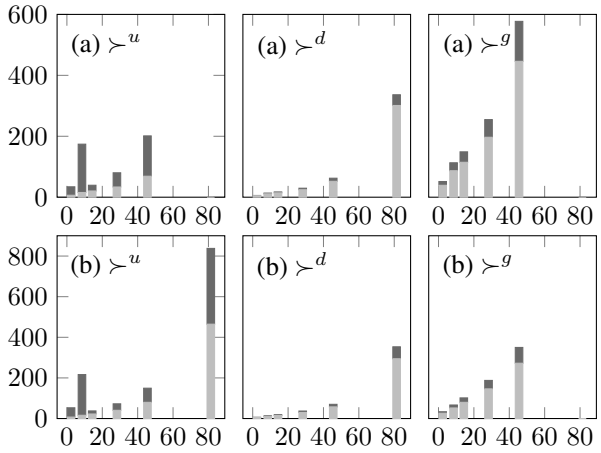


Figure 1: Time (in sec.) to compute $\succ^x$ from the pre-computed conflicts for u1cY given as a program Π_{conf} and $\Pi_\mathcal{D} \cup \Pi_\mathcal{F} \cup \Pi_P \cup \Pi_{\succ^x}$ in scenarios (a) and (b) w.r.t. the number (in thousands) of conflicts. Empty bars for u1c50 (81K conflicts) mean t.o. or oom. The lower part of each bar (light grey) shows the time to ground the ASP program while the upper part is the time to solve it.

to evaluate inconsistency-tolerant query answering over DL-Lite KBs. We also consider its extension with two priority relations given by the ORBITS benchmark (Bienvenu and Bourgaux 2022) for the comparison with ORBITS. In this case, we translate the oriented conflict graph and causes for potential answers provided in the benchmark into $\Pi_{conf_\succ}$ and $\Pi_{\vec{a}}$ (for each potential answer $\vec{a}$). Experiments were run with 16Go of RAM in a cluster node running CentOS Linux 7.6.1810 (Core) with linux kernel 3.10.0, with processor 2x 16-core Skylake Intel Xeon Gold 6142 @ 2.6 GHz. Reported times are averaged over 5 runs.

Datasets and meta-database We build programs $\Pi_\mathcal{D}$ for the uXcY datasets of the CQAPri benchmark with $\mathsf{X} \in \{1, 5, 20\}$ and $\mathsf{Y} \in \{1, 5, 10, 20, 30, 50\}$. These datasets are such that uXcY $\subseteq$ uXcY′ for $\mathsf{Y} \leq \mathsf{Y}'$ and uXcY $\subseteq$ uX′cY for $\mathsf{X} \leq \mathsf{X}'$, with X and Y related to the size and the proportion of facts involved in some conflicts respectively. Their sizes range from 75K to 2M facts and their proportions of facts involved in some (binary) conflict from 3% to 46%. We ensure that for every fact α, $\mathbf{id}(\alpha)$ is the same in all uXcY, and we obtain each program $\Pi_\mathcal{F}$ as a subset of the one generated for the largest dataset u20c50, so that the same meta-data is used across the uXcY. For $\Pi_\mathcal{F}$, we randomly generate two facts per $\alpha \in \mathcal{D}$: $\mathtt{date}(\mathbf{id}(\alpha), n)$ and $\mathtt{source}(\mathbf{id}(\alpha), k)$, where n, k are integers between 0 and 1000. For each k, we also generate a fact $\mathtt{reliability}(k, m)$ with m an integer between 0 and 3.

Ontology and queries We use the DL-Lite ontology and CQs of the CQAPri benchmark to generate Π_C and Π_Q. For Π_C, we first build a denial constraint per concept or role disjointness axiom. To experiment with non-binary conflicts, we also add a denial constraint with 10 relational atoms. We then rewrite all these constraints w.r.t. the ontology using Rapid (Chortaras, Trivela, and Stamou 2011). For the queries, we similarly rewrite each query into a set of CQs.

Preference rules We use the following preferences rules, and test four scenarios: (a) $\Sigma_1^a = \{\rho_3, \rho_4\}$, $\Sigma_2^a = \{\rho_2\}$, $\Sigma_3^a = \{\rho_1\}$; (b) $\Sigma_1^b = \{\rho_1, \rho_3, \rho_4\}$, $\Sigma_2^b = \{\rho_2\}$; (c) $\Sigma_1^c = \{\rho_1, \rho_2, \rho_3, \rho_4\}$; (d) $\Sigma_1^d = \{\rho_3, \rho_4\}$ (dropping ρ_1, ρ_2).

$$\rho_1 : \mathsf{date}(x_1, y_1) \wedge \mathsf{date}(x_2, y_2) \wedge y_1 > y_2 \rightarrow \mathsf{pref}(x_1, x_2)$$
$$\rho_2 : \mathsf{source}(x_1, y_1) \wedge \mathsf{source}(x_2, y_2) \wedge \mathsf{reliability}(y_1, z_1) \wedge \mathsf{reliability}(y_2, z_2) \wedge z_1 > z_2 \rightarrow \mathsf{pref}(x_1, x_2)$$
$$\rho_3 : x_1 = \mathbf{id}(\mathsf{FPr}(y)) \wedge x_2 = \mathbf{id}(\mathsf{APr}(y)) \rightarrow \mathsf{pref}(x_1, x_2)$$
$$\rho_4 : x_1 = \mathbf{id}(\mathsf{APr}(y)) \wedge x_2 = \mathbf{id}(\mathsf{GrSt}(y)) \rightarrow \mathsf{pref}(x_1, x_2)$$

5.2 Experimental Results

Table 3 and Figure 1 present some results of the evaluation of the priority relation computation. We were not able to compute $\succ^{ru}$ even on u1c1 because it overflows the number of atoms clingo can handle. However, we managed to compute the other priority relations for almost all small datasets (>75K), several medium size datasets (>463K), and one large dataset (>1,983K) even in cases with a large proportion of facts in conflicts (44% for u1c50) or high numbers of `pref_init` facts (319K for u5c20 in scenario (c)). All datasets have exactly 40 conflicts of size 10, which yields 1,800 pairs of facts, and other conflicts are binary (so that e.g., u1c1 has 4,114 pairs of conflicting facts). Several preference statements ($\succ_\Sigma$) can be made on each such pair (in both directions and on different priority levels) while the priority relation ($\succ^x$) compares each pair of facts at most once so that, e.g., $\succ^g$ compares 90% of the pairs of conflicting facts of u1c1 in scenario (a). Interestingly, $\succ^d$ and $\succ^g$ often coincide and never differ by more than 5% of `pref` facts on instances for which we computed them, while $\succ^u$ is often reduced to the empty relation. From a computational point of view, $\succ^d$ is significantly faster to compute than $\succ^g$ and $\succ^u$ (except in scenario (d) which yields a very small and acyclic $\succ_\Sigma$). Hence $\succ^d$ may be a good method in practice.

The times given in Figure 1 do not include the time needed to compute the conflicts, which may be far from negligible: while the evaluation of $\Pi_{\mathcal{D}} \cup \Pi_C$ never takes more than about 1min (u20c50), the time needed to minimize the conflicts takes from less than 1sec to 206sec for the u1cY cases, but more than 45 hours for u20c50! In the case where conflicts have size at most two, however, this takes at most 1.5sec for the u1cY cases and no more than 42sec (u20c50).

Regarding the computation of optimal repair-based semantics, we select 8 queries with a reasonable number of potential answers (between 3 and 16,969) because very high numbers of answers lead to time-out (30 minutes per query). Our system is in general by far slower than ORBITS on datasets that are large or with a high percentage of conflicting facts: e.g., on u20c1 and u1c50, our implementation always takes more than 16 times longer and up to more than 1,200 times longer to filter answers under P-AR or P-brave semantics. On the simplest case, u1c1, the difference is less striking (at least if we include the time to load the input in the computation time for ORBITS), but still of orders of magnitude for many queries. However, it is notable that we do manage to answer a few queries under C-AR and C-brave semantics in cases where ORBITS fails.

6 Related Work

We draw inspiration from different preference specification formalisms defined for related settings, such as preference-based query answering over databases (Stefanidis, Koutrika, and Pitoura 2011) or (consistent) KBs (Lukasiewicz, Martinez, and Simari 2013). In the latter work, for example, preference formulas consist of a condition, given by an FO-formula, which induces a preference between two atoms. In the context of controlled query evaluation in DL, Cima et al. (2021) define a preference relation among ontology predicates, which straightforwardly induces one among the facts. Closer to our own work, Calautti et al. (2022) consider preference rules that generate preferences between atoms in order to select preferred repairs of inconsistent KBs. Differently from us, their preference rules are evaluated over the repairs themselves, whereas our rules are evaluated over the dataset (and meta-database) and yield a priority relation between facts, which is then lifted to get optimal repairs.

Our preference rules generalize the preceding preference formalisms by allowing rule bodies that express more complex conditions, e.g., that may refer to meta-data, include negated atoms, or quantify over ontology predicates. In this manner, we obtain an easy and flexible way of defining inconsistency management policies, as considered in (Martinez et al. 2014). Moreover, a distinguishing contribution of our work is that we propose methods for dealing with cycles among the induced preference statements. Our cycle resolution techniques can take into account priorities amongst the preference rules themselves. Rules with priorities are also considered in prioritized logic programming (Sakama and Inoue 2000; Brewka and Eiter 1999), but there serve the purpose of identifying preferred answer sets.

Our work has high-level similarities with (Fagin et al. 2016), which employs optimal repairs from (Staworko, Chomicki, and Marcinkowski 2012) to clean inconsistencies arising amongst facts extracted using document spanners. They introduce priority-generating dependencies to define a priority relation and explore some properties of the induced relations. However, the formalization and techniques differ significantly due to the very different settings.

Another line of related work uses logic programming for consistent query answering over relational databases (Greco, Greco, and Zumpano 2003; Eiter et al. 2008; Manna, Ricca, and Terracina 2013). These works consider different kinds of repair: on the one hand, they allow repairs to restore consistency by adding facts, while we focus on subset repairs, only involving deletions, which are standard for KBs interpreted under the open-world assumption; on the other hand, we consider priority-based optimal repairs. Greco, Greco, and Zumpano (2003), however, define constraints that express conditions on the insertion or deletion of atoms, and rules defining priorities among such updates, sharing the intuition that the user should be able to specify preferences on how to treat inconsistency. On the implementation side, we remark that compared to our experimental setting, the evaluations of previous ASP approaches typically either use databases with very few conflicts (few hundreds), or whose conflicts have a specific structure that ensures that the conflicts form small independent connected components.

7 Conclusion and Future Work

In this paper, we present a rule-based approach to specifying a priority relation between conflicting facts, in order to adopt optimal repair-based inconsistency-tolerant semantics. We investigate the problem of deciding whether the relation induced by a set of preference rules is guaranteed to be acyclic and propose several strategies to remove cycles. We also present an implementation of the approach, including the computation of query answers that hold under a given semantics, which was not yet implemented for the case of non-binary conflicts and optimal repairs. While our comparison show that existing SAT implementation is more efficient for the latter task (though the SAT implementation is optimized for binary conflicts while ours handles conflicts of any size so the comparison is not entirely fair), ASP retains a number of advantages. Besides allowing the user to directly and easily express preference rules, logic programs are easy to modify to treat other problems (such as the computation of repairs, which is not tackled by ORBITS). Moreover, ASP is more expressive than SAT, so that it is theoretically possible to employ ASP to compute answers under globally-optimal-repair-based semantics (which have Σ_2^p / Π_2^p complexity), even if finding an efficient encoding remains a challenge.

There are several directions for future work. First, we could extend the static analysis of Section 3.2, by considering more classes of logical theories and preference rules. Besides the problem of deciding whether a theory and set of preference rules ensure that the induced relation is acyclic, one could wonder whether they guarantee that there exists a unique optimal repair. On the practical side, we want to implement the last missing blocks to have a truly end-to-end system for query answering over inconsistent KBs with preference rules (in particular to generate the input logic programs of Table 1 from data/theory given in various formats).

Acknowledgments

This work was supported by the ANR AI Chair INTENDED (ANR-19-CHIA-0014) and JST CREST Grant Number JPMJCR22D3, Japan.

References

Benferhat, S.; Bouraoui, Z.; and Tabia, K. 2015. How to select one preferred assertional-based repair from inconsistent and prioritized DL-Lite knowledge bases? In *Proceedings of IJCAI*.

Bertossi, L. E. 2019. Database repairs and consistent query answering: Origins and further developments. In *Proceedings of PODS*.

Bienvenu, M., and Bourgaux, C. 2020. Querying and repairing inconsistent prioritized knowledge bases: Complexity analysis and links with abstract argumentation. In *Proceedings of KR*.

Bienvenu, M., and Bourgaux, C. 2022. Querying inconsistent prioritized data with ORBITS: algorithms, implementation, and experiments. In *Proceedings of KR*.

Bienvenu, M., and Bourgaux, C. 2023. Inconsistency handling in prioritized databases with universal constraints: Complexity analysis and links with active integrity constraints. In *Proceedings of KR*.

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

Bienvenu, M.; Bourgaux, C.; Inoue, K.; and Jean, R. 2025. A rule-based approach to specifying preferences over conflicting facts and querying inconsistent knowledge bases. Extended version with appendix available at: https://arxiv.org/abs/2508.07742.

Bienvenu, M.; Bourgaux, C.; and Goasdoué, F. 2014. Querying inconsistent description logic knowledge bases under preferred repair semantics. In *Proceedings of AAAI*.

Bienvenu, M. 2020. A short survey on inconsistency handling in ontology-mediated query answering. *Künstliche Intelligenz* 34(4):443–451.

Bourgaux, C. 2016. *Inconsistency Handling in Ontology-Mediated Query Answering. (Gestion des incohérences pour l'accès aux données en présence d'ontologies)*. Ph.D. Dissertation, University of Paris-Saclay, France.

Bourgaux, C. 2024. Querying inconsistent prioritized data (abstract of invited talk). In *Proceedings of DL*.

Brewka, G., and Eiter, T. 1999. Preferred answer sets for extended logic programs. *Artif. Intell.* 109(1-2):297–356.

Calautti, M.; Greco, S.; Molinaro, C.; and Trubitsyna, I. 2022. Preference-based inconsistency-tolerant query answering under existential rules. *Artif. Intell.* 312:103772.

Calvanese, D.; Giacomo, G. D.; Lembo, D.; Lenzerini, M.; and Rosati, R. 2007. Tractable reasoning and efficient query answering in description logics: The DL-Lite family. *J. Autom. Reason.* 39(3):385–429.

Chortaras, A.; Trivela, D.; and Stamou, G. 2011. Optimized query rewriting for OWL 2 QL. In *Proceedings of CADE*.

Cima, G.; Lembo, D.; Marconi, L.; Rosati, R.; and Savo, D. F. 2021. Controlled query evaluation over prioritized ontologies with expressive data protection policies. In *Proceedings of ISWC*.

Du, J.; Qi, G.; and Shen, Y. 2013. Weight-based consistent query answering over inconsistent SHIQ knowledge bases. *Knowl. Inf. Syst.* 34(2):335–371.

Egly, U.; Gaggl, S. A.; and Woltran, S. 2008. ASPARTIX: implementing argumentation frameworks using answer-set programming. In *Proceedings of ICLP*.

Eiter, T.; Fink, M.; Greco, G.; and Lembo, D. 2008. Repair localization for query answering from inconsistent databases. *ACM Trans. Database Syst.* 33(2):10:1–10:51.

Fagin, R.; Kimelfeld, B.; Reiss, F.; and Vansummeren, S. 2016. Declarative cleaning of inconsistencies in information extraction. *ACM Trans. Database Syst.* 41(1):6:1–6:44.

Gebser, M.; Kaufmann, B.; Kaminski, R.; Ostrowski, M.; Schaub, T.; and Schneider, M. 2011. Potassco: The potsdam answer set solving collection. *AI Commun.* 24(2):107–124.

Gebser, M.; Kaminski, R.; Kaufmann, B.; and Schaub, T. 2012. *Answer Set Solving in Practice*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers.

Greco, G.; Greco, S.; and Zumpano, E. 2003. A logical framework for querying and repairing inconsistent databases. *IEEE Trans. Knowl. Data Eng.* 15(6):1389–1408.

Kimelfeld, B.; Livshits, E.; and Peterfreund, L. 2017. Detecting ambiguity in prioritized database repairing. In *Proceedings of ICDT*.

Kimelfeld, B.; Livshits, E.; and Peterfreund, L. 2020. Counting and enumerating preferred database repairs. *Theor. Comput. Sci.* 837:115–157.

Lifschitz, V. 2019. *Answer Set Programming*. Springer.

Livshits, E., and Kimelfeld, B. 2017. Counting and enumerating (preferred) database repairs. In *Proceedings of PODS*.

Lopatenko, A., and Bertossi, L. E. 2007. Complexity of consistent query answering in databases under cardinality-based and incremental repair semantics. In *Proceedings of ICDT*.

Lukasiewicz, T.; Malizia, E.; and Molinaro, C. 2023. Complexity of inconsistency-tolerant query answering in datalog+/- under preferred repairs. In *Proceedings of KR*.

Lukasiewicz, T.; Martinez, M. V.; and Simari, G. I. 2013. Preference-based query answering in datalog+/- ontologies. In *Proceedings of IJCAI*.

Lutz, C.; Seylan, I.; Toman, D.; and Wolter, F. 2013. The combined approach to OBDA: Taming role hierarchies using filters. In *Proceedings of ISWC*.

Manna, M.; Ricca, F.; and Terracina, G. 2013. Consistent query answering via ASP from different perspectives: Theory and practice. *Theory Pract. Log. Program.* 13(2):227–252.

Martinez, M. V.; Parisi, F.; Pugliese, A.; Simari, G. I.; and Subrahmanian, V. S. 2014. Policy-based inconsistency man-

agement in relational databases. *Int. J. Approx. Reason.* 55(2):501–528.

Rosati, R. 2011. On the complexity of dealing with inconsistency in description logic ontologies. In *Proceedings of IJCAI*.

Sakama, C., and Inoue, K. 2000. Prioritized logic programming and its application to commonsense reasoning. *Artif. Intell.* 123(1-2):185–222.

Staworko, S.; Chomicki, J.; and Marcinkowski, J. 2012. Prioritized repairing and consistent query answering in relational databases. *Annals of Mathematics and Artificial Intelligence (AMAI)* 64(2-3):209–246.

Stefanidis, K.; Koutrika, G.; and Pitoura, E. 2011. A survey on representation, composition and application of preferences in database systems. *ACM Trans. Database Syst.* 36(3):19:1–19:45.