# Minimizing Side-effects in Virtual Knowledge Graph Updates

Romuald Esdras Wandji[1] [0009−0008−5036−2452] and
Diego Calvanese[2] [0000−0001−5174−9693]

[1] Department of Computing Science, Umeå Universitet, Umeå, Sweden
romuald.esdras.wandji@umu.se
[2] Faculty of Engineering, Free University of Bozen-Bolzano, Bolzano, Italy
diego.calvanese@unibz.it

**Abstract.** In Virtual Knowledge Graphs (VKGs), access to a relational data source is provided through an ontology that is linked to the data source via declarative mappings. While the problem of query answering in VKGs has been studied extensively over the past years, much less attention has been devoted to the problem of instance-level updates over the VKG, realized by updating the underlying data source. Due to the form of VKG mappings, translating VKG updates into a source updates might lead to side-effects in the VKG, i.e., unwanted insertions or deletions. In this paper, we build on a recent proposal for translating VKG updates into source updates, and extend it by introducing the notion of compensation, which are additional updates that aim at reverting side-effects. We provide a novel algorithm relying on multiple levels of compensation and show that it computes source updates with minimal side-effects in the VKG.

## 1 Introduction

The Virtual Knowledge Graph (VKG) approach (formerly known as ontology-based data Access – OBDA) [16,5,19,20] is a well-established paradigm for data access and integration, which has been investigated extensively, especially in the context where the data sources to be accessed or integrated are relational. In VKGs, an ontology encapsulates essential domain knowledge and is linked to the data source via declarative mappings, exposing to users a virtual knowledge graph. Users can issue high-level ontological queries over such VKG, and these are automatically translated, using both the ontology axioms and the mappings, into equivalent low-level queries (like SQL in a relational setting) that the underlying database engine can execute. The ontology is usually specified in a lightweight language, and in this paper we consider as ontology language *DL-Lite*$_R$, which can capture conceptual modeling formalisms and enjoys efficient reasoning. Most importantly, *DL-Lite*$_R$ based VKG systems enjoy *first-order rewritability*, i.e., any (union of) conjunctive queries issued over the ontology can be rewritten considering both the ontology's axioms and the declarative mappings into a SQL query that, when executed over the underlying relational data sources, computes the entailed answers [6,5].

The primary reasoning service offered by VKG systems is *query answering*, which is carried out through *query rewriting* and *query unfolding* [16,5]. However, little attention has been paid to the issue of updating VKGs, which allows one to take advantage

of the knowledge graph's capacity to handle incomplete information and provide support for update operations over the source data through the lens of the ontology. More specifically, we are interested in instance-level (or ABox) updates in VKGs, where updates are applied over the extensional level of the system (the VKG), and need to be translated into equivalent updates over the source. Such a feature will allow content owners to fully manage all information at the level of the ontology, and hence detach from low-level details of the underlying source structure and organization.

Several challenges need to be taken into account when updating a VKG through the ontology. One has to deal first with potential inconsistencies between the ontology and the provided update, and second, with the translation of the update to the underlying data source using the schema mappings. The first problem has been studied in the literature in the context of knowledge base update and belief revision [18,13,12,9], which also consider ontologies specified in DLs [22]. The second one is connected to the well-known and well-studied *view update* problem [3,8,11]. Hence, the typical steps in the VKG update framework are: *(1)* the user poses an instance-level update request $\mathcal{U}_A$ over the knowledge base (KB) $\mathcal{K} = \langle \mathcal{T}, \mathcal{M}(D) \rangle$ of a VKG instance. *(2)* The instance level (i.e., virtual ABox) $\mathcal{M}(D)$ of $\mathcal{K}$ is updated and possibly repaired w.r.t. the ontology $\mathcal{T}$ (according to a chosen repair semantics, see, e.g., [22]), which produces the actual update $\mathcal{U}'_A$ to be executed over $\mathcal{K}$. *(3)* A process called *translation* takes $\mathcal{U}'_A$ and $\mathcal{K}$ to produce a set $\mathbf{U}_D = \{\mathcal{U}^1_D, \ldots, \mathcal{U}^n_D\}$ of possible updates over the source database. *(4)* Finally, a translation $\mathcal{U}^i_D$ in $\mathbf{U}_D$ is chosen, for which the knowledge base $\mathcal{K}' = \langle \mathcal{T}, \mathcal{M}(D_i) \rangle$ is as close as possible to $\mathcal{K}$, where $D_i$ is the source database obtained by applying $\mathcal{U}^i_D$ to $D$.

Considering that there is already a vast literature on ontology update and corresponding semantics to handle possible inconsistencies, in this paper, we abstract from repair at the ontology level (Step 2), and concentrate instead on the translation of the ABox update into source updates (Step 3), and on the selection of the actual translation to adopt (Step 4). Therefore, we assume that the actual update that gets executed over the VKG instance already includes the update operations realizing the repair (hence, coincides with $\mathcal{U}'_A$). Then, the distance between $\mathcal{K}'$ and $\mathcal{K}$ is the *side-effect* that represents insertions or deletions in the VKG that are due to the way the mapping $\mathcal{M}$ propagates the source update to the ontology level. Recently, two methods have been proposed to translate ABox deletions and insertions into source deletions and insertions, respectively [17]. However, the assumption of considering translations that are of the same type as the corresponding ABox updates might lead to side-effects that are not minimal.

*Example 1.* As a motivating example, consider a data source with two relations: $RS(res, sup)$ that relates researchers to their supervisor(s), and $SG(sup, gr)$ that relates supervisors to the grants they have access to. The information in this source has to be integrated into a VKG whose ontology contains a role $access$, relating researchers to grants, and a class $supervises$ relating supervisors with their students. We consider a mapping $\mathcal{M}$ between this source schema and the ontology consisting of the following assertions:

$$\begin{aligned} \exists y.RS(x,y) \wedge SG(y,z) &\rightsquigarrow access(x,z); \\ RS(x,y) &\rightsquigarrow supervises(y,x). \end{aligned}$$

Let us assume that we have a database with a single tuple $D = \{SG(\texttt{sup1}, \texttt{grant1})\}$, and an update that consists of inserting the ABox fact $supervises(\texttt{sup1}, \texttt{john})$. Based on the mapping $\mathcal{M}$, the only translation of such update is the insertion of $RS(\texttt{john}, \texttt{sup1})$ in $D$, which will lead to an extra insertion of $access(\texttt{john}, \texttt{grant1})$. However, the translation that consists of updating the database by inserting $RS(\texttt{john}, \texttt{sup1})$ and deleting $SG(\texttt{sup1}, \texttt{grant1})$, is an exact translation in $D$ of the ABox update. We observe that combining both insertions and deletions in the source data has lead to an exact translation of a given ABox insertion. Notice also that deleting a database fact concerning grants to reflect an ABox insertion concerning supervision might seem counterintuitive. However, in the situation represented by our (admittedly very simple) example, where the only mapping assertions are the ones provided above, the specified database update, consisting of both a deletion of a tuple and an insertion of a tuple, exactly captures the intention of the user to only insert the ABox fact $supervises(\texttt{sup1}, \texttt{john})$. ◁

In this paper, we propose to translate each ABox update operation (consisting of insertions or deletions only) in general through a combination of both database insertions and deletions. The additional update operations are meant to compensate possible side-effects caused by database updates, and we formalize the notion of minimal side-effect in our enriched setting. We then propose methods to recursively compute maximal compensations so as to find translations with minimal side-effects, and in particular translations that are side-effect-free (*exact translation*) whenever an exact translation exists.

The rest of the paper is structured as follows. In Section 2, we provide the necessary technical preliminaries. In Section 3 we provide an algorithm for minimal (both in the source and the ABox) direct translations of ABox updates. In Section 4 we introduce the notion of maximal compensation, and in Section 5 we use it to compute updates with minimal side-effects. Finally, Section 6 concludes the paper.

## 2 Preliminaries

We now introduce the notions about description logics (DLs), databases, and VKGs necessary to understand the technical development in the paper, assuming familiarity with the syntax and semantics of first-order logic (FOL). In general, when convenient, we will view a tuple of elements (constants, variables, etc.) as equivalent to the set of such elements. E.g., if $\boldsymbol{c}$ is a tuple of constants and $D$ is a set, we might write $\boldsymbol{c} \subseteq D$ to mean that every element of $\boldsymbol{c}$ is in $D$. We consider countably infinite and pairwise disjoint alphabets $\mathbf{N}_C$ of *concept names*, $\mathbf{N}_P$ of *role names*, and $\mathbf{N}_I$ of constants.

*Description Logic Knowledge Bases.* A DL *knowledge base* (KB) $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ consists of a *TBox* $\mathcal{T}$ (also called *ontology*), capturing intensional information, and an *ABox* $\mathcal{A}$, providing extensional information. We consider DLs of the *DL-Lite* family [6,16], and specifically *DL-Lite*$_R$, which is the formal counterpart of the tractable OWL 2 QL profile of the Web Ontology Language (OWL 2) [15]. A *DL-Lite*$_R$ TBox is a finite set of assertions of the form $B_1 \sqsubseteq B_2$ (*concept inclusion*), $B_1 \sqsubseteq \neg B_2$ (*concept disjointness*), $R_1 \sqsubseteq R_2$ (*role inclusion*), or $R_1 \sqsubseteq \neg R_2$ (*role disjointness*). Here, $R$ (possibly subscripted) denotes an atomic role $P \in \mathbf{N}_P$ or its inverse $P^-$, while $B$ (possibly subscripted) denotes a *basic concept*, which is either an atomic concept $A \in \mathbf{N}_C$, or a

concept of the form $\exists R$. For a TBox $\mathcal{T}$, we use $\mathcal{T}^+$ to denote the set of concept and role inclusions in $\mathcal{T}$, and $\mathcal{T}^-$ to denote the set of concept and role disjointness assertions in $\mathcal{T}$, hence $\mathcal{T} = \mathcal{T}^+ \cup \mathcal{T}^-$. A *DL-Lite*$_R$ ABox is a finite set of assertions of the form $A(c)$ or $P(c, c')$, with $A \in \mathbf{N}_C$, $P \in \mathbf{N}_P$, and $c, c' \in \mathbf{N}_I$.

The semantics of a DL KB is given, as usual, in terms of first-order interpretations [2]. We make use of the standard notions of *model*, *consistency*, and *logical implication*.

*Instance Level Updates in DL KBs.* We consider updates to a DL KB $\langle \mathcal{T}, \mathcal{A} \rangle$ consisting of a pair $\mathcal{U}_A = \langle \mathcal{U}_A^-, \mathcal{U}_A^+ \rangle$, where $\mathcal{U}_A^-$ and $\mathcal{U}_A^+$ are sets of ABox facts, respectively to be deleted and inserted from the KB, such that $\mathcal{U}_A^- \subseteq \mathcal{A}$ and $\mathcal{U}_A^+ \cap \mathcal{A} = \emptyset$. In this paper, we are concerned with how VKG mappings affect updates, not how the TBox affects the consistency of the updates. Therefore, we follow [17] and consider that inserting $\mathcal{U}_A^+$ amounts to inserting $cl_\mathcal{T}(\mathcal{U}_A^+)$, which represents the *closure* of $\mathcal{U}_A^+$ w.r.t. $\mathcal{T}$, that is, the set of ABox assertions over individuals in $\mathcal{U}_A^+$ that are logically implied by $\langle \mathcal{T}, \mathcal{U}_A^+ \rangle$. Similarly, deleting $\mathcal{U}_A^-$ amounts to deleting $invcl_\mathcal{T}(\mathcal{U}_A^-)$, which represent the *inverse closure* of $\mathcal{U}_A^-$ [22]. Both $cl_\mathcal{T}(\mathcal{U}_A^+)$ and $invcl_\mathcal{T}(\mathcal{U}_A^-)$ can be computed in polynomial time in the size of $\mathcal{U}_A^+$, $\mathcal{U}_A^-$, and $\mathcal{T}$ [6,22]. We also assume that the update does not request to both insert and delete the same ABox fact, and since we prioritize data consistency, this means that $cl_\mathcal{T}(\mathcal{U}_A^+) \cap invcl_\mathcal{T}(\mathcal{U}_A^-) = \emptyset$.

*Relational Databases and Queries.* A *database schema* is a finite set $\mathcal{S} = \{r_1/n_1, \ldots, r_k/n_k\}$ of relation schemas, where each $r_i$ is a predicate name of arity $n_i$. A database instance $D$ over $\mathcal{S}$ maps each predicate $r/n$ in $\mathcal{S}$ to an $n$-ary relation, denoted $r^D$. An *atom* for $r/n$ has the form $r(t_1, \ldots, t_n)$, or simply $r(\boldsymbol{t})$, where each $t_j$ is a term, which can be a constant from $\mathbf{N}_I$ or a variable. If all $t_j$'s are constants, the atom is called ground, or simply a *tuple*.

A FOL formula over a relational schema $\mathcal{S}$ is constructed over the relation names in $\mathcal{S}$, the equality predicate $=$, and the constants in $\mathbf{N}_I$. A formula $\varphi$ with free variables $\boldsymbol{x}$ is denoted $\varphi(\boldsymbol{x})$, and is also called a (relational) *query* with *answer variables* $\boldsymbol{x}$. The formula is *closed* when $\boldsymbol{x}$ is empty. The closed formula obtained from $\varphi(\boldsymbol{x})$ by replacing the variables in $\boldsymbol{x}$ by the corresponding constants in $\boldsymbol{c}$ is denoted $\varphi(\boldsymbol{c})$.

A *conjunctive query* (CQ) $Q(\boldsymbol{x})$ over the schema $\mathcal{S}$ is a query defined by a formula of the form $\exists \boldsymbol{y}.\varphi(\boldsymbol{x}, \boldsymbol{y})$, where $\varphi = r_1(\boldsymbol{t}_1) \wedge \cdots \wedge r_n(\boldsymbol{t}_n)$ is a conjunction of atomic formulae whose variables belong to $\boldsymbol{x} \cup \boldsymbol{y}$. The variables $\boldsymbol{y}$ are called *existential variables*. We also express such CQ as a logical rule of the form $Q(\boldsymbol{x}) \leftarrow r_1(\boldsymbol{t}_1), \ldots, r_n(\boldsymbol{t}_n)$, where $Q(\boldsymbol{x})$ is called the *head* and $r_1(\boldsymbol{t}_1), \ldots, r_n(\boldsymbol{t}_n)$ the *body* of the rule. A *union of conjunctive queries* (UCQ) is a finite disjunction of CQs with the same answer variables, also represented as a finite set of rules with the same head.

### 2.1 Virtual Knowledge Graphs

We recall the main elements of the VKG framework, also known as *Ontology-based Data Access* (OBDA) [16,19], where declarative mappings are used to connect a TBox $\mathcal{T}$ to a relational data source with schema $\mathcal{S}$.

A *VKG mapping* $\mathcal{M}$ from $\mathcal{S}$ to $\mathcal{T}$ consists of a set of *mapping assertions* of the form $Q(\boldsymbol{x}) \rightsquigarrow E(\boldsymbol{t}(\boldsymbol{x}))$, where $Q(\boldsymbol{x})$ is a CQ (called *source query*) over $\mathcal{S}$ of arity $n > 0$, and $E(\boldsymbol{t}(\boldsymbol{x}))$ is an atom (called *target atom*) over $\mathcal{T}$ with variables in $\boldsymbol{x}$. Such atom has the form $A(t_1(\boldsymbol{x}_1))$, with $A \in \mathbf{N}_C$, or $P(t_1(\boldsymbol{x}_1), t_2(\boldsymbol{x}_2))$, with $P \in \mathbf{N}_P$, where the terms $t_1(\boldsymbol{x}_1)$ and $t_2(\boldsymbol{x}_2)$ denote so-called *IRI-templates*, obtained by applying (skolem) functions to the answer variables of the source query. Such IRI-templates[3] are used to generate strings representing object *IRIs* (Internationalized Resource Identifiers) or (RDF) literals, starting from database values retrieved by the source query in the mapping. In practice, it is desired that each IRI can be constructed by at most one IRI template, and we make such an assumption here; formally, the union of all the IRI templates is an *injective* function, i.e., there is a unique way to reconstruct from a string $s$, the actual IRI-template $t(\boldsymbol{x})$ and the constituent values $\boldsymbol{v}$ used to generate $s = t(\boldsymbol{v})$. Moreover we assume that for each IRI-template $f(x_1, \ldots, x_n)$, we have also $n$ *inverse templates*[4] $f^1, \ldots, f^n$ such that $f^i(f(v_1, \ldots, v_n)) = v_i$, for each $i \in [1..n]$ and for all possible values $v_1, \ldots, v_n$ instantiating the variables $x_1, \ldots, x_n$.

A VKG mapping $\mathcal{M}$ from $\mathcal{S}$ to $\mathcal{T}$, which we also call a *source-to-ontology mapping* (*so-mapping*), maps a database instance $D$ of $\mathcal{S}$ to the (*unique*) ABox

$$\mathcal{M}(D) = \{\mathsf{as}(E(\boldsymbol{t}(\boldsymbol{x})), \boldsymbol{x} \mapsto \boldsymbol{o}) \mid (\boldsymbol{x} \mapsto \boldsymbol{o}) \in Q(\boldsymbol{x})^D, (Q(\boldsymbol{x}) \rightsquigarrow E(\boldsymbol{t}(\boldsymbol{x}))) \in \mathcal{M}\}$$

where $\boldsymbol{x} \mapsto \boldsymbol{o}$ represents a solution mapping from the evaluation $Q(\boldsymbol{x})^D$ of the source query $Q(\boldsymbol{x})$ over the database instance $D$, and the term $\mathsf{as}(E(\boldsymbol{t}(\boldsymbol{x})), \boldsymbol{x} \mapsto \boldsymbol{o}))$ denotes the ABox assertion obtained by applying the solution mapping $\boldsymbol{x} \mapsto \boldsymbol{o}$ to $E(\boldsymbol{t}(\boldsymbol{x}))$. This application typically involves replacing $\boldsymbol{x}$ with $\boldsymbol{o}$ in the templates in $\boldsymbol{t}(\boldsymbol{x})$ (using appropriate string concatenation operations). It is also convenient to consider a so-mapping $\mathcal{M}$ as the set of pairs $\{(D, \mathcal{M}(D)) \mid D \text{ is a database instance of } \mathcal{S}\}$, so that $(D, \mathcal{A}) \in \mathcal{M}$ is an alternative notation for $\mathcal{A} = \mathcal{M}(D)$. Notice that, since a concept name $A$ (or role name $P$) might appear as target atom of multiple mapping assertions, the source query that generates the instances of $A$ (or $P$) is in general a UCQ.

We are now ready to formalize the VKG setting. A *VKG specification* is a tuple $\mathcal{P} = \langle \mathcal{T}, \mathcal{M}, \mathcal{S} \rangle$, where $\mathcal{T}$ is a *DL-Lite*$_R$ TBox (typically expressed in OWL 2 QL), $\mathcal{S}$ is a data source schema, and $\mathcal{M}$ is a set of so-mappings from $\mathcal{S}$ to $\mathcal{T}$. A *VKG instance* is a pair $\langle \mathcal{P}, D \rangle$ where $D$ is a source instance of $\mathcal{S}$. Its semantics is defined in terms of FO interpretations over $\mathcal{T}$. An interpretation $\mathcal{I}$ is a *model* of $\langle \mathcal{P}, D \rangle$ if it is a model of the DL KB $\langle \mathcal{T}, \mathcal{M}(D) \rangle$.

We recall also the notion of *mapping saturation* [14], which allows one to compile into the mapping $\mathcal{M}$ the inclusion assertions of $\mathcal{T}$ that do *not* involve an existential quantification $\exists R$ in the right-hand side. Specifically, to compute the *mapping saturation* $sat_{\mathcal{T}}(\mathcal{M})$ *of* $\mathcal{M}$ *w.r.t.* $\mathcal{T}$, we start from $\mathcal{M}$, and repeatedly add implied mapping assertions. E.g., if $Q(\boldsymbol{x}) \rightsquigarrow A_1(t(\boldsymbol{x}))$ is in $sat_{\mathcal{T}}(\mathcal{M})$ and $A_1 \sqsubseteq A_2$ is in $\mathcal{T}$, we add to $sat_{\mathcal{T}}(\mathcal{M})$ also $Q(\boldsymbol{x}) \rightsquigarrow A_2(t(\boldsymbol{x}))$; similarly for inclusions $\exists R \sqsubseteq A$ and $R_1 \sqsubseteq R_2$.

As shown in [17], when dealing with ABox updates and their side-effects in the VKG setting, we can assume w.l.o.g. that $\langle \mathcal{P}, D \rangle$ is a VKG instance, where $\mathcal{P} = \langle \mathcal{T}^-, \mathcal{M}, \mathcal{S} \rangle$ is a VKG specification whose mapping $\mathcal{M}$ has been saturated w.r.t. an

[3] IRI-templates correspond to the R2RML *string templates*.

[4] These correspond to the `rr:inverseExpression` of R2RML.

original TBox $\mathcal{T}$, and where $\mathcal{T}^{-}$ consists of the disjointness assertions of $\mathcal{T}$. In the following, we make such an assumption.

### 2.2 Lineage and Schema Mapping Recovery

We adopt some definitions by [17]. For a given assertion in the ABox, its lineage (also called *provenance*) consists of the minimal set of source tuples that generate the assertion through the VKG mapping. Formally:

**Definition 1 (Lineage).** Let $\mathcal{P} = \langle \mathcal{T}, \mathcal{M}, \mathcal{S} \rangle$ be a VKG specification, $\langle \mathcal{P}, D \rangle$ a VKG instance, and $f \in \mathcal{M}(D)$ an ABox assertion. A subset $B \subseteq D$ is a *lineage branch of* $f$ if $f \in \mathcal{M}(B)$, and for every $B' \subsetneq B$, $f \notin \mathcal{M}(B')$. The *lineage of* $f$, denoted $\mathsf{lineage}(f, \mathcal{P}, D)$, is the set of all lineage branches of $f$. ◁

For an so-mapping $\mathcal{M}$, a *reverse mapping* describes a novel mapping now going from $\mathcal{T}$ back to $\mathcal{S}$, called *ontology-to-source mapping* (*os-mapping*). We consider such a mapping $\widehat{\mathcal{M}}$ as a set of pairs $(\mathcal{A}, D)$, with $\mathcal{A}$ an ABox for $\mathcal{T}$ and $D$ a database instance of $\mathcal{S}$, called *solution* of $\mathcal{A}$ under $\widehat{\mathcal{M}}$. Moreover, we define $\widehat{\mathcal{M}}(\mathcal{A}) = \{D \mid (\mathcal{A}, D) \in \widehat{\mathcal{M}}\}$. We are interested in os-mappings that maintain semantic consistency with the relationship established by $\mathcal{M}$, and which intuitively represent the inverse of such relationship, in line with the notion of *inverse mapping* for the relational setting [10].

Notice that an so-mapping $\mathcal{M}$ is a function, since it generates from a database instance $D$ a unique ABox $\mathcal{M}(D)$. Instead, due to the non-injectiveness of so-mappings, its reverse $\widehat{\mathcal{M}}$ in general is not a function, but a relation that maps an ABox to a set of database instances. In order to construct the reverse of an so-mapping $\mathcal{M}$, we rely on the notion of *MR-os-mapping* as introduced by [17], which is based on the *maximum recovery* for schema mappings introduced in the database exchange setting [1], but adapt it in Section 3 to the more general case we consider here.

## 3 Update Translation in VKGs

In VKGs, the central challenge in dealing with ABox updates, lies in translating them into appropriate source updates. In [17], two algorithms are proposed that respectively handle translations of only ABox deletions and only ABox insertions. Also in the solution proposed there, the fundamental assumption is made that ABox deletions are translated into a set of source deletions only, and similarly for ABox insertions. As discussed in the introduction, such kinds of translations might not produce translations of ABox updates with minimal side-effects.

In this paper, we overcome the limitations of the previous work and extend our former translation framework along two directions: *(i)* We lift the restriction on the form of ABox translation: A set of ABox deletions (or of ABox insertions) could now be translated to a combination of both source deletions and source insertions, with the aim of obtaining the minimum side-effects. *(ii)* We consider also update operations that consist of a combination of deletions and insertions. *(iii)* We allow the use of different IRI-templates for the same predicate in the targets of mappings.

We now extend the notion of update translation in VKGs as introduced in [17]. Moreover, we introduce the notion of direct translation, which refers to translations that are of the same nature as the original ABox update, i.e., ABox insertions translate to source insertions, and ABox deletions translate to source deletions.

**Definition 2 (Direct deletion translation).** Let $\mathcal{U}_A^-$ be a set of ABox deletions. A set $\mathcal{U}_D^-$ of source deletions is a *direct translation of* $\mathcal{U}_A^-$ *in* $\mathcal{J}$ if $\mathcal{U}_A^- \cap \mathcal{M}(D \setminus \mathcal{U}_D^-) = \emptyset$. We say that $\mathcal{U}_D^-$ is an *exact direct translation of* $\mathcal{U}_A^-$ *in* $\mathcal{J}$ if $\mathcal{M}(D \setminus \mathcal{U}_D^-) = \mathcal{M}(D) \setminus \mathcal{U}_A^-$.◁

An exact direct translation of the ABox deletion $\mathcal{U}_A^-$ ensures that requested deletions are perfectly reflected in the final (virtual) ABox generated from the data source via the mapping $\mathcal{M}$. A non-exact translation may inadvertently delete additional ABox facts not initially specified in $\mathcal{U}_A^-$. Following, we call the set of these facts that are unintendedly deleted from the ABox a *side-effect*. We can proceed in a symmetric way for ABox insertions.

**Definition 3 (Direct insertion translation).** Let $\mathcal{U}_A^+$ be a set of ABox insertions. A set $\mathcal{U}_D^+$ of source insertions is a *direct translation of* $\mathcal{U}_A^+$ *in* $\mathcal{J}$ if $\mathcal{U}_A^+ \subseteq \mathcal{M}(\mathcal{U}_D^+)$ and $\langle \mathcal{T}^-, \mathcal{M}(D \cup \mathcal{U}_D^+) \rangle$ is consistent. $\mathcal{U}_D^+$ is an *exact direct translation of* $\mathcal{U}_A^+$ *in* $\mathcal{J}$ if $\mathcal{M}(D \cup \mathcal{U}_D^+) = \mathcal{M}(D) \cup \mathcal{U}_A^+$. ◁

In the definition, we require that $\mathcal{U}_A^+ \subseteq \mathcal{M}(\mathcal{U}_D^+)$ to ensure that the translation $\mathcal{U}_D^+$ alone is enough to generate $\mathcal{U}_A^+$ through $\mathcal{M}$. However, notice that, by complying with the minimal change principle, $\mathcal{U}_D^+$ might contain tuples already present in $D$ and therefore will result in fewer tuples inserted over $D$ when $\mathcal{U}_D^+$ is added to $D$.

Hence, an exact translation of the ABox insertion $\mathcal{U}_A^+$ will insert only the requested facts into the (virtual) ABox. In contrast, a non-exact translation could lead to the insertion of additional facts beyond $\mathcal{U}_A^+$. The side-effect is the set of these facts that are unintendedly inserted into the ABox. If the side-effect causes an inconsistency with the disjointness assertions in $\mathcal{T}^-$, then the (non-exact) translation $\mathcal{U}_D^+$ is ruled out to maintain data integrity in the VKG instance. Notice also that a (direct) insertion translation might not exist, either because some facts in $\mathcal{U}_A^+$ cannot be obtained via the mapping (because the set of lineage branches is empty), or because the side-effects would necessarily violate the disjointness assertions in $\mathcal{T}^-$.

To comply with the minimal change principle in update translation, it is essential to ensure that a change in the ABox results in a minimal change in the source data. Building on this principle, we propose extending the definition of translation for ABox updates so as to ensure that the translation of ABox updates remains minimal in the source data, characterized by the following definition.

**Definition 4 (Minimal direct translation).** Let $\mathcal{U}_A^-$ be a set of ABox deletions. A direct translation $\mathcal{U}_D^-$ of $\mathcal{U}_A^-$ in $\mathcal{J}$ is *minimal* if every proper subset of $\mathcal{U}_D^-$ is not a translation of $\mathcal{U}_A^-$ in $\mathcal{J}$. Similarly, let $\mathcal{U}_A^+$ be a set of ABox insertions. A direct translation $\mathcal{U}_D^+$ of $\mathcal{U}_A^+$ in $\mathcal{J}$ is *minimal* if every proper subset of $\mathcal{U}_D^+$ is not a translation of $\mathcal{U}_A^+$ in $\mathcal{J}$. ◁

Alg. 1 (MINTRANSDEL) extends the algorithm presented in [17] so that it returns all minimal direct translations of a set of ABox deletions. It makes use of the function $\mathsf{lineage}(f, \mathcal{P}, D)$, defined in [17], which computes the set of lineage branches in

**Algorithm 1:** MINTRANSDEL $(\mathcal{J}, \mathcal{U}_A^-)$

**input :** A VKG instance $\mathcal{J} = \langle \mathcal{P}, D \rangle$. A set $\mathcal{U}_A^-$ of ABox deletions.
**output:** A set of source direct translations of $\mathcal{U}_A^-$.

$\mathbf{B} = \{B_1, \ldots, B_n\} \leftarrow \bigcup_{f \in \mathcal{U}_A^-} \mathsf{lineage}(f, \mathcal{P}, D)$; *// Compute lineage branches in $D$*
$\mathbf{S} \leftarrow \{\{t_1, \ldots, t_n\} \mid t_i \in B_i, \text{for } i \in [1..n]\}$;
$\mathbf{U}_D^- \leftarrow \{\}$;
**for each** translation $T \in \mathbf{S}$ **do**
  $add \leftarrow true$;
  **for each** translation $T^\star \in \mathbf{U}_D^-$ **do**
    **if** $T^\star \subsetneq T$ **or** $\text{COMPARE}_{\mathcal{J}}^-(T^\star, T) =$ '<' **then** $add \leftarrow false$;
    **else if** $T \subsetneq T^\star$ **or** $\text{COMPARE}_{\mathcal{J}}^-(T, T^\star) =$ '<' **then** $\mathbf{U}_D^- \leftarrow \mathbf{U}_D^- \setminus \{T^\star\}$;
  **if** $add$ **then** $\mathbf{U}_D^- \leftarrow \mathbf{U}_D^- \cup \{T\}$
**return** $\mathbf{U}_D^-$

**Algorithm 2:** FILTERBRANCHES $(\mathbf{S})$

**input:** A set $\mathbf{S}$ of branches, each of the form $\bigcup_{i=1}^n \{\exists \boldsymbol{w}_i.\varphi_i(\boldsymbol{a}_i, \boldsymbol{w}_i)\}$.
**output:** A set $\mathbf{B}^*$ of CQs, each representing a minimal update.

$\mathbf{B} \leftarrow \{\bigwedge_{i=1}^n \exists \boldsymbol{w}_i.\varphi_i(\boldsymbol{a}_i, \boldsymbol{w}_i) \mid \bigcup_{i=1}^n \{\exists \boldsymbol{w}_i.\varphi_i(\boldsymbol{a}_i, \boldsymbol{w}_i)\} \in \mathbf{S}\}$
$\mathbf{B}^* \leftarrow \{\}$
**for each** branch $B \in \mathbf{B}$ **do**
  $add \leftarrow true$;
  **for each** branch $B^* \in \mathbf{B}^*$ **do**
    **if** $B \sqsubseteq B^*$ **then** $add \leftarrow false$;
    **else if** $B^* \sqsubseteq B$ **then** $\mathbf{B}^* \leftarrow \mathbf{B}^* \setminus \{B\}$;
  **if** $add$ **then** $\mathbf{B}^* \leftarrow \mathbf{B} \cup \{B\}$;
**return** $\mathbf{B}^*$

database $D$ for an ABox assertion $f$. It also exploits an abstract function $\text{COMPARE}_{\mathcal{J}}^-$ that compares two translations $T_1$ and $T_2$ of ABox deletions in terms of their side-effect and returns '<', if $T_1$ has less side-effect than $T_2$ based on the considered metric.

**Theorem 5.** *Let $\mathcal{U}_A^-$ be a set of ABox deletions. Then* MINTRANSDEL$(\mathcal{J}, \mathcal{U}_A^-)$ *returns every minimal source direct translation of $\mathcal{U}_A^-$ in $\mathcal{J}$.*

For the minimal direct translation of ABox insertions, we make use of some notation and definitions taken from [17], but adjust them to the more general setting adopted here. First, we assume w.l.o.g. that mapping assertions are *normalized* in the following sense: the head of a mapping assertion contains only variables of the form $x_i$ for $i \in \mathbb{N}$, no variable appears more than once, and the indexes $i$ of variables $x_i$ increase from left to right. Hence, in any $n$-ary tuple $\boldsymbol{t}(\boldsymbol{x}) = (t_1(\boldsymbol{x}_1), \ldots, t_n(\boldsymbol{x}_n))$, where $\boldsymbol{x}_i \subseteq \boldsymbol{x}$, of IRI-templates appearing in the head of a mapping assertion, $t_i(\boldsymbol{x}_i) \neq t_j(\boldsymbol{x}_j)$ when $i \neq j$, and we define $\boldsymbol{u}_{\boldsymbol{t}(\boldsymbol{x})}$ as an $n$-tuple $(u_1, \ldots, u_n)$ of pairwise distinct variables. We then define $V_{\boldsymbol{t}(\boldsymbol{x})} = V^1 \wedge \cdots \wedge V^n$, where each $V^i$ is obtained as follows: *(a)* if $t_i(\boldsymbol{x}_i)$

**Algorithm 3:** MINTRANSINS $(\mathcal{J}, \mathcal{U}_A^+)$

**input:** A VKG instance $\mathcal{J} = \langle\langle \mathcal{T}, \mathcal{M}, \mathcal{S}\rangle, D\rangle$.
A set $\mathcal{U}_A^+$ of ABox insertions.
**output:** A set of source direct translations of $\mathcal{U}_A^+$.

1 Compute the MR-os-mapping $\widehat{\mathcal{M}}^{mr}$ of $\mathcal{M}$;
2 Compute the insertion tree $\mathbf{B}$ of $\mathcal{U}_A^+$ w.r.t. $\widehat{\mathcal{M}}^{mr}$;
3 $\mathbf{U}_D^+ \leftarrow \{\}$;
4 **for each** branch $B \in$ FILTERBRANCHES$(\mathbf{B})$ of the form $\exists \boldsymbol{w}.\varphi(\boldsymbol{a}, \boldsymbol{w})$ **do**
5 $\quad \Delta_{\boldsymbol{w}} \leftarrow \{b_1, \ldots, b_{|\boldsymbol{w}|}\} \cup \Delta_A \cup \Delta_D$, where $\Delta_A$ and $\Delta_D$ are the constants in $\mathcal{U}_A^+$ and $D$ respectively, and each $b_i$ is a fresh value from $\mathbf{N}_I$ not in $\Delta_A \cup \Delta_D$;
6 $\quad$ **for each** assignment $\eta(\boldsymbol{w}) \subseteq \Delta_{\boldsymbol{w}}$ **do**
7 $\quad\quad T \leftarrow \gamma(\varphi(\boldsymbol{a}, \eta(\boldsymbol{w})))$;
8 $\quad\quad add \leftarrow true$;
9 $\quad\quad$ **for each** translation $T^\star \in \mathbf{U}_D^+$ **do**
10 $\quad\quad\quad$ **if** $(T^* \setminus D) \subsetneq (T \setminus D)$ **or** COMPARE$^+_{\mathcal{J}}(T^\star, T) =$ '<' **then** $add \leftarrow false$;
11 $\quad\quad\quad$ **else if** $(T \setminus D) \subsetneq (T^* \setminus D)$ **or** COMPARE$^+_{\mathcal{J}}(T, T^\star) =$ '<' **then** $\mathbf{U}_D^+ \leftarrow \mathbf{U}_D^+ \setminus \{T^\star\}$;
12 $\quad\quad$ **if** $add$ **then** $\mathbf{U}_D^+ \leftarrow \mathbf{U}_D^+ \cup \{T\}$
13 **return** $\mathbf{U}_D^+$

is a variable $y$ in $\boldsymbol{x}$, then $V^i$ is the equality $y = u_i$, *(b)* otherwise, if $t_i(\boldsymbol{x}_i)$ is an IRI-template $f(y_1, \ldots, y_k)$, where each $y_j$ is a variable in $\boldsymbol{x}$, then $V^i$ is $(y_1{=}f^1(u_i)) \wedge \cdots \wedge (y_k{=}f^k(u_i))$, where each $f^j$ is the $j$-th inverse template of $f$. Finally, for a concept or role $F$ in the target of a mapping $\mathcal{M}$, let $\mathbf{IRI}_{\mathcal{M}}(F) = \{\boldsymbol{t}(\boldsymbol{x}) \mid F(\boldsymbol{t}(\boldsymbol{x}))$ is the target of some mapping assertion in $\mathcal{M}\}$.

**Definition 6 (MR-os-mapping).** For a concept or role $F$ in the target of $\mathcal{M}$, Let $\mathcal{M}_{F(\boldsymbol{t}(\boldsymbol{x}))} = \bigcup_{\ell=1}^{k} \{\exists \boldsymbol{w}_\ell.\varphi_\ell(\boldsymbol{x}, \boldsymbol{w}_\ell) \rightsquigarrow F(\boldsymbol{t}(\boldsymbol{x}))\}$ be the set of all mapping assertions of $\mathcal{M}$ with $F(\boldsymbol{t}(\boldsymbol{x}))$ in the target. Then, we define the so-mapping $\widehat{\mathcal{M}}^{mr}_{F(\boldsymbol{t}(\boldsymbol{x}))} = \bigcup_{\ell=1}^{k} \{F(\boldsymbol{u}_{\boldsymbol{t}(\boldsymbol{x})}) \rightsquigarrow \exists \boldsymbol{w}_\ell.\varphi_\ell(\boldsymbol{x}, \boldsymbol{w}_\ell) \wedge V_{\boldsymbol{t}(\boldsymbol{x})}\}$. Finally, we call $\widehat{\mathcal{M}}^{mr} = \bigcup_{F \in \mathcal{T}} \bigcup_{\boldsymbol{t}(\boldsymbol{x}) \in \mathbf{IRI}_{\mathcal{M}}(F)} \widehat{\mathcal{M}}^{mr}_{F(\boldsymbol{t}(\boldsymbol{x}))}$ the *MR-os-mapping of* $\mathcal{M}$. ◁

The *insertion tree* of $\mathcal{U}_A^+$ is defined as the set of all *insertion branches* of $\mathcal{U}_A^+$, where each insertion branch represents a possible rewriting of $\mathcal{U}_A^+$ in the source schema. For convenience, we also make use of $\gamma$, defined as $\gamma(\bigwedge_{i=1}^{n} r_i(\boldsymbol{a_i})) = \bigcup_{i=1}^{n} \{r_i(\boldsymbol{a_i})\}$, to transform a conjunction of facts into a set of tuples.

As an extension to the method proposed in [17], the insertion algorithm MINTRANSINS (Alg. 3) consists of two stages. In the first stage, it first computes (at Line 2) the insertion tree with respect to the MR-os-mapping $\widehat{\mathcal{M}}^{mr}$, and then filters out redundant branches from the insertion tree using FILTERBRANCHES (Alg. 2). Since each branch is a CQ, the latter algorithm removes a branch $B$ if (as a CQ) it is contained in another branch $B^*$. Indeed, if $B \sqsubseteq B^*$, every atom of $B^*$ can be homomorphically mapped to an atom of $B$ [7], hence $B^*$ will result in fewer facts that need to be inserted

in the database, if chosen instead of $B$ as a branch for the insertion. Hence, branch $B$ can be removed. The second stage is analogous to the one of [17], but uses the branches returned by FILTERBRANCHES to find the translations to insert in the database that give rise to minimal side-effects. Notice that, when comparing translations for containment (Lines 10 and 11), we do so considering only the tuples that are not already in $D$.

**Theorem 7.** *Let $\mathcal{U}_A^+$ be a set of ABox insertions. Then* MINTRANSINS$(\mathcal{J}, \mathcal{U}_A^+)$ *returns every minimal source direct translation of $\mathcal{U}_A^+$ in $\mathcal{J}$.*

## 4 Compensations and their Maxima

As noticed, given a VKG instance and an ABox update $\mathcal{U}_A$, if there is no exact direct translation $\mathcal{U}_D$ (with no side-effect) of $\mathcal{U}_A$, we can attempt to find an extra update operation that can be carried out in the data source to minimize the side-effect in the ABox caused by the update in the data. We call such an extra operation a *compensation of the side-effect*. In the rest of the paper, we assume that an ABox update has the form $\mathcal{U}_A = \langle \mathcal{U}_A^-, \mathcal{U}_A^+ \rangle$, such that $\mathcal{U}_A^- \cap \mathcal{U}_A^+ = \emptyset$[5], and similarly we consider as corresponding source updates $\mathcal{U}_D = \langle \mathcal{U}_D^-, \mathcal{U}_D^+ \rangle$ where $\mathcal{U}_D^- \cap \mathcal{U}_D^+ = \emptyset$. Coherently, we extend the notion of side-effect to handle such combined updates. In what follows, we denote $D \cup \mathcal{U}_D^+ \setminus \mathcal{U}_D^-$ by $D \bullet \mathcal{U}_D$. Similarly, for $\mathcal{A} \bullet \mathcal{U}_A$.

**Definition 8 (Combined side-effect).** Let $\mathcal{U}_A = \langle \mathcal{U}_A^-, \mathcal{U}_A^+ \rangle$ be an ABox update, and $\mathcal{U}_D = \langle \mathcal{U}_D^-, \mathcal{U}_D^+ \rangle$ a source update. The *combined side-effect $E_{\mathcal{J}}(\mathcal{U}_A, \mathcal{U}_D)$ caused by $\mathcal{U}_D$ for $\mathcal{U}_A$* is $\langle E^-, E^+ \rangle$, where $E^- = (\mathcal{M}(D) \bullet \mathcal{U}_A) \setminus \mathcal{M}(D \bullet \mathcal{U}_D)$ is called the *deletion side-effect* and $E^+ = \mathcal{M}(D \bullet \mathcal{U}_D) \setminus (\mathcal{M}(D) \bullet \mathcal{U}_A)$ is called the *insertion side-effect*. ◁

Intuitively, the deletion side-effect $E^-$ represents tuples that should remain in the ABox but have been deleted and therefore should be added back. Symmetrically, the insertion side-effect $E^+$ represents unwanted tuples that have been inserted and that therefore should be removed. Note that none, one, or both of $E^-$ and $E^+$ in the combined side-effect might be empty.

For a given ABox update $\mathcal{U}_A$ and its corresponding source update $\mathcal{U}_D$ with deletion side-effect $E^-$, we would now like to find extra source insertions that minimize $E^-$.

**Definition 9 (Compensation by insertion).** A non-empty set $\Theta^+$ is a *compensation by insertion* of $E_{\mathcal{J}}(\mathcal{U}_A, \mathcal{U}_D)$ if there are two disjoint sets $E_\Delta^-$ and $E_\Delta^+$ of ABox assertions such that $\mathcal{U}_A^- \cap E_\Delta^+ = \emptyset$, $E_\Delta^- \neq \emptyset$, $E_\Delta^- \subseteq E^-$, and $E_{\mathcal{J}}(\mathcal{U}_A, \langle \mathcal{U}_D^-, \mathcal{U}_D^+ \cup \Theta^+ \rangle) = \langle E^- \setminus E_\Delta^-, E^+ \cup E_\Delta^+ \rangle$. ◁

In the compensation by insertion, the aim is to minimize the deletion side-effect $E^-$ caused by the deletion of $\mathcal{U}_D^-$ in the data source. And this is achieved when a non-empty subset $E_\Delta^-$ of $E^-$ is re-inserted when the compensation is applied in the source. However, attempting to minimize $E^-$ might also lead to extra insertions $E_\Delta^+$ in the ABox. To maintain the initially requested deletion $\mathcal{U}_A^-$ unaffected by the compensation, we require that $\mathcal{U}_A^- \cap E_\Delta^+ = \emptyset$.

[5] Since it suffices to consider TBoxes that consists of disjointness assertions only, we have that $\mathcal{U}_A^- = \mathit{invcl}(\mathcal{U}_A^-)$, and $\mathcal{U}_A^+ = \mathit{cl}(\mathcal{U}_A^+)$.

Similar to compensating deletion side-effects by extra source insertions, we can compensate insertion side-effects by extra source deletions.

**Definition 10 (Compensation by deletion).** A non-empty set $\Theta^- \subseteq D$ is a *compensation by deletion* of $E_{\mathcal{J}}(\mathcal{U}_A, \mathcal{U}_D)$ if there are two disjoint sets $E^-_\Delta$ and $E^+_\Delta$ of ABox assertions such that $\mathcal{U}^+_A \cap E^-_\Delta = \emptyset$, $E^+_\Delta \neq \emptyset$, $E^+_\Delta \subseteq E^+$, and $E_{\mathcal{J}}(\mathcal{U}_A, \langle \mathcal{U}^-_D \cup \Theta^-, \mathcal{U}^+_D \rangle) = \langle E^- \cup E^-_\Delta, E^+ \setminus E^+_\Delta \rangle$. ◁

The aim of compensating by deletion is to minimize the insertion side-effect $E^+$ caused by the insertion of $\mathcal{U}^+_D$ in the data source. And this is achieved when a non-empty subset $E^+_\Delta$ is removed from $E^+$ when the compensation is applied in the source. However, attempting to minimize $E^+$ might lead to deletions $E^-_\Delta$ in the ABox in addition to $E^-_\Delta$. Similarly to the case of compensation by insertion, to maintain the initially requested insertion $\mathcal{U}^+_A$ unaffected by the compensation, we require that $\mathcal{U}^+_A \cap E^-_\Delta = \emptyset$.

*Example 2.* Consider the VKG instance $\mathcal{J} = \langle \mathcal{P}_1, D \rangle$ of our motivating example, where $\mathcal{P}_1 = \langle \emptyset, \mathcal{M}_1, \mathcal{S}_1 \rangle$, and $D$ in this case consists of the following tuples $D = \{SG(\texttt{sup1}, \texttt{grant1}); SG(\texttt{sup1}, \texttt{grant2})\}$. Then, given $\mathcal{M}_1$, the ABox insertion $\mathcal{U}^+_A = \{supervises(\texttt{sup1}, \texttt{john})\}$ and a corresponding source insertion $\mathcal{U}^+_D = \{RS(\texttt{john}, \texttt{sup1})\}$, we obtain the side-effect $E_{\mathcal{J}}(\langle \emptyset, \mathcal{U}^+_A \rangle, \mathcal{U}_D) = \langle \emptyset, \{access(\texttt{john}, \texttt{grant1}); access(\texttt{john}, \texttt{grant2})\} \rangle$. A possible compensation by deletion might consists of an extra source deletion $\Theta^-_1 = \{SG(\texttt{sup1}, \texttt{grant1})\}$. Therefore, we obtain $E_{\mathcal{J}}(\langle \emptyset, \mathcal{U}^+_A \rangle, \langle \Theta^-_1, \mathcal{U}^+_D \rangle) = \langle \emptyset, \{access(\texttt{john}, \texttt{grant2})\} \rangle\}$. ◁

Our formalization of compensation in VKGs relates to any possible operation that can be carried out in the source to bring the resulting ABox closer to what the end user expects after a given ABox update operation. Given the nature of VKG mappings, different compensations can be applied for a given ABox update and its source translation, which leads to the need of comparing compensations.

In Example 2, if there is another compensation by deletion $\Theta^-_2$ such that $E_{\mathcal{J}}(\langle \emptyset, \mathcal{U}^+_A \rangle, \langle \mathcal{U}^-_D \cup \Theta^-_2, \mathcal{U}^+_D \rangle) = \langle \emptyset, \emptyset \rangle$, then one would prefer $\Theta^-_2$ over $\Theta^-_1$. In general, if $\Theta^-$ is a compensation by deletion of $E_{\mathcal{J}}(\mathcal{U}_A, \mathcal{U}_D)$, then the smaller the side-effect in $E_{\mathcal{J}}(\mathcal{U}_A, \langle \mathcal{U}^-_D \cup \Theta^-, \mathcal{U}^+_D \rangle)$ due to undesired insertions, the better $\Theta^-$ is. When the insertion side-effects are equal, we also consider the deletion side-effects.

**Definition 11.** Let $\Theta^-_1$, $\Theta^-_2$ be compensations by deletion of $E_{\mathcal{J}}(\mathcal{U}_A, \mathcal{U}_D)$ s.t. $E_{\mathcal{J}}(\mathcal{U}_A, \langle \mathcal{U}^-_D \cup \Theta^-_1, \mathcal{U}^+_D \rangle = \langle E^- \cup E^-_1, E^+ \setminus E^+_1 \rangle$ and $E_{\mathcal{J}}(\mathcal{U}_A, \langle \mathcal{U}^-_D \cup \Theta^-_2, \mathcal{U}^+_D \rangle = \langle E^- \cup E^-_2, E^+ \setminus E^+_2 \rangle$. We say that $\Theta^-_1$ is *better than* $\Theta^-_2$ for $E_{\mathcal{J}}$, and write $\Theta^-_2 \prec_{\mathcal{J}} \Theta^-_1$, if $E^+_2 \subsetneq E^+_1$ or we have that both $E^+_2 = E^+_1$ and $E^-_1 \subsetneq E^-_2$. ◁

Similarly, we can define an order between compensations by insertion. This means that if $\Theta^+$ is a compensation by insertion of $E_{\mathcal{J}}(\mathcal{U}_A, \mathcal{U}_D)$, then the smaller the side-effect in $E_{\mathcal{J}}(\mathcal{U}_A, \langle \mathcal{U}^-_D, \mathcal{U}^+_D \cup \Theta^+ \rangle)$ due to undesired deletions, the better $\Theta^+$ is. And if the deletion side-effects are equal, we also consider the insertion side-effects.

**Definition 12.** Let $\Theta^+_1, \Theta^+_2$ be compensations by insertion of $E_{\mathcal{J}}(\mathcal{U}_A, \mathcal{U}_D)$ s.t. $E_{\mathcal{J}}(\mathcal{U}_A, \langle \mathcal{U}^-_D, \mathcal{U}^+_D \cup \Theta^+_1 \rangle = \langle E^- \setminus E^-_1, E^+ \cup E^+_1 \rangle$ and $E_{\mathcal{J}}(\mathcal{U}_A, \langle \mathcal{U}^-_D, \mathcal{U}^+_D \cup \Theta^+_2 \rangle = \langle E^- \setminus E^-_2, E^+ \cup E^+_2 \rangle$. We say that $\Theta^+_1$ is *better than* $\Theta^+_2$ for $E_{\mathcal{J}}$, and write $\Theta^+_2 \prec_{\mathcal{J}} \Theta^+_1$ if $E^-_2 \subsetneq E^-_1$ or we have that both $E^-_2 = E^-_1$ and $E^+_1 \subsetneq E^+_2$. ◁

If, for a given ABox update $\mathcal{U}_A$ and its source translation $\mathcal{U}_D$, there exists a compensation by deletion $\Theta^-$ such that there is no other compensation of $E_{\mathcal{J}}(\mathcal{U}_A, \mathcal{U}_D)$ that is better, then we say that $\Theta^-$ is a *maximal compensation by deletion* that can be executed over the source data to minimize the insertion side-effects. Analogously, we can define a *maximal compensation by insertion* $\Theta^+$ that can be executed over the source data to minimize the deletion side-effects.

**Definition 13.** A set $\Theta_1^-$ is a *maximal compensation by deletion* of $E_{\mathcal{J}}(\mathcal{U}_A, \mathcal{U}_D)$ if there is no compensation by deletion $\Theta_2^-$ of $E_{\mathcal{J}}(\mathcal{U}_A, \mathcal{U}_D)$, such that $\Theta_1^- \prec_{\mathcal{J}} \Theta_2^-$. ◁

*Example 3 (Continued from Example 2).* The set $\Theta_2^-$ that consists of the extra source deletion $\Theta_2^- = \{SG(\texttt{sup1}, \texttt{grant1}); SG(\texttt{sup1}, \texttt{grant2})\}$ is a maximal compensation by deletion, and we have that $E_{\mathcal{J}}(\langle \emptyset, \mathcal{U}_A^+ \rangle, \langle \mathcal{U}_D^- \cup \Theta_2^-, \mathcal{U}_D^+ \rangle) = \langle \emptyset, \emptyset \rangle$. ◁

**Definition 14.** A set $\Theta_1^+$ is a *maximal compensation by insertion* of $E_{\mathcal{J}}(\mathcal{U}_A, \mathcal{U}_D)$ if there is no compensation by insertion $\Theta_2^+$ of $E_{\mathcal{J}}(\mathcal{U}_A, \mathcal{U}_D)$, such that $\Theta_1^+ \prec_{\mathcal{J}} \Theta_2^+$. ◁

### 4.1 Characterizing Maximal Compensations

From the definition of maximal compensation for both deletion and insertion given in the previous section, we notice that, in principle, it can be difficult to verify whether a compensation $\Theta^-$ is maximal or not. This is mainly due to the fact that it will require comparing $\Theta^-$ with all other compensations. In other words, it will require comparing the source deletion implied by $\Theta^-$ with every other possible source deletion, leading to minimizing the corresponding insertion side-effects. In this section, we focus on the problem of characterizing maximal compensations by deletion or insertion for a given ABox insertion or deletion, respectively.

For ABox insertions, a maximal compensation by deletion should ensure that no further tuples can be deleted from the data source to minimize the insertion side-effect further. This is the case when all ABox assertions in the insertion side-effect have at least one branch in the database that contributes to the initial ABox update.

**Proposition 15.** *Let $E_{\mathcal{J}}(\mathcal{U}_A, \mathcal{U}_D) = \langle E^-, E^+ \rangle$ and let $\Theta^-$ be a compensation by deletion such that $E_{\mathcal{J}}(\mathcal{U}_A, \langle \mathcal{U}_D^- \cup \Theta^-, \mathcal{U}_D^+ \rangle = \langle E^- \cup E_{\Delta}^-, E^+ \setminus E_{\Delta}^+ \rangle$. Then, $\Theta^-$ is a maximal compensation by deletion if and only if for every $f \in E^+ \setminus E_{\Delta}^+$ there exists a branch $B \in \mathsf{lineage}(f, \mathcal{P}, D \bullet \langle \mathcal{U}_D^- \cup \Theta^-, \mathcal{U}_D^+ \rangle)$ such that $B \subseteq \mathcal{U}_D^+$.*

For ABox deletions, one needs to ensure that no extra tuples can be inserted in the data source to minimize the deletion side-effect. Based on that, an intuitive way to characterize a maximal compensation by insertion is to check whether the assertions in the deletion side-effect cannot be inserted back without affecting the initially requested ABox deletion.

**Proposition 16.** *Let $E_{\mathcal{J}}(\mathcal{U}_A, \mathcal{U}_D) = \langle E^-, E^+ \rangle$ and let $\Theta^+$ be a compensation by insertion such that $E_{\mathcal{J}}(\mathcal{U}_A, \langle \mathcal{U}_D^-, \mathcal{U}_D^+ \cup \Theta^+ \rangle) = \langle E^- \setminus E_{\Delta}^-, E^+ \cup E_{\Delta}^+ \rangle$. Then, $\Theta^+$ is a maximal compensation by insertion if and only if for every $f \in E^- \setminus E_{\Delta}^-$, we have that $\mathcal{U}_A^- \cap \mathcal{M}(D \bullet \langle \mathcal{U}_D^-, \mathcal{U}_D^+ \cup \Theta^+ \rangle \cup T) \neq \emptyset$ for every minimal translation by insertion $T$ of $\{f\}$.*

**Algorithm 4:** CompByDel $(\mathcal{J}, \mathcal{U}_A, \mathcal{U}_D)$

**input :** A VKG instance $\mathcal{J} = \langle \mathcal{P}, D \rangle$. An ABox update $\mathcal{U}_A$.
A translation $\mathcal{U}_D = \langle \mathcal{U}_D^-, \mathcal{U}_D^+ \rangle$ of $\mathcal{U}_A$, where $\mathcal{U}_D^+$ is minimal.
**output:** A set of source deletions.

1 $\langle E^-, E^+ \rangle \leftarrow E_{\mathcal{J}}(\mathcal{U}_A, \mathcal{U}_D)$;
2 $\mathbf{T} = \{T_1, \ldots, T_n\} \leftarrow$ MinTransDel$(\langle \mathcal{P}, D \bullet \mathcal{U}_D \rangle, E^+)$;
3 $\mathbf{U}_D^- \leftarrow \{T \setminus \mathcal{U}_D^+ \mid T \in \mathbf{T}\}$;
4 **return** $\mathbf{U}_D^- \setminus \{\emptyset\}$;

## 5 Side-Effect Minimization in VKG Updates

We discuss now how to effectively minimize side-effects in VKG updates through an approach that recursively applies compensations for previously carried out data source insertions or deletions. We deal first with the case where we start with the insertion operation and with the corresponding compensation by deletion.

### 5.1 Computing Maximal Compensations by Deletion

For an ABox insertion $\mathcal{U}_A^+$, the goal of the compensation by deletion is to reduce the insertion side-effect $E^+$, while ensuring that the initial insertion $\mathcal{U}_A^+$ remains unaffected. However, we observe that while attempting to reduce $E^+$, we might inadvertently increase the deletion side-effect $E^-$. Due to the nature of VKG mappings, it might not always be possible to eliminate the insertion side-effects without affecting the initial ABox insertion. E.g., in Example 2, if $\mathcal{M}$ includes the assertion $RS(x, y) \rightsquigarrow Researcher(x)$, then the translation $RS(\texttt{john}, \texttt{sup1})$ of the ABox insertion $supervise(\texttt{sup1}, \texttt{john})$ will lead to the insertion side-effect $E^+ = \{Researcher(\texttt{john}), access(\texttt{john}, \texttt{grant1})\}$. In this case, it is impossible to delete the assertion $Researcher(\texttt{john})$ without affecting the original insertion.

When the insertion side-effect $E^+$ cannot be completely removed, we try to find its maximal subset that can be removed without affecting the original ABox insertion. In other words, we try to find a subset of the data source whose deletion will lead to a maximal compensation by deletion. Algorithm CompByDel (Alg. 4) takes as input a VKG instance $\mathcal{J}$, an ABox update $\mathcal{U}_A$, and its translation $\mathcal{U}_D = \langle \mathcal{U}_D^-, \mathcal{U}_D^+ \rangle$, and computes the set of direct translations by deletion (computed by MinTransDel) of the insertion side-effect (i.e., the second component of $E_{\mathcal{J}}(\mathcal{U}_A, \mathcal{U}_D)$) for the updated database $D \bullet \mathcal{U}_D$. It then eliminates (at Line 3) the tuples in $\mathcal{U}_D^+$ from each such translation, to ensure that tuples that are to be inserted are not part of the compensation by deletion. This is why we require that in the translation $\mathcal{U}_D$ given as input, $\mathcal{U}_D^+$ is minimal. We also observe that $\mathcal{U}_D^+$ might include tuples that originally were in $D$, and that therefore are actually *not* inserted (again) as part of the database update. However, since they are part of $\mathcal{U}_D^+$, Line 3 ensures that all tuples of $\mathcal{U}_A^+$ are kept in the ABox update.

**Theorem 17.** *Every set in* CompByDel$(\mathcal{J}, \mathcal{U}_A, \mathcal{U}_D)$ *is a maximal compensation by deletion of* $E_{\mathcal{J}}(\mathcal{U}_A, \mathcal{U}_D)$.

**Algorithm 5:** CompByIns $(\mathcal{J}, \mathcal{U}_A, \mathcal{U}_D)$

**input :** A VKG instance $\mathcal{J} = \langle \mathcal{P}, D \rangle$. An ABox update $\mathcal{U}_A = \langle \mathcal{U}_A^-, \mathcal{U}_A^+ \rangle$.
A translation $\mathcal{U}_D = \langle \mathcal{U}_D^-, \mathcal{U}_D^+ \rangle$ of $\mathcal{U}_A$, where $\mathcal{U}_D^-$ is minimal.
**output:** A set of source insertions.

1 $\langle E^-, E^+ \rangle \leftarrow E_{\mathcal{J}}(\mathcal{U}_A, \mathcal{U}_D)$;
2 $\mathbf{T} = \{T_1, \ldots, T_n\} \leftarrow$ MinTransIns$(\langle \mathcal{P}, D \bullet \mathcal{U}_D \rangle, E^-)$;
3 $\mathbf{S} \leftarrow \{T \setminus \mathcal{U}_D^- \mid T \in \mathbf{T}\}$;
4 $\mathbf{U}_D^+ \leftarrow \emptyset$;
5 **for each** translation $\Theta^+ \in \mathbf{S}$ such that $\Theta^+ \neq \emptyset$ **do**
6 $\quad \langle E_1^-, E_1^+ \rangle \leftarrow E_{\mathcal{J}}(\mathcal{U}_A, \langle \mathcal{U}_D^-, \mathcal{U}_D^+ \cup \Theta^+ \rangle)$;
7 $\quad$ **if** $E_1^+ \cap \mathcal{U}_A^- = \emptyset$ **then** $\mathbf{U}_D^+ \leftarrow \mathbf{U}_D^+ \cup \{\Theta^+\}$;
8 **return** $\mathbf{U}_D^+$;

### 5.2 Computing Maximal Compensations by Insertion

Similarly to how we address compensation by deletion for ABox insertions, the goal of compensation by insertion for ABox deletions is to reduce the deletion side-effect $E^-$, while ensuring that the initial deletion $\mathcal{U}_A^-$ remains unaffected. Also, note that while attempting to reduce $E^-$, we might unintendedly increase the insertion side-effect $E^+$.

Algorithm CompByIns (Alg. 5) takes as input a VKG instance $\mathcal{J}$, an ABox update $\mathcal{U}_A$, and its source translation $\mathcal{U}_D$, and computes the set of direct translations by insertion (computed by MinTransIns) of the deletion side-effect (i.e., the first component of $E_{\mathcal{J}}(\mathcal{U}_A, \mathcal{U}_D)$) for the updated database $D \bullet \mathcal{U}_D$. It then eliminates (at Line 3) the tuples in $\mathcal{U}_D^-$ from each translation to ensure disjointness between the final insertion and deletion translations. Finally, for each translation, it computes the insertion side-effect $E_1^+$ (Line 6) and rejects the translation if it contains some tuple from $\mathcal{U}_A^-$ (Line 7).

**Theorem 18.** *Every set $\Theta^+ \in$ CompByIns$(\mathcal{J}, \mathcal{U}_A, \mathcal{U}_D)$ is a maximal compensation by insertion of $E_{\mathcal{J}}(\mathcal{U}_A, \mathcal{U}_D)$.*

### 5.3 Computing Source Translations with Minimal Side-Effects

We now leverage the notion of compensation to minimize side-effects for translating a given ABox update. Since insertions and deletions are managed separately, our compensation mechanism will consist of a recursive sequence of insertions and deletions to minimize side-effects in the ABox. This means that, for a given ABox insertion $\mathcal{U}_A^+$ (resp., deletion $\mathcal{U}_A^-$) in $\mathcal{J}$, our algorithm returns a set of possible translations $\mathcal{U}_D = \langle \mathcal{U}_D^-, \mathcal{U}_D^+ \rangle$ over the source with minimum side-effects.

For the case of insertion, Algorithm PostInsertion (Alg. 6) takes as input an ABox insertion $\mathcal{U}_A^+$ and a set $\mathbf{T}$ of possible source translations. For each translation $\mathcal{U}_D = \langle \mathcal{U}_D^-, \mathcal{U}_D^+ \rangle \in \mathbf{T}$, it tries to minimize the insertion side-effect by applying CompByDel (Alg. 4) (at Line 3), which will lead to compensations by deletion of the initial insertion $\mathcal{U}_D^+$. For each compensation by deletion $\Theta^-$, it tries to minimize the deletion side-effect by applying CompByIns (Alg. 5) (at Line 7), which will lead to compensations by insertion $\Theta^+$. At Line 11, it adds to the solution $\mathbf{U}_D$ a recursive call

**Algorithm 6:** PostInsertion $(\mathcal{J}, \mathcal{U}_A^+, \mathbf{T})$

**input :** A VKG instance $\mathcal{J} = \langle \mathcal{P}, D \rangle$. A set $\mathcal{U}_A^+$ of ABox insertions. A set $\mathbf{T}$ of source translations of $\mathcal{U}_A^+$.
**output:** A set of source translations of $\mathcal{U}_A^+$.

1 $\mathbf{U}_D \leftarrow \emptyset$;
2 **for each** translation $\langle \mathcal{U}_D^-, \mathcal{U}_D^+ \rangle \in \mathbf{T}$ **do**
3 $\quad \mathbf{\Theta}^- = \{\Theta_1^-, \ldots, \Theta_n^-\} \leftarrow$ CompByDel$(\mathcal{J}, \mathcal{U}_A, \langle \mathcal{U}_D^-, \mathcal{U}_D^+ \rangle)$;
4 $\quad$ **if** $\mathbf{\Theta}^- = \emptyset$ **then** $\mathbf{U}_D \leftarrow \mathbf{U}_D \cup \{\langle \mathcal{U}_D^-, \mathcal{U}_D^+ \rangle\}$;
5 $\quad$ **else**
6 $\quad\quad$ **for each** compensation $\Theta^- \in \mathbf{\Theta}^-$ **do**
7 $\quad\quad\quad \mathbf{\Theta}^+ = \{\Theta_1^+, \ldots, \Theta_m^+\} \leftarrow$ CompByIns$(\mathcal{J}, \mathcal{U}_A, \langle \mathcal{U}_D^- \cup \Theta^-, \mathcal{U}_D^+ \rangle)$;
8 $\quad\quad\quad$ **if** $\mathbf{\Theta}^+ = \emptyset$ **then** $\mathbf{U}_D \leftarrow \mathbf{U}_D \cup \{\langle \mathcal{U}_D^- \cup \Theta^-, \mathcal{U}_D^+ \rangle\}$;
9 $\quad\quad\quad$ **else**
10 $\quad\quad\quad\quad \mathbf{S} = \{\langle \mathcal{U}_D^- \cup \Theta^-, \mathcal{U}_D^+ \cup \Theta^+ \rangle \mid \Theta^+ \in \mathbf{\Theta}^+\}$;
11 $\quad\quad\quad\quad \mathbf{U}_D \leftarrow \mathbf{U}_D \cup$ PostInsertion$(\mathcal{J}, \mathcal{U}_A^+, \mathbf{S})$;
12 **return** $\mathbf{U}_D$;

of PostInsertion over $\mathcal{U}_A^+$ and the set $\mathbf{S}$ of all possible pairs of compensations by deletion $\Theta^-$ and by insertion $\Theta^+$. Note that, if the set of compensations is empty, the algorithm keeps the solution without compensation (see Lines 4 and 8).

**Theorem 19.** *Let $\mathcal{J} = \langle \mathcal{P}, D \rangle$ be a VKG instance where $\mathcal{P} = \langle \emptyset, \mathcal{M}, \mathcal{S} \rangle$, $\mathcal{U}_A^+$ an ABox insertion, and $\mathbf{T}$ a set of translations of $\mathcal{U}_A^+$ over $\mathcal{J}$. Then, Algorithm* PostInsertion$(\mathcal{J}, \mathcal{U}_A^+, \mathbf{T})$ *always terminates.*

Finally, we propose Algorithm Insertion (Alg. 7) for an ABox insertion $\mathcal{U}_A^+$ over a VKG instance. It first computes the set of direct translations of $\mathcal{U}_A^+$ (Line 1) and applies

**Algorithm 7:** Insertion $(\mathcal{J}, \mathcal{U}_A^+)$

**input :** A VKG instance $\mathcal{J} = \langle \mathcal{P}, D \rangle$. A set $\mathcal{U}_A^+$ of ABox insertions.
**output:** A set of source translations of $\mathcal{U}_A^+$ with minimal side-effect.

1 $\mathbf{T} = \{T_1, \ldots, T_n\} \leftarrow$ MinTransIns$(\mathcal{J}, \mathcal{U}_A^+)$;
2 $\mathbf{\Theta} \leftarrow$ PostInsertion$(\mathcal{J}, \mathcal{U}_A^+, \{\langle \emptyset, T \rangle \mid T \in \mathbf{T}\})$;
3 $\mathbf{U}_D \leftarrow \{\}$;
4 **for each** translation $\Theta \in \mathbf{\Theta}$ **do**
5 $\quad$ **if** $\langle \mathcal{T}, \mathcal{M}(D \bullet \Theta) \rangle$ is consistent **then**
6 $\quad\quad add \leftarrow true$;
7 $\quad\quad$ **for each** translation $\Theta^\star \in \mathbf{U}_D$ **do**
8 $\quad\quad\quad$ **if** Compare$_\mathcal{J}(\Theta^\star, \Theta) =$ '<' **then** $add \leftarrow false$;
9 $\quad\quad\quad$ **else if** Compare$_\mathcal{J}(\Theta, \Theta^\star) =$ '<' **then** $\mathbf{U}_D \leftarrow \mathbf{U}_D \setminus \{\Theta^\star\}$;
10 $\quad\quad$ **if** $add$ **then** $\mathbf{U}_D \leftarrow \mathbf{U}_D \cup \{\Theta\}$;
11 **return** $\mathbf{U}_D$;

the POSTINSERTION (Alg. 6) mechanism on each direct translation (Line 2). It then compares the obtained translations and filters out the ones that lead to larger side-effects according to the abstract comparison function $\text{COMPARE}_{\mathcal{J}}$ (Lines 5 to 10). Notice that $\text{COMPARE}_{\mathcal{J}}$ compares two translations $\Theta_1$ and $\Theta_2$ in terms of their side-effect, and similar to $\text{COMPARE}^{-}_{\mathcal{J}}$ and $\text{COMPARE}^{+}_{\mathcal{J}}$, returns '<' if $\Theta_1$ has less side-effects that $\Theta_2$.

**Theorem 20.** *Let $\mathcal{J} = \langle \mathcal{P}, D \rangle$ be a VKG instance where $\mathcal{P} = \langle \emptyset, \mathcal{M}, \mathcal{S} \rangle$ and $\mathcal{U}_A^+$ an ABox insertion. Then, Algorithm* $\text{INSERTION}(\mathcal{J}, \mathcal{U}_A^+)$ *computes a set of source translations of $\mathcal{U}_A^+$ with minimal side-effect.*

The deletion procedure is symmetric to the insertion procedure. Algorithm POSTDELETION takes as input an ABox deletion $\mathcal{U}_A^-$ and a set $\mathbf{T}$ of possible source translations, and recursively applies compensations by insertion and deletion until no further compensation is possible. Algorithm DELETION takes as input an ABox deletion $\mathcal{U}_A^-$, computes a set $\mathbf{T}$ of possible direct translations, and also applies Algorithm POSTDELETION to minimize the side-effects of the deletions caused by the translations in $\mathbf{T}$.

## 6 Conclusions

This paper builds upon recent work on instance-level updates in VKGs [17]. Specifically, we propose a compensation procedure to address side-effects caused by ABox deletions and insertions, aiming to minimize unintended deletions and insertions, respectively. We introduce the concept of order among compensations for given ABox updates, leading to the notion of maximum compensation, and we explore its properties and justify its role in minimizing side-effects. Based on that, we proposed two methods, DELETION and INSERTION, that respectively take ABox deletions and ABox insertions, and recursively apply a sequence of maximum compensations by insertion and deletion in order to converge towards an update with minimum side-effect in the ABox. The computation of the maximum recovery and the lineage of ABox assertions that are essential parts of our proposed methods can be done at compile time, and the translation of ABox deletions can be done at run-time and is exponential in the size of its lineage branches. However, the translation of ABox insertions remains challenging due to the non-injective nature of VKG mappings. The complexity of finding the right combination is exponential in the size of the data source and the provided ABox insertion. In a practical setting, constraints over the source data or various techniques can be used to derive the right assignments.

We are currently working on implementing our algorithms by exploiting the query reformulation techniques of state-of-the-art tools for VKGs, specifically the open source system Ontop [4,21].

**Acknowledgments.** This research has been partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP), funded by the Knut and Alice Wallenberg Foundation, by the HEU project CyclOps (GA n. 101135513), by the Province of Bolzano and FWF through project OnTeGra (DOI 10.55776/PIN8884924). by the Province of Bolzano and EU through projects ERDF-FESR 1078 CRIMA, and ERDF-FESR 1047 AI-Lab, by MUR through the PRIN project 2022XERWK9 S-PIC4CHU, and by the EU and MUR through the PNRR project PE0000013-FAIR.

## References


1. Arenas, M., Pérez, J., Reutter, J., Riveros, C.: The language of plain SO-tgds: Composition, inversion and structural properties. J. of Computer and System Sciences **79**(6), 763–784 (2013). https://doi.org/10.1016/j.jcss.2013.01.002
2. Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Patel-Schneider, P.F. (eds.): The Description Logic Handbook: Theory, Implementation and Applications. Cambridge University Press (2003). https://doi.org/10.1017/CBO9780511711787
3. Bancilhon, F., Spyratos, N.: Update semantics of relational views. ACM Trans. on Database Systems **6**(4), 557–575 (1981). https://doi.org/10.1145/319628.319634
4. Calvanese, D., Cogrel, B., Komla-Ebri, S., Kontchakov, R., Lanti, D., Rezk, M., Rodriguez-Muro, M., Xiao, G.: Ontop: Answering SPARQL queries over relational databases. Semantic Web J. **8**(3), 471–487 (2017). https://doi.org/10.3233/SW-160217
5. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Poggi, A., Rodriguez-Muro, M., Rosati, R.: Ontologies and databases: The *DL-Lite* approach. In: Reasoning Web: Semantic Technologies for Informations Systems – 5th Int. Summer School Tutorial Lectures (RW), Lecture Notes in Computer Science, vol. 5689. Springer (2009). https://doi.org/10.1007/978-3-642-03754-2_7
6. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Rosati, R.: Tractable reasoning and efficient query answering in description logics: The DL-Lite family. J. of Automated Reasoning **39**, 385–429 (2007). https://doi.org/10.1007/s10817-007-9078-x
7. Chandra, A.K., Merlin, P.M.: Optimal implementation of conjunctive queries in relational data bases. In: Proc. of the 9th ACM Symp. on Theory of Computing (STOC). pp. 77–90 (1977)
8. Dayal, U., Bernstein, P.A.: On the correct translation of update operations on relational views. ACM Trans. on Database Systems **7**(3), 381–416 (1982). https://doi.org/10.1145/319732.319740
9. De Giacomo, G., Oriol, X., Rosati, R., Savo, D.F.: Instance-level update in DL-Lite ontologies through first-order rewriting. J. of Artificial Intelligence Research (2021). https://doi.org/10.1613/jair.1.12414
10. Fagin, R.: Inverting schema mappings. ACM Trans. on Database Systems **32**(4), 25:2–25:53 (2007). https://doi.org/10.1145/1292609.1292615
11. Fagin, R., Ullman, J.D., Vardi, M.Y.: On the semantics of updates in databases. In: Proc. of the 2nd ACM Symp. on Principles of Database Systems (PODS). pp. 352–365 (1983). https://doi.org/10.1145/588058.588100
12. Flouris, G.: On belief change in ontology evolution. AI Communications—The Eur. J. on Artificial Intelligence **19**(4) (2006)
13. Katsuno, H., Mendelzon, A.: On the difference between updating a knowledge base and revising it. In: Proc. of the 2nd Int. Conf. on Principles of Knowledge Representation and Reasoning (KR). pp. 387–394 (1991)
14. Kontchakov, R., Rezk, M., Rodriguez-Muro, M., Xiao, G., Zakharyaschev, M.: Answering SPARQL queries over databases under OWL 2 QL entailment regime. In: Proc. of the 13th Int. Semantic Web Conf. (ISWC). Lecture Notes in Computer Science, Springer (2014). https://doi.org/10.1007/978-3-319-11964-9_35
15. Motik, B., Cuenca Grau, B., Horrocks, I., Wu, Z., Fokoue, A., Lutz, C.: OWL 2 Web Ontology Language profiles (second edition). W3C Recommendation, World Wide Web Consortium (Dec 2012), available at http://www.w3.org/TR/owl2-profiles/
16. Poggi, A., Lembo, D., Calvanese, D., De Giacomo, G., Lenzerini, M., Rosati, R.: Linking data to ontologies. J. on Data Semantics **10**, 133–173 (2008). https://doi.org/10.1007/978-3-540-77688-8_5

17. Wandji, R.E., Calvanese, D.: Ontology-based update in virtual knowledge graphs via schema mapping recovery. In: Proc. of the 8th Int. Joint Conf. on Rules and Reasoning (RuleML+RR). pp. 59–74 (2024). https://doi.org/10.1007/978-3-031-72407-7_6
18. Winslett, M.: Updating Logical Databases. Cambridge University Press (1990)
19. Xiao, G., Calvanese, D., Kontchakov, R., Lembo, D., Poggi, A., Rosati, R., Zakharyaschev, M.: Ontology-based data access: A survey. In: Proc. of the 27th Int. Joint Conf. on Artificial Intelligence (IJCAI). pp. 5511–5519. IJCAI Org. (2018). https://doi.org/10.24963/ijcai.2018/777
20. Xiao, G., Ding, L., Cogrel, B., Calvanese, D.: Virtual Knowledge Graphs: An overview of systems and use cases. Data Intelligence (2019). https://doi.org/10.1162/dint_a_00011
21. Xiao, G., Lanti, D., Kontchakov, R., Komla-Ebri, S., Güzel-Kalayci, E., Ding, L., Corman, J., Cogrel, B., Calvanese, D., Botoeva, E.: The virtual knowledge graph system Ontop. In: Proc. of the 19th Int. Semantic Web Conf. (ISWC). Lecture Notes in Computer Science, vol. 12507, pp. 259–277. Springer (2020). https://doi.org/10.1007/978-3-030-62466-8_17
22. Zheleznyakov, D., Kharlamov, E., Nutt, W., Calvanese, D.: On expansion and contraction of DL-Lite knowledge bases. J. of Web Semantics **57**, 100484 (2019). https://doi.org/10.1016/j.websem.2018.12.002