An Incremental Algorithm for Computing All Repairs in Inconsistent Knowledge Bases

Bruno Yun[1] and Madalina Croitoru[2]

[1]University of Aberdeen
[2]University of Montpellier

Abstract. Repair techniques are used for reasoning in presence of inconsistencies. Such techniques rely on optimisations to avoid the computation of all repairs while certain applications need the generation of all repairs. In this paper, we show that the problem of all repair computation is not trivial in practice. To account for a scalable solution, we provide an incremental approach for the computation of all repairs when the conflicts have a cardinality of at most three. We empirically study its performance on generated knowledge bases (where the knowledge base generator could be seen as a secondary contribution in itself).

Keywords: Repairs · Knowledge Base · Existential Rule

1 Introduction

We place ourselves in the context of reasoning with knowledge bases (KBs) expressed using Datalog± [11] and investigate *inconsistent KBs*, i.e. KBs with the inconsistency solely stemming from the factual level and a coherent ontology. For instance, a prominent practical application in this setting is Ontology Based Data Access (OBDA) [22] that considers the querying of multiple heterogeneous data sources via an unifying ontology. With few exceptions [7], approaches performing query answering under inconsistency in the aforementioned setting rely on repairs [3]. Repairs, originally defined for database approaches [1] are maximal subsets of facts consistent with the ontology. Inconsistency tolerant semantics [21] avoid the computational overhead of computing all repairs by various algorithmic strategies [10]. Unfortunately, certain tasks need the repair enumeration problem, such as inconsistency-based repair ranking frameworks [26] or argumentation-based decision-making [12,25].

We focus on the problem of *computing possibly some or all repairs from Datalog± inconsistent KBs*. Our proposal relies on the notion of conflict (i.e. set of facts that trigger an inconsistency). Although approaches exist for computing conflicts in SAT instances or propositional logic [16,17], there are few works addressing conflict computation for Datalog± that come with additional challenges given the expressivity of the language [23]. In this paper we extend the state of the art with a *computationally efficient manner* to generate the set of all repairs using an incremental algorithm adapted from stable set computation in hypergraphs [9]. To this end, we make the hypothesis that the KB

allows for bounded sized conflicts (limited here at three). Our proposed algorithm, in a first step, finds the conflicts of the KBs using specific sequences of directed hyperedges (derivations) of the Graph of Atom Dependency (GAD) [19] leading to *falsum*. In the second step, we use an efficient incremental algorithm for finding all repairs of a set of facts from the set of conflicts of a given KB. This efficient algorithm was inspired by the problem of extending a given list of maximal independent sets in hypergraphs when hyperedges have a bounded dimension [9]. The aforementioned graph theoretical problem was proven to be in the NC complexity class if the size of the hyperedges were bounded by three [9,13] which means that the task can be efficiently solved on a parallel computer where processors are permitted to flip coins. Please note that although conflicts of size more than three can easily occur even when the arity of the negative constraints is limited to two, we do not find that this condition is limiting as, in reality, it is not unlikely to find KBs with only conflicts of size two.

Therefore, the proposed algorithm is more efficient than the approach of [23] for two reasons: (1) We do not compute all the "causes" and "consequences" of all the atoms and restrict ourselves to the derivations that lead to an inconsistency. (2) We use an efficient algorithm for incrementally computing repairs from conflicts.

When implementing our technique we noticed two key aspects of our approach: (1) getting some repairs from a KB can be relatively easy as the average number of repairs found during the allotted time did not change when the KB grew and (2) finding the last repairs was comparatively harder than the first repairs. Please note that although the computational problem of getting all repairs is in EXPTIME as the number of repairs can be exponential w.r.t. the number of facts [8], the proposed algorithm has a two fold significance: (1) it improves upon the state of the art for the task of all repair computation and (2) it is a viable alternative for applications that require the *repair enumeration*.

2 Background notions

We introduce some notions of the Datalog± language. A *fact* is a ground atom of the form $p(t_1, \ldots, t_k)$ where p is a predicate of arity k and for every $i \in \{1, \ldots, k\}, t_i$ is a constant. An existential *rule* is of the form $r = \forall \overrightarrow{X}, \overrightarrow{Y}\ B[\overrightarrow{X}, \overrightarrow{Y}] \rightarrow \exists \overrightarrow{Z}\, H[\overrightarrow{Z}, \overrightarrow{X}]$ where B (called the body) and H (called the head) are existentially closed atoms or conjunctions of existentially closed atoms and $\overrightarrow{X}, \overrightarrow{Y}, \overrightarrow{Z}$ their respective vectors of variables. A *rule is applicable* on a set of facts $\mathcal{F}$ iff there exists a homomorphism from the body of the rule to $\mathcal{F}$. Applying a rule to a set of facts (also called *chase*) consists of adding the set of atoms of the conclusion of the rule to the facts according to the application homomorphism. Different *chase* mechanisms use different simplifications that prevent infinite redundancies [5]. We use recognisable classes of existential rules where the chase is guaranteed to stop [5]. A *negative constraint* is a rule of the form $\forall \overrightarrow{X}, \overrightarrow{Y}\ B[\overrightarrow{X}, \overrightarrow{Y}] \rightarrow \bot$ where B is an existentially closed atom or conjunctions of existentially closed atoms, $\overrightarrow{X}, \overrightarrow{Y}$, their respective vectors of variables and $\bot$ is *falsum*.

Definition 1. *A KB is a tuple* $\mathcal{K} = (\mathcal{F}, \mathcal{R}, \mathcal{N})$ *where* $\mathcal{F}$ *is a finite set of facts,* $\mathcal{R}$ *a set of rules and* $\mathcal{N}$ *a set of negative constraints.*

Example 1. Let $\mathcal{K} = (\mathcal{F}, \mathcal{R}, \mathcal{N})$ with $\mathcal{F} = \{a(m), b(m), c(m), d(m), e(m), f(m), g(m), h(m), i(m), j(m)\}$, $\mathcal{R} = \{\forall x(f(x) \wedge h(x) \rightarrow k(x)), \forall x(i(x) \wedge j(x) \rightarrow l(x))\}$ and $\mathcal{N} = \{\forall x(a(x) \wedge b(x) \wedge c(x) \rightarrow \bot), \forall x(c(x) \wedge d(x) \rightarrow \bot), \forall x(e(x) \wedge f(x) \wedge d(x) \rightarrow \bot), \forall x(e(x) \wedge f(x) \rightarrow \bot), \forall x(i(x) \wedge k(x) \rightarrow \bot), \forall x(l(x) \wedge h(x) \rightarrow \bot)\}$.

In a KB $\mathcal{K} = (\mathcal{F}, \mathcal{R}, \mathcal{N})$, the saturation $\mathcal{S}at_{\mathcal{R}}(X)$ of a set of facts X is the set of atoms obtained after successively applying the set of rules $\mathcal{R}$ on X until a fixed point. A set $X \subseteq \mathcal{F}$ is $\mathcal{R}$-inconsistent iff *falsum* can be entailed from the saturation of X by $\mathcal{R} \cup \mathcal{N}$, i.e. $\mathcal{S}at_{\mathcal{R}\cup\mathcal{N}}(X) \models \bot$. A conflict of a KB is a minimal $\mathcal{R}$-inconsistent subset of facts.

Definition 2. *Let us consider* $\mathcal{K} = (\mathcal{F}, \mathcal{R}, \mathcal{N})$. $X \subseteq \mathcal{F}$ *is a conflict of* $\mathcal{K}$ *iff* $\mathcal{S}at_{\mathcal{R}\cup\mathcal{N}}(X) \models \bot$ *and for every* $X' \subset X$, $\mathcal{S}at_{\mathcal{R}\cup\mathcal{N}}(X') \not\models \bot$.

The set of all conflicts of $\mathcal{K}$ is denoted $Conflict(\mathcal{K})$.

Example 2. [Cont'd Example 1] We have $Conflict(\mathcal{K}) = \{\{a(m), b(m), c(m)\}, \{c(m), d(m)\}, \{e(m), f(m)\}, \{f(m), h(m), i(m)\}, \{h(m), i(m), j(m)\}\}$. The set $\{d(m), e(m), f(m)\}$ is not a conflict since $\mathcal{S}at_{\mathcal{R}\cup\mathcal{N}}(\{e(m), f(m)\}) \models \bot$.

To practically compute the conflicts, we use a special directed hypergraph [14] called the Graph of Atom Dependency (GAD) and defined by [19].

Definition 3. *Given a KB* $\mathcal{K} = (\mathcal{F}, \mathcal{R}, \mathcal{N})$*, the GAD of* $\mathcal{K}$*, denoted by* $GAD_{\mathcal{K}}$*, is a pair* (V, D) *such that* V *is the set of atoms in* $\mathcal{S}at_{\mathcal{R}\cup\mathcal{N}}(\mathcal{F})$ *and* $D = \{(U, W) \in 2^V \times 2^V$ *s.t. there exists* $r \in \mathcal{R} \cup \mathcal{N}$ *and a homomorphism* π *such that* W *is obtained by applying* r *on* U *using* $\pi\}$.

Example 3. [Cont'd Example 1] Here, we have that $V = \{\bot, a(m), b(m), c(m), \ldots, l(m)\}$ and $D = \{D_1, D_2, \ldots, D_8\}$ where $D_1 = (\{f(m), h(m)\}, \{k(m)\}), D_2 = (\{i(m)\}, j(m)\}, \{l(m)\}), D_3 = (\{a(m), c(m), b(m)\}, \{\bot\}), D_4 = (\{c(m), d(m)\}, \{\bot\}), D_5 = (\{k(m), i(m)\}, \{\bot\}), D_6 = (\{h(m), l(m)\}, \{\bot\}), D_7 = (\{e(m), f(m)\}, \{\bot\})$ and $D_8 = (\{d(m), e(m), f(m)\}, \{\bot\})$.

A derivation is a sequence of rule applications such that each rule can be applied successively. A derivation for a specific atom a is a finite minimal sequence of rule applications starting from a set of facts and ending with a rule application that generates a. We now define the notion of fix and repair w.r.t. a set $Y \subseteq 2^{\mathcal{F}}$ of a KB $\mathcal{K} = (\mathcal{F}, \mathcal{R}, \mathcal{N})$, that will be used in the proposed algorithms.

Definition 4. *Let* $\mathcal{F}$ *be a set of facts,* $F \subseteq \mathcal{F}$ *is a fix of* $\mathcal{F}$ *w.r.t.* $Y \subseteq 2^{\mathcal{F}}$ *iff for every* $X \in Y, X \cap F \neq \emptyset$. *A fix* F *of* $\mathcal{F}$ *w.r.t.* Y *is called a minimal fix of* $\mathcal{F}$ *w.r.t.* Y *iff for all* $F' \subset F$*, it holds that* F' *is not a fix of* $\mathcal{F}$ *w.r.t.* Y. *The set of all minimal fixes of* $\mathcal{F}$ *w.r.t.* Y *is denoted by* $MFix(\mathcal{F}, Y)$.

The notion of KB reparation is linked to that of conflict via the minimal fixes. We define a repair of a set of facts $\mathcal{F}$ w.r.t. a set $Y \subseteq 2^{\mathcal{F}}$.

Definition 5. *Let $\mathcal{F}$ be a set of facts, $X \subseteq \mathcal{F}$ is a repair of $\mathcal{F}$ w.r.t. $Y \subseteq 2^{\mathcal{F}}$ iff there exists $F \in MFix(\mathcal{F}, Y)$ such that $X = \mathcal{F} \setminus F$.*

Please note that there is a bijection between the set of repairs and the set of minimal fixes of a set $\mathcal{F}$ w.r.t. a set $Y \subseteq 2^{\mathcal{F}}$. We denote the set of all repairs of $\mathcal{F}$ w.r.t. Y by $Repair(\mathcal{F}, Y)$. Moreover, the repairs of $\mathcal{F}$ w.r.t. $Conflict(\mathcal{K})$ are the maximal, for set inclusion, consistent subsets of $\mathcal{F}$.

3 Repairs Generation

In this section, we detail a framework for computing all maximal consistent sets of a KB. The approach is given in Algorithm 1 and composed of two steps:

1. **(Conflicts Generation)** First, the GAD is constructed. Then, all conflicts are computed by extracting the facts used in the minimal derivations for $\bot$.
2. **(From Conflicts to Repairs)** Second, repairs of $\mathcal{F}$ w.r.t. $Conflict(\mathcal{K})$ are constructed using the *FindAllRepairs* call. The provided algorithm is efficient for computing repairs in the case where the conflicts are at most of size 3.

Algorithm 1: Finding all maximal consistent sets of $\mathcal{K}$

input : A KB $\mathcal{K} = (\mathcal{F}, \mathcal{R}, \mathcal{N})$
output: A set I of repairs of $\mathcal{F}$ w.r.t. $Conflict(\mathcal{K})$

```
1 GAD_K ← GADConstructor(K);
2 C ← FindAllConflicts(K, GAD_K);
3 Result ← FindAllRepairs(K, C);
4 return Result;
```

Conflicts are subsets of $\mathcal{F}$ but there is not always a negative constraint directly triggered by the conflict (since the actual "clash" can be between the atoms generated using rules). These two kinds of conflicts are referred to as "conflicts" and "naive conflicts" in Rocher [23]. The use of the GAD allows to keep track of all the rule applications and to propagate those "clashes" into $\mathcal{F}$[1].

FindAllConflicts takes as input a KB $\mathcal{K}$ and $GAD_{\mathcal{K}}$ and outputs the set of all conflicts of this KB. It is based on three steps: (1) It builds the GAD. This step has been proven to be efficient as the GAD can be constructed alongside the chase [18]. (2) The GAD is used for finding the set of all possible minimal derivations for $\bot$. (3) For each derivation for $\bot$, the facts in $\mathcal{F}$ that enabled the generation of this derivation are extracted. They correspond to conflicts of $\mathcal{K}$.

Example 4 (Cont'd Example 2). The set of all minimal derivations for $\bot$ is $\{(D_2, D_6), (D_1, D_5), (D_7), (D_4), (D_3)\}$. The set of conflicts is $\{\{h(m), i(m), j(m)\}, \{f(m), h(m), i(m)\}, \{e(m),\ f(m)\},\ \{c(m),\ d(m)\}, \{a(m), b(m), c(m)\}\}$.

[1] The algorithms avoid the problem of derivation loss [19] which is important for the completeness of our approach. Note that in Hecham et al. [19] the authors discuss how finding all derivations for an atom is practically feasible despite the problem being exponential for combined complexity but polynomial for data complexity.

3.1 From Conflicts to Repairs

Our approach for computing the set of maximal consistent sets of a KB from the set of conflicts is composed of four algorithms: *FindAllRepairs*, *NewMinimalFix*, *FindRepairs* and *SubRepair*. In order to compute the repairs of $\mathcal{F}$ w.r.t. the set of conflicts of $\mathcal{K}$, we need to first compute the set of all minimal fixes of $\mathcal{F}$ w.r.t. $Conflict(\mathcal{K})$. *FindAllRepairs* computes the repairs of $\mathcal{F}$ w.r.t. $Conflict(\mathcal{K})$ by iteratively computing the set of all minimal fixes of $\mathcal{F}$ w.r.t. $Conflict(\mathcal{K})$ before converting them into repairs of $\mathcal{F}$ w.r.t. $Conflict(\mathcal{K})$. More precisely, $FindAllRepairs$ repeatedly calls *NewMinimalFix* which returns a new minimal fix of $\mathcal{F}$ w.r.t. $Conflict(\mathcal{K})$ not previously found. The idea behind $NewMinimalFix$ is that it produces new sets (U and A) depending on the minimal fixes of $\mathcal{F}$ w.r.t. $Conflict(\mathcal{K})$ that were previously found. A repair for U w.r.t. A can be modified in order to return a new fix of $\mathcal{F}$ w.r.t. $Conflict(\mathcal{K})$. Lastly, *FindRepair* computes a repair for U w.r.t. a set $A \subseteq 2^U$ by relying on *SubRepair* for iteratively constructing the repair. In the rest of this section, we detail the general outline of *FindAllRepairs* and *NewMinimalFix*.

FindAllRepairs (see Algorithm 2) takes as input a KB $\mathcal{K}$ and its set of conflicts $Conflict(\mathcal{K})$ and returns the set of all repairs of $\mathcal{F}$ w.r.t. $Conflict(\mathcal{K})$. The sets B and I contain the set of minimal fixes and repairs of $\mathcal{F}$ w.r.t. $Conflict(\mathcal{K})$ respectively and are initially empty. $NewMinimalFix(\mathcal{K}, Conflict(\mathcal{K}), B)$ is called for finding a new minimal fix of $\mathcal{F}$ w.r.t. $Conflict(\mathcal{K})$ that is not contained in B. If $NewMinimalFix(\mathcal{K}, Conflict(\mathcal{K}), B)$ returns the empty set then B already contains all possible minimal fixes of $\mathcal{F}$ w.r.t. $Conflict(\mathcal{K})$. Otherwise, the new minimal fix of $\mathcal{F}$ w.r.t. $Conflict(\mathcal{K})$ is stored in B and converted into a repair of $\mathcal{F}$ w.r.t. $Conflict(\mathcal{K})$ that is stored in I.

Algorithm 2: FindAllRepairs

```
input : A KB K = (F, R, N) and a set of conflicts Conflict(K)
output: A set I of repairs of F w.r.t. Conflict(K)
1 B ← ∅, I ← ∅, stops ← false;
2 while stops = false do
3 |   MF ← NewMinimalFix(K, Conflict(K), B);
4 |   if MF = ∅ then
5 |   |   stops = true;
6 |   else
7 |   |   B ← B ∪ {MF};
8 |   |   I ← I ∪ (F \ MF);
9 return I;
```

NewMinimalFix (see Algorithm 3) takes as input a KB $\mathcal{K}$, the corresponding set of conflicts $Conflict(\mathcal{K})$ and a set of minimal fixes B of $\mathcal{F}$ w.r.t. $Conflict(\mathcal{K})$ and returns the empty set if B contains all the minimal fixes of $\mathcal{F}$ w.r.t. $Conflict(\mathcal{K})$, otherwise it returns a new minimal fix of $\mathcal{F}$ w.r.t. $Conflict(\mathcal{K})$ that is not contained in B. First, the facts in $\mathcal{F}$ that are not in any conflict of $\mathcal{K}$ are removed. By definition, these facts cannot be in a minimal fix

of $\mathcal{F}$ w.r.t. $Conflict(\mathcal{K})$. Then, we check whether or not each fact is at least in one element of B. If this is not the case, we can build a minimal fix of $\mathcal{F}$ w.r.t. $Conflict(\mathcal{K})$ that is not in B (line 6 to 10). To do so, we first pick an arbitrary fact u that is not in any set of B. Then, we pick an arbitrary conflict A_u containing u and find a repair Rep of U w.r.t. $\mathcal{A}'$ where $\mathcal{A}'$ is the set of restricted conflicts by U[2] where $U = \mathcal{F} \setminus A_u$. The resulting $U \setminus Rep$ is a minimal fix of U w.r.t. $\mathcal{A}'$. It is extended to a minimal fix of $\mathcal{F}$ w.r.t. $Conflict(\mathcal{K})$ by adding the fact u. It can thus be added to B as the first minimal fix containing u.

Example 5 (Cont'd Example 4). At step 1 in Table 1, $g(m)$ is removed because it is in no conflicts. Then, since $\bigcup B = \emptyset$ is included in $\mathcal{F}$, an arbitrary fact $u = a(m)$ in $\mathcal{F}$ is picked. Then, an arbitrary conflict $A_u = \{a(m), b(m), c(m)\}$ that contains u is selected. $U = \mathcal{F} \setminus A_u$ is $\{d(m), e(m), f(m), h(m), i(m), j(m)\}$ and the restricted set of conflict by U is $\{\{d(m)\}, \{e(m), f(m)\}, \{f(m), h(m), i(m)\}, \{h(m),\ i(m),\ j(m)\}\}$. The repair of U w.r.t. $\mathcal{A}'$ returned is $\{j(m), i(m), f(m)\}$ which means that $\{d(m), e(m), h(m)\}$ is a minimal fix of U w.r.t. $\mathcal{A}'$. We conclude that the set $\{a(m), d(m), e(m), h(m)\}$ is a minimal fix of $\mathcal{F}$ w.r.t. $Conflict(\mathcal{K})$. This process is repeated by iteratively selecting the facts $b(m), c(m), f(m), i(m)$ and $j(m)$. As the reader can note, after step 6, we have that $\bigcup B = \mathcal{F}$.

If each fact is at least in one element of B then each conflict a' of $\mathcal{K}$ is a fix of $\mathcal{F}$ w.r.t. B. However, if a' is not a minimal fix of $\mathcal{F}$ w.r.t. B then we can find u such that $a' \setminus \{u\}$ is still a fix of $\mathcal{F}$ w.r.t. B (line 13 to 17). We use the previous method and find a repair Rep of U w.r.t. $\mathcal{A}'$ where $\mathcal{A}'$ is the restricted set of conflicts by U with $U = \mathcal{F} \setminus (a' \setminus \{u\})$. $U \setminus Rep$ is thus a minimal fix of U w.r.t. $\mathcal{A}'$ but also a minimal fix of $\mathcal{F}$ w.r.t. $Conflict(\mathcal{K})$. In the example, we skipped this as each a' in $Conflict(\mathcal{K})$ is a minimal fix of $\mathcal{F}$ w.r.t. B.

In the case where each fact is at least in one minimal fix of $\mathcal{F}$ w.r.t. B and each conflict is a minimal fix of $\mathcal{F}$ w.r.t. B, we can still find new minimal fix $\mathcal{F}$ w.r.t. $Conflict(\mathcal{K})$ that is not in B (line 18 to 28). We first find subsets S of $\mathcal{F}$ that have a size equal or less than 3[3], that are not be included in any conflict of $\mathcal{K}$ and such that any conflict of $\mathcal{K}$ are not be included in S. If there is a set S that satisfies every aforementioned conditions then it can be extended into a minimal fix of B. If that is the case, the set Z does not contain any conflict of $\mathcal{K}$ [9]. Then, we use the previous method and find a repair Rep of U w.r.t. $\mathcal{A}'$ where $\mathcal{A}'$ is the set of restricted conflicts by U with $U = \mathcal{F} \setminus (V \setminus Z)$. $U \setminus Rep$ is a minimal fix of U w.r.t. $\mathcal{A}'$ but also a minimal fix of $\mathcal{F}$ w.r.t. $Conflict(\mathcal{K})$ (line 28) because Z does not contain any conflict of $\mathcal{K}$. Note that this algorithm relies on Algorithm 4 for finding a repair w.r.t. some restricted conflicts.

Example 6 (Cont'd Example 5). Let us consider $S = \{c(m), e(m)\}$ at step 7 in Table 1. We have $|S| \leq 3$, $S \not\subseteq a$ and $a \not\subseteq S$ for every $a \in Conflict(\mathcal{K})$.

[2] The set of restricted conflicts by a set U is the set containing each intersection of a conflict with U. Namely, it is equal to $\{X \cap U \mid X \in Conflict(\mathcal{K})\}$.

[3] The computational problem of finding a single repair of $\mathcal{F}$ w.r.t. a set $Y \subseteq 2^{\mathcal{F}}$ is only in the NC complexity class when every $y \in Y$ is such that $|y| \leq 3$, otherwise it has been proven to be in the RNC complexity class [6,20].

Algorithm 3: NewMinimalFix

input : A KB $\mathcal{K} = (\mathcal{F}, \mathcal{R}, \mathcal{N})$, the corresponding set of conflicts $Conflict(\mathcal{K})$ and a set B of minimal fixes of $\mathcal{F}$ w.r.t. $Conflict(\mathcal{K})$
output: Either a new minimal fix of $\mathcal{F}$ w.r.t. $Conflict(\mathcal{K})$ that is not in B or $\emptyset$ if B contains all of them

$V \leftarrow \mathcal{F}$;
for $v \in V$ **do**
  if *there is no* $a \in Conflict(\mathcal{K})$ *such that* $v \in a$ **then**
    $V \leftarrow V \setminus \{v\}$;
if $\bigcup B \subset V$ **then**
  $u \leftarrow$ random fact in $V \setminus \bigcup B$;
  $A_u \leftarrow$ random conflict in $Conflict(\mathcal{K})$ that contains u;
  $U \leftarrow V \setminus A_u$;
  $\mathcal{A}' \leftarrow \{a \cap U \mid a \in Conflict(\mathcal{K}), u \notin a\}$;
  return $\{u\} \cup (U \setminus \texttt{FindRepair}(\mathcal{A}', U))$;
else
  for $a' \in Conflict(\mathcal{K})$ **do**
    if a' *is not a minimal fix of* $\mathcal{F}$ *w.r.t.* B **then**
      $u \leftarrow$ fact in a' s.t. a' is still a fix of $\mathcal{F}$ w.r.t. B after its removal;
      $U \leftarrow V \setminus (a' \setminus \{u\})$;
      $A' \leftarrow \{a \cap U \mid a \in Conflict(\mathcal{K})\}$;
      return $U \setminus \texttt{FindRepair}(\mathcal{A}', U)$;
  for $S \subseteq V, |S| \leq 3, S \not\subseteq a$ *and* $a \not\subseteq S$, *for every* $a \in Conflict(\mathcal{K})$ **do**
    for $v \in S$ **do**
      $BS_v \leftarrow \{X \in B$ such that $B \cap S = \{v\}\}$;
    $BS_0 \leftarrow \{X \in B$ such that $B \cap S = \emptyset\}$;
    for $\{B_v \mid v \in S\} \subseteq \prod_{v \in S} BS_v$ **do**
      if $B_v \neq \emptyset$ *for every* $v \in S$ **then**
        if *for every* $X \in BS_0, X \not\subseteq \bigcup\limits_{v \in S} B_v$ **then**
          $Z \leftarrow S \cup \left(V \setminus \bigcup\limits_{v \in S} B_v\right)$;
          $U \leftarrow V \setminus Z$;
          $\mathcal{A}' \leftarrow \{a \cap U \mid a \in Conflict(\mathcal{K})\}$;
          return $U \setminus \texttt{FindRepair}(\mathcal{A}', U)$;
  return $\emptyset$;

We have $BS_{c(m)} = \{\{c(m), h(m), f(m)\}, \{c(m), f(m), j(m)\}\}$ and $BS_{e(m)} = \{\{d(m), e(m), h(m), a(m)\}, \{d(m), e(m), h(m), b(m)\}\}$. Since $BS_0 = \emptyset$, for every $X \in BS_0, X \not\subseteq \bigcup E$ where $E = \{\{d(m), e(m), h(m), a(m)\}, \{c(m), h(m), f(m)\}\} \in BS_{c(m)} \times BS_{e(m)}$. Thus, we have $Z = \{c(m), e(m), b(m),\ i(m), j(m)\}$, $U = \{a(m), d(m), f(m), h(m)\}$ and $\mathcal{A}' = \{\{a(m)\},\ \{d(m)\}, \{f(m)\}, \{f(m), h(m)\},\ \{h(m)\}\}$. The only repair of U w.r.t. $\mathcal{A}'$ is $\emptyset$. We conclude that the set U is a minimal fix of $\mathcal{F}$ w.r.t. $Conflict(\mathcal{K})$.

Step	New elements of B	New elements of I
1	$\{d, e, h, a\}$	$\{b, c, f, g, i, j\}$
2	$\{d, e, h, b\}$	$\{a, c, f, g, i, j\}$
3	$\{e, h, c\}$	$\{a, b, d, f, g, i, j\}$
4	$\{c, h, f\}$	$\{a, b, d, e, g, i, j\}$
5	$\{c, e, i\}$	$\{a, b, d, f, g, h, j\}$
6	$\{c, f, j\}$	$\{a, b, d, e, g, h, i\}$
7	$\{a, d, f, h\}$	$\{b, c, e, g, i, j\}$
8	$\{b, d, f, h\}$	$\{a, c, e, g, i, j\}$
⋮	⋮	⋮

Table 1. List of minimal fixes and repairs for $\mathcal{F}$ w.r.t. $Conflict(\mathcal{K})$ found at each step.

3.2 Generating a Repair Efficiently

We show how to efficiently find a single repair of U w.r.t. $\mathcal{A} \subseteq 2^U$ when $|a| \leq 3$ for every $a \in \mathcal{A}$. The problem of finding a single repair is in the NC complexity class (but as soon as $|a| > 3$, it falls into the RNC complexity class [6,20]). *FindRepair* gradually build a repair by successively finding subrepairs C of large size with *SubRepair* and by restricting U and $\mathcal{A}$[4]. We now detail the general outline of the two algorithms *FindRepair* and *SubRepair*.

FindRepair (see Algorithm 4) takes as input a set of facts U and a set $\mathcal{A} \subseteq 2^U$ and returns a repair of U w.r.t. $\mathcal{A}$. We first initialise the algorithm with $\mathcal{A}' = \mathcal{A}, V' = U$ and $I = \emptyset$ (I will eventually be the repair of U w.r.t. $\mathcal{A}$). As long as the set $\mathcal{A}'$ is not empty, we update $\mathcal{A}'$ and V' by removing the facts found in a large subset of a repair of V' w.r.t. $\mathcal{A}'$. The two for-loop at line 10 and 13 remove supersets and sets of size one that may arise in $\mathcal{A}'$. The reason behind the removal of supersets is that they do not change the sets of repairs obtained. Furthermore, at line 14, we remove facts that are in a set of size one in $\mathcal{A}'$ because they cannot be in any repairs. Finally, when $\mathcal{A}' = \emptyset$, I is returned with the remaining facts in V'.

SubRepair (see Algorithm 5) is used for finding a large subrepair of U w.r.t. $\mathcal{A} \subseteq 2^U$. This algorithm uses two constants d_0 and d_1. When these constants are initialised with $d_0 = 0.01$ and $d_1 = 0.25$, [13] showed that SubRepair returns either a subrepair j such that $|j \cup N(j, \mathcal{A}, U)| \geq d_0 \times \frac{p}{\log(p)}$ or a subrepair that is at least of size $d_0 \times \frac{p}{\log(p)}$ where p is the size of V'. SubRepair maintains a collection of sets J initialised with sets of the form $\{v\}$ for all facts $v \in U$ such that $\{v\}$ is not a set in $\mathcal{A}$. The sets in J are subrepairs and will remain mutually disjoint throughout the algorithm. The algorithm iteratively checks if there is a set in J that is large enough to be returned and if not, it will select which sets of J should be merged and merge them (lines 12 to 21). However, merging subrepairs does not always produce a subrepair, that is why it removes some facts after the merging. Although the procedure for finding a matching M of Q at line 18 is not described, the reader can find it in [15]. Note that the functions N and D are defined as $N(C, \mathcal{A}, U) = \{u \in U \setminus C \mid \exists a \in \mathcal{A}, a \setminus C = \{u\}\}$ and $D(C, C', \mathcal{A}, U) =$

[4] A subrepair of U w.r.t. $\mathcal{A}$ is a subset of a repair of U w.r.t. $\mathcal{A}$.

Algorithm 4: FindRepair

input : A set of facts U and a set $\mathcal{A} \subseteq 2^U$ such that $|a| \leq 3$ for every $a \in \mathcal{A}$
output: A repair of U w.r.t. $\mathcal{A}$

```
1  A' ← A, V' ← U, I ← ∅;
2  while A' ≠ ∅ do
3      C ← SubRepair(V', A');
4      I ← I ∪ C;
5      V' ← V' \ C;
6      for a' ∈ A' do
7          a' ← a' ∩ V';
8      for a' ∈ A' do
9          if there exists a'' ∈ A' such that a'' ⊂ a' then
10             A' ← A' \ {a'};
11     for a' ∈ A' do
12         if |a'| = 1 then
13             V' ← V' \ a';
14             A' ← A' \ {a'};
15 return I ∪ V';
```

$(N(C, \mathcal{A}, U) \cap C') \cup (N(C', \mathcal{A}, U) \cap C)$. It has been proven that SubRepair runs in $O(log^2 n)$ time on $n+m$ processors and since FindRepair makes at most $O(log^2 n)$ calls to SubRepair (because the subrepairs have a minimal size), FindRepair runs in time $O(log^4 n)$ on $n + m$ EREW processors.

Example 7 (Cont'd Example 6). Suppose that $U = \{d(m), e(m), f(m), h(m), i(m), j(m)\}$ and $\mathcal{A} = \{\{d(m)\}, \{e(m), f(m)\}, \{f(m), h(m), i(m)\}, \{h(m), i(m), j(m)\}\}$, SubRepair returns the set $\{j(m)\}$. We remove $j(m)$ from V' and $\mathcal{A}'$. Thus, $V' = \{d(m), e(m), f(m), h(m), i(m)\}$ and $\mathcal{A}' = \{\{d(m)\}, \{e(m), f(m)\}, \{f(m), i(m), h(m)\}, \{h(m), i(m)\}\}$. $\{f(m), i(m), h(m)\}$ and $\{d(m)\}$ are removed from $\mathcal{A}'$ because they are respectively a superset and a set of size one. The same process is repeated and *FindRepair* returns $\{i(m), j(m), f(m)\}$. Here, the condition $|j \cup N(j, \mathcal{A}, U)| \geq \frac{c_0 \times p}{log(p)}$ is always satisfied with $d_0 = 0.01$.

The approach is correct since (1) *FindAllConflicts* returns the set of all conflicts, (2) *FindAllRepairs* returns the set of all repairs of $\mathcal{F}$ w.r.t. $Conflict(\mathcal{K})$ [9] and (3) *FindRepair* returns a repair of U w.r.t. $\mathcal{A} \subseteq 2^U$ [13].

4 Evaluation

Since the benchmarks of [10] was too large to be handled by the algorithm with a reasonable duration, we created a generator for Datalog± KBs as follows:

1. **(Facts Generation)** We generates N_1 facts with a fixed probability p_0 of generating a new predicate. The arity of the predicates are randomly picked between two fixed constants c_0 and c_1. We used the idea that some constants

Algorithm 5: SubRepair

input : A set of facts U and a set $\mathcal{A} \subseteq 2^U$ such that $|a| \leq 3$ for every $a \in \mathcal{A}$
output: A subrepair of U w.r.t. $\mathcal{A}$

$Q \leftarrow \emptyset, p \leftarrow |U|, J \leftarrow \emptyset$;
for $u \in U$ **do**
  if $\{u\} \notin \mathcal{A}$ **then**
    $J \leftarrow J \cup \{\{u\}\}$;
$stops \leftarrow false$;
while $stops = false$ **do**
  for $j \in J$ **do**
    if $|j \cup N(j, \mathcal{A}, U)| \geq d_0 \times \frac{p}{log(p)}$ **then**
      $stops \leftarrow true$;
      $result \leftarrow j$;
    if $stops = false$ **then**
      for $j \in J$ **do**
        for $j' \in J$ **do**
          if $|D(j, j', \mathcal{A}, U)| \leq \frac{d_1 \times p}{|J| \times log(p)}$ **then**
            $Q \leftarrow Q \cup \{\{j, j'\}\}$;
      $M \leftarrow$ matching of Q of size $\lceil \left(\frac{1}{4} - 2\frac{d_0}{d_1}\right) \times |J| \rceil$;
      for $\{j, j'\} \in M$ **do**
        $J \leftarrow J \setminus j$;
        $J \leftarrow J \setminus j'$;
        $J \leftarrow J \cup \{(j \cup j') \setminus D(j, j', \mathcal{A}, U)\}$;
return $result$;

only appear at a specific position in a predicate. Thus, we linked multiple positions of atoms into groups that share a distinct pool of constants with a probability p_1. The size of each pool of constants is increased such that for every predicate $pred_i$, the product of the size of all the pools of constants of positions in predicate $pred_i$ is superior to the number of atoms with predicate $pred_i$. Last, we created the necessary constants and "filled" the positions of atoms with constants from the corresponding pool of constants such that N_1 distinct facts are generated.

2. **(Rules Generation)** We generate N_2 rules and the number of atoms in the head and body of each rule is picked between four fixed constants c_2, c_3, c_4 and c_5 respectively. In order to avoid infinite rule applications, we only use new predicates in the head of the rules. We used the idea that rules should be split in levels such that a rule in level i can be applied to the N_1 original facts but also to the atoms generated by the all the rules in the level $j < i$. Thus, we split the rules randomly in each level such that $|L_i| \geq 1$ for $i \in \{1, \ldots, c_6\}$ and $|\bigcup_{1 \leq i \leq c_6} L_i| = N_2$ where c_6 is a constant corresponding to the maximum level of a rule. In order to build the rules in level one, we randomly "filled" the

body of the rules with the N_1 ground atom that were previously generated. The heads of the rules are filled with atoms with new predicates but, within the same rule, there is a probability p_0 that the same predicate is reused. The positions of the new predicates in the head of a rule r_1 are also linked to the position of the predicates in the body of r_1 with a probability p_2. At this point, the atoms in the heads of the rules are also filled with constants from the corresponding pool of constants and each rule contains only ground atoms. Variables are added into the bodies of the rules by replacing constants with a probability p_3. There is a probability p_4 that a variable is reused when replacing a constant at a position belonging to the same group. Each constant in the heads of the rules is replaced by a variable that is used in the body at a position that belongs to the same group with a probability p_5. The process is repeated for the rules in level superior to one by filling the bodies of the rules with the N_1 original facts and the atoms in the head of the rules in inferior levels instead of only the original facts.

3. **(Negative constraints Generation)** We generate N_3 negative constraints and the number of atoms in the body of each negative constraint is randomly fixed between two fixed constants c_7 and c_8. In order to generate the negative constraints, one has to be careful not to make the set of rules incoherent, i.e. the union of the set of rules with the set of negative constraints has to be satisfiable. $GAD_{\mathcal{K}} = (V, D)$ is constructed on the KB with the facts and rules generated according to the aforementioned steps. For each ground atom $v \in V$, we compute the set $W_v = \bigcup_{S \in S_v} \bigcup_{D_i \in S} U(D_i)$ where S_v is the set of all possible derivations for v and D_i is a rule application in S. Thus, in order to create a negative constraint neg_1 of size K, we first pick an atom v_1 in V and add it in the body of neg_1. We then remove W_{v_1} from V and pick an atom v_2 in $V \setminus W_{v_1}$ such that every atom in the body of neg_1 does not belong to W_{v_2}, we add v_2 to the body of neg_1 and remove W_{v_2} from $V \setminus W_{v_1}$. The process is repeated until the body of neg_1 reaches the size K.

In order to generate a set of different KBs, we decided to vary some parameters and fix others. Namely: N_1 varies between $10, 100, 1000$ and 10000, p_1 varies between 0 and 0.05, N_2 varies between $10, 100$ and 200, c_6 varies between 1 and 3 and N_3 varies between $20, 40$ and 60. The other parameters were fixed such that $p_0 = 0.5$, $p_2 = 0.05$, $c_1 = 3$, $c_0 = c_2 = c_4 = 1$, $c_3 = c_5 = 2$, $p_3 = 0.7$ and $p_4 = p_5 = 0.8$. This resulted in the generation of 144 different combinations of the parameters and a total of 720 different KBs since we generated 5 different KBs for each combination. The generated KBs are in DLGP format [4], the tool for generating the KBs and the tool for computing all the repairs are all available online at: `https://gite.lirmm.fr/yun/Generated-KB`.

4.1 Evaluation Results

The algorithm for finding the repairs was launched on each KB with a timeout after 10 minutes. We recorded both the time for finding each repair and the time

for the conflict computation. We made the following observations: (1) On KBs with 10 facts, the tool successfully terminated with a total median time of 418 ms and an average of 11.7 repairs. Although all instances with 100, 1000 and 10000 facts did not finish before the timeout, the program returned an average number of 86.5 repairs by KB and this number seems to be independent of the number of facts. (2) The last repairs are harder to find: Across all KBs, the first repair takes an average of 763 ms to be computed whereas the last found repair took an average of 64461 ms. Lastly, finding all the conflicts takes a small part of the total computational time as it amounts to only an average of 12.8% of the total time. (3) On the one hand, the parameter c_6 does not impact the average number of repairs (66.8 with $c_6 = 1$ and 68.7 with $c_6 = 3$) but the median time for finding the conflicts does slightly increase (479 ms with $c_6 = 1$ and 800 ms with $c_6 = 3$). On the other hand, we noticed a sharp increase in the average time for finding the conflicts when the parameter N_3 is increased (888.1 ms for $N_3 = 20$, 1477.5 ms for $N_3 = 40$ and 3737.7 ms for $N_3 = 60$).

4.2 Conclusion

We showed an efficient incremental algorithm that allows for some or all repairs computation and empirically evaluated the proposed algorithm with a benchmark on inconsistent KBs expressed using Datalog±. We empirically showed that our approach is able to find the first repairs after a reasonable amount of time. We argue that our approach is useful for applications where an enumeration, even partial, of the repairs is necessary. For example, in life science applications such as biodegradable packaging selection [24] or wheat transformation [2] the repairs are used by the experts in order to enrich the KB with further information. In this setting, enumerating the set of repairs could be of practical value.

Last but not least, let us also highlight that the paper also provides a knowledge base generator, a contribution in itself for the OBDA community.

Acknowledgement

The second author acknowledges the support of the Docamex project, funded by the French Ministry of Agriculture.

References

1. Arenas, M., Bertossi, L.E., Chomicki, J.: Consistent Query Answers in Inconsistent Databases. In: Proceedings of the Eighteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, May 31 - June 2, 1999, Philadelphia, Pennsylvania, USA. pp. 68–79 (1999), http://doi.acm.org/10.1145/303976.303983

2. Arioua, A., Croitoru, M., Buche, P.: DALEK: A Tool for Dialectical Explanations in Inconsistent Knowledge Bases. In: Computational Models of Argument - Proceedings of COMMA 2016, Potsdam, Germany, 12-16 September, 2016. pp. 461–462 (2016), https://doi.org/10.3233/978-1-61499-686-6-461
3. Baget, J.F., Benferhat, S., Bouraoui, Z., Croitoru, M., Mugnier, M.L., Papini, O., Rocher, S., Tabia, K.: A General Modifier-Based Framework for Inconsistency-Tolerant Query Answering. In: Principles of Knowledge Representation and Reasoning: Proceedings of the Fifteenth International Conference, KR 2016, Cape Town, South Africa, April 25-29, 2016. pp. 513–516 (2016)
4. Baget, J.F., Gutierrez, A., Leclère, M., Mugnier, M.L., Rocher, S., Sipieter, C.: DLGP: An extended Datalog Syntax for Existential Rules and Datalog+/- Version 2.0 (Jun 2015)
5. Baget, J.F., Leclère, M., Mugnier, M.L., Salvat, E.: On rules with existential variables: Walking the decidability line. Artif. Intell. **175**(9-10), 1620–1654 (2011)
6. Beame, P., Luby, M.: Parallel Search for Maximal Independence Given Minimal Dependence. In: Proceedings of the First Annual ACM-SIAM Symposium on Discrete Algorithms, 22-24 January 1990, San Francisco, California, USA. pp. 212–218 (1990)
7. Benferhat, S., Bouraoui, Z., Croitoru, M., Papini, O., Tabia, K.: Non-Objection Inference for Inconsistency-Tolerant Query Answering. In: Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016. pp. 3684–3690 (2016)
8. Bienvenu, M.: On the Complexity of Consistent Query Answering in the Presence of Simple Ontologies. In: Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence, July 22-26, 2012, Toronto, Ontario, Canada. (2012), http://www.aaai.org/ocs/index.php/AAAI/AAAI12/paper/view/4928
9. Boros, E., Elbassioni, K.M., Gurvich, V., Khachiyan, L.: An Efficient Incremental Algorithm for Generating All Maximal Independent Sets in Hypergraphs of Bounded Dimension. Parallel Processing Letters **10**(4), 253–266 (2000)
10. Bourgaux, C.: Inconsistency Handling in Ontology-Mediated Query Answering. Ph.D. thesis, Université Paris-Saclay, Paris (Sep 2016), https://tel.archives-ouvertes.fr/tel-01378723
11. Calì, A., Gottlob, G., Lukasiewicz, T., Marnette, B., Pieris, A.: Datalog+/-: A Family of Logical Knowledge Representation and Query Languages for New Applications. In: Proceedings of the 25th Annual IEEE Symposium on Logic in Computer Science, LICS 2010, 11-14 July 2010, Edinburgh, United Kingdom. pp. 228–242 (2010). https://doi.org/10.1109/LICS.2010.27, https://doi.org/10.1109/LICS.2010.27
12. Croitoru, M., Vesic, S.: What Can Argumentation Do for Inconsistent Ontology Query Answering? In: Scalable Uncertainty Management - 7th International Conference, SUM 2013, Washington, DC, USA, September 16-18, 2013. Proceedings. pp. 15–29 (2013)
13. Dahlhaus, E., Karpinski, M., Kelsen, P.: An Efficient Parallel Algorithm for Computing a Maximal Independent Set in a Hypergraph of Dimension 3. Inf. Process. Lett. **42**(6), 309–313 (1992). https://doi.org/10.1016/0020-0190(92)90228-N, https://doi.org/10.1016/0020-0190(92)90228-N
14. Gallo, G., Longo, G., Pallottino, S.: Directed Hypergraphs and Applications. Discrete Applied Mathematics **42**(2), 177–201 (1993). https://doi.org/10.1016/0166-218X(93)90045-P, https://doi.org/10.1016/0166-218X(93)90045-P

15. Goldberg, M.K., Spencer, T.H.: A New Parallel Algorithm for the Maximal Independent Set Problem. SIAM J. Comput. **18**(2), 419–427 (1989). https://doi.org/10.1137/0218029, https://doi.org/10.1137/0218029
16. Grégoire, É., Mazure, B., Piette, C.: Boosting a Complete Technique to Find MSS and MUS Thanks to a Local Search Oracle. In: IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, January 6-12, 2007. pp. 2300–2305 (2007), http://ijcai.org/Proceedings/07/Papers/370.pdf
17. Grégoire, É., Mazure, B., Piette, C.: Using local search to find MSSes and MUSes. European Journal of Operational Research **199**(3), 640–646 (2009). https://doi.org/10.1016/j.ejor.2007.06.066, https://doi.org/10.1016/j.ejor.2007.06.066
18. Hecham, A.: Defeasible reasoning for existential rules. (Raisonnement defaisable dans les règles existentielles). Ph.D. thesis (2018), https://tel.archives-ouvertes.fr/tel-01904558
19. Hecham, A., Bisquert, P., Croitoru, M.: On the Chase for All Provenance Paths with Existential Rules. In: Rules and Reasoning - International Joint Conference, RuleML+RR 2017, London, UK, July 12-15, 2017, Proceedings. pp. 135–150 (2017)
20. Kelsen, P.: On the Parallel Complexity of Computing a Maximal Independent Set in a Hypergraph. In: Proceedings of the 24th Annual ACM Symposium on Theory of Computing, May 4-6, 1992, Victoria, British Columbia, Canada. pp. 339–350 (1992). https://doi.org/10.1145/129712.129745, https://doi.org/10.1145/129712.129745
21. Lembo, D., Lenzerini, M., Rosati, R., Ruzzi, M., Savo, D.F.: Inconsistency-Tolerant Semantics for Description Logics. In: Web Reasoning and Rule Systems - Fourth International Conference, RR 2010, Bressanone/Brixen, Italy, September 22-24, 2010. Proceedings. pp. 103–117 (2010)
22. Poggi, A., Lembo, D., Calvanese, D., Giacomo, G.D., Lenzerini, M., Rosati, R.: Linking Data to Ontologies. J. Data Semantics **10**, 133–173 (2008)
23. Rocher, S.: Interrogation tolérante aux incohérences. Tech. rep., Université de Montpellier (2013)
24. Tamani, N., Mosse, P., Croitoru, M., Buche, P., Guillard, V., Guillaume, C., Gontard, N.: Eco-Efficient Packaging Material Selection for Fresh Produce: Industrial Session. In: Hernandez, N., Jäschke, R., Croitoru, M. (eds.) Graph-Based Representation and Reasoning: 21st International Conference on Conceptual Structures, ICCS 2014, Iaşi, Romania, July 27-30, 2014, Proceedings, pp. 305–310. Springer International Publishing, Cham (2014)
25. Yun, B., Bisquert, P., Buche, P., Croitoru, M.: Arguing About End-of-Life of Packagings: Preferences to the Rescue. In: Metadata and Semantics Research - 10th International Conference, MTSR 2016, Göttingen, Germany, November 22-25, 2016, Proceedings. pp. 119–131 (2016)
26. Yun, B., Vesic, S., Croitoru, M., Bisquert, P.: Inconsistency Measures for Repair Semantics in OBDA. In: Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden. pp. 1977–1983 (2018). https://doi.org/10.24963/ijcai.2018/273, https://doi.org/10.24963/ijcai.2018/273