A Dynamic Extension of ALCO for Repairing via Preferred Updates

Guillaume Feuillade, Andreas Herzig, Christos Rantsoudis

A Dynamic Extension of $\mathcal{ALCO}$ for Repairing via Preferred Updates

Guillaume Feuillade, Andreas Herzig, and Christos Rantsoudis

IRIT, Univ. Toulouse, France

Abstract. In the database literature it has been proposed to resort to active integrity constraints in order to restore database integrity. Such active integrity constraints consist of a classical constraint together with a set of preferred update actions that can be triggered when the constraint is violated. In this paper, we adapt this idea to Description Logics: we extend TBox axioms by a set of preferred updates of atomic concepts. We resort to a dynamic logic framework in order to provide an account of active TBox-based ABox repairs.

Keywords: ABox Repairs · Active Constraints · Dynamic Logic

1 Introduction

In Description Logics (DLs) the TBoxes are finite sets of axioms, i.e., general inclusions or definitions, that are viewed as non-contingent knowledge: knowledge that is typically stable in time. The ABoxes on the other hand contain contingent knowledge that typically changes more frequently than that of the TBox. Such changes may lead to inconsistencies w.r.t. TBoxes. Repairing an ABox that is inconsistent w.r.t. a TBox is a standard task [17, 5, 6]. The notion comes from the database literature: whenever a database does not conform to a set of integrity constraints it has to be repaired. This is usually called *database integrity maintenance* [9, 10, 4]. Active integrity constraints were proposed to provide preferred ways, out of all the possible ones, to restore the integrity of a database [13, 7, 8]. The idea was to enhance each integrity constraint with update actions indicating the preferred repair in case of inconsistency. For example, the integrity constraint $(\forall X)[\mathsf{Bachelor}(X) \wedge \mathsf{Married}(X) \rightarrow \bot]$ which says that no one should be identified as bachelor and married at the same time can be turned into the active constraint $(\forall X)[\mathsf{Bachelor}(X) \wedge \mathsf{Married}(X) \rightarrow \bot, \{-\mathsf{Bachelor}(X)\}]$, whose meaning is that when there is a person in the database who has both the status of being a bachelor and the status of being married then the preferred repair is to remove from the database the bachelor status (as opposed to removing the married status) since married status can be achieved from being bachelor but not the other way. In this way, the possible repairs are narrowed down as well as better match designer preferences when maintaining the database.

Now, applying the same idea to the DL setting requires the extension of the TBox axioms with update actions. Note that our perspective differs from [19, 21] since we do not want to single out integrity constraints from the TBox. The resulting extended

TBoxes are called *active* TBoxes in accordance with the nomenclature of *active* constraints. The authors have already investigated this research avenue in [20], which proposes a syntactic approach. Here we take a semantic approach and investigate how the various definitions of repairs from the database literature behave in the DL setting. We represent update actions as atomic dynamic logic programs and show that repairs can be represented as complex programs. The main reason for using this framework is to account for dynamic repairing procedures that check in multiple steps the status of a possible repair and terminate once consistency with the TBox has been achieved. This comes in contrast with methods used for active integrity constraints where the update is applied in one go and later checked if it follows the active axioms using certain criteria. The dynamic logic framework also allows us to lift the approach of [12] from propositional databases to DLs.

We consider that TBoxes are composed of *concept inclusion axioms* in the language of $\mathcal{ALC}$, without the unique name assumption. Unfortunately, there exist $\mathcal{ALC}$ ABoxes that when semantically updated cannot be expressed in $\mathcal{ALC}$ [18]. It is for that reason that our ABoxes may contain nominals. Both the TBox and the ABox can be represented in our language by single formulas.

Before moving on let us showcase an example of an active TBox that we will be able to construct and work with. First, consider the following TBox $\mathcal{T}$:

$$\{\mathsf{Sibling} \sqsubseteq \mathsf{Brother} \sqcup \mathsf{Sister}, \forall \mathsf{hasSibling}.\bot \sqsubseteq \mathsf{OnlyChild}\}$$

An active TBox extending $\mathcal{T}$ then is $\mathsf{a}\mathcal{T} = \{\eta_1, \eta_2\}$ where:

$$\eta_1 : \langle \mathsf{Sibling} \sqcap \neg \mathsf{Brother} \sqcap \neg \mathsf{Sister} \sqsubseteq \bot, \{-\mathsf{Sibling}\}\rangle$$
$$\eta_2 : \langle \forall \mathsf{hasSibling}.\bot \sqcap \neg \mathsf{OnlyChild} \sqsubseteq \bot, \{+\mathsf{OnlyChild}\}\rangle$$

Through these enhanced concept inclusions, we can be more specific in the update actions that we prefer when repairing an ABox that is inconsistent with $\mathcal{T}$: the active axioms of $\mathsf{a}\mathcal{T}$ 'dictate' that an individual who is neither a brother nor a sister of someone should drop their sibling status, whereas an individual who has no siblings should change its status and be an only child.

The paper is structured as follows. Section 2 presents syntax and semantics of formulas and programs. In Section 3 we prove decidability through reduction axioms eliminating programs from formulas. Section 4 is the main contribution of the paper where we import the idea behind active integrity constraints to DL. We then adapt the notions of *founded repairs* and *dynamic repairs* to the DL setting and use the programs of our language to constructively represent them. We conclude in Section 5.

2 Syntax and Semantics

We introduce the basics of DLs and Propositional Dynamic Logic PDL, referring to [2] and [14, 15] for more details. The update of an ABox will be represented with the help of PDL programs. Their behavior is identical with the programs of PDL, with formulas of the form $\langle\pi\rangle\varphi$ expressing that φ is true after *some* possible execution of the program π. The main difference is at the atomic level: whereas PDL has abstract

atomic programs, the atomic programs of our language have the form $\pm A(a)$ where $A(a)$ is an atomic assertion. This makes a big difference: the programs of our language can be eliminated and every formula is reducible to a *static formula*, i.e., a formula without programs. This is crucial in acquiring a unique representation of the update of an ABox through a static formula. We note that the models of our logic are different from (and quite simpler than) the Kripke models of PDL: the familiar interpretations of Description Logic suffice.

2.1 Language

Let Con be a countable set of atomic concepts, R a countable set of roles and Ind a countable set of individual names. The language of formulas and programs is defined by the following grammar:

$$\varphi ::= A \mid \{a\} \mid \bot \mid \varphi \to \varphi \mid \exists r.\varphi \mid \mathcal{U}\varphi \mid \langle\pi\rangle\varphi \mid \langle\pi\rangle^{c}\varphi$$
$$\pi ::= +A(a) \mid -A(a) \mid \pi;\pi \mid \pi \cup \pi \mid \pi^{*} \mid \varphi?$$

where $A \in \mathsf{Con}$, $a \in \mathsf{Ind}$, and $r \in \mathsf{R}$. *Atomic programs* have the form $+A(a)$ and $-A(a)$. Sometimes we will use the expression $\pm A(a)$ to talk about both. The operators of sequential and nondeterministic composition, the Kleene star and the test are familiar from PDL. $\mathcal{U}\varphi$ expresses the fact that φ is true for every individual of an interpretation.

The logical connectives $\neg$, $\top$, $\wedge$, $\vee$ and $\leftrightarrow$ as well as the universal quantifier $\forall$ are abbreviated in the usual way. The expression π^{n} abbreviates the program $\pi;\ldots;\pi$ where π appears n times, and $\pi^{\leq n}$ abbreviates the program $(\pi \cup \top?)^{n}$. We also define π^{+} to be $\pi;\pi^{*}$. Furthermore, **while** φ **do** π abbreviates the program $(\varphi?;\pi)^{*};\neg\varphi?$ and $a : \varphi$ abbreviates the formula $\mathcal{U}(\{a\} \to \varphi)$. Finally, $r(a,b)$ abbreviates the formula $\mathcal{U}(\{a\} \to \exists r.\{b\})$.

We use the language of the basic description logic $\mathcal{ALC}$ for TBoxes and that of its extension $\mathcal{ALCO}$ with nominals for ABoxes. The TBoxes are composed of concept inclusion axioms and the ABoxes contain concept assertions $a : C$ and role assertions $r(a,b)$ where C is any concept description in $\mathcal{ALCO}$, r is a role and a, b are individuals. We will identify a TBox $\mathcal{T}$ and an ABox $\mathcal{A}$, respectively, with the formulas:

$$\bigwedge_{C \sqsubseteq D \in \mathcal{T}} \mathcal{U}(C \to D) \quad \text{and} \quad \bigwedge_{a:C \in \mathcal{A}} (a:C) \wedge \bigwedge_{r(a,b) \in \mathcal{A}} r(a,b)$$

Finally, $\mathsf{Con}(\pi)$ denotes the set of all atomic concepts occurring in the program π. Similarly, $\mathsf{Ind}(\pi)$ denotes the set of all individuals occurring in π.

2.2 Interpretations and their Updates

Interpretations are couples $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ where $\Delta^{\mathcal{I}}$ is the domain and $\cdot^{\mathcal{I}}$ maps each atomic concept A to a subset of $\Delta^{\mathcal{I}}$, each role r to a binary relation on $\Delta^{\mathcal{I}}$ and each individual name a to an element of $\Delta^{\mathcal{I}}$, i.e., $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$, $r^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ and $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$.

An *indexical update action* is an expression of the form $+A$ or $-A$, where A is an atomic concept. A *(non-indexical) update action* is an expression of the form $+A(a)$ or $-A(a)$, where A is an atomic concept and a an individual. A set of update actions U is consistent iff it does not contain both $+A(a)$ and $-A(a)$ for some assertion $A(a)$.

Definition 1. *Let U be a consistent set of update actions. The update of the interpretation $\mathcal{I}$ by U, denoted by $\mathcal{I} \diamond U$, is the interpretation $\mathcal{I}'$ such that:*

- $\Delta^{\mathcal{I}'} = \Delta^{\mathcal{I}}$
- $A^{\mathcal{I}'} = (A^{\mathcal{I}} \cup \{a^{\mathcal{I}} \mid +A(a) \in U\}) \setminus \{a^{\mathcal{I}} \mid -A(a) \in U\}$
- $r^{\mathcal{I}'} = r^{\mathcal{I}}$
- $a^{\mathcal{I}'} = a^{\mathcal{I}}$

Note that a consistent U can be identified with a program π_U, supposing that the update actions of U are applied in sequence (where, thanks to consistency, the order does not matter).

2.3 Semantics

The extension of $\cdot^{\mathcal{I}}$ to complex formulas is defined inductively as follows:

$$
\begin{aligned}
\{a\}^{\mathcal{I}} &= \{a^{\mathcal{I}}\} \\
\bot^{\mathcal{I}} &= \emptyset \\
(\varphi \to \psi)^{\mathcal{I}} &= (\Delta^{\mathcal{I}} \setminus \varphi^{\mathcal{I}}) \cup \psi^{\mathcal{I}} \\
(\exists r.\varphi)^{\mathcal{I}} &= \{\delta \in \Delta^{\mathcal{I}} \mid \text{ there is } \delta' \in \Delta^{\mathcal{I}} \text{ such that } (\delta, \delta') \in r^{\mathcal{I}} \text{ and } \delta' \in \varphi^{\mathcal{I}}\} \\
(\mathcal{U}\varphi)^{\mathcal{I}} &= \begin{cases} \Delta^{\mathcal{I}} & \text{if } \varphi^{\mathcal{I}} = \Delta^{\mathcal{I}} \\ \emptyset & \text{otherwise} \end{cases} \\
(\langle \pi \rangle \varphi)^{\mathcal{I}} &= \bigcup_{(\mathcal{I},\mathcal{I}') \in \|\pi\|} \varphi^{\mathcal{I}'} \\
(\langle \pi \rangle^{\circ} \varphi)^{\mathcal{I}} &= \bigcup_{(\mathcal{I}',\mathcal{I}) \in \|\pi\|} \varphi^{\mathcal{I}'}
\end{aligned}
$$

while the semantics of programs is:

$$
\begin{aligned}
(\mathcal{I}, \mathcal{I}') \in \| +A(a) \| &\text{ iff } \mathcal{I}' = \mathcal{I} \diamond \{+A(a)\} \\
(\mathcal{I}, \mathcal{I}') \in \| -A(a) \| &\text{ iff } \mathcal{I}' = \mathcal{I} \diamond \{-A(a)\} \\
(\mathcal{I}, \mathcal{I}') \in \| \pi_1; \pi_2 \| &\text{ iff there exists } \mathcal{I}'' \text{ such that } (\mathcal{I}, \mathcal{I}'') \in \| \pi_1 \| \text{ and } (\mathcal{I}'', \mathcal{I}') \in \| \pi_2 \| \\
(\mathcal{I}, \mathcal{I}') \in \| \pi_1 \cup \pi_2 \| &\text{ iff } (\mathcal{I}, \mathcal{I}') \in \| \pi_1 \| \cup \| \pi_2 \| \\
(\mathcal{I}, \mathcal{I}') \in \| \varphi? \| &\text{ iff } \mathcal{I}' = \mathcal{I} \text{ and } \varphi^{\mathcal{I}} = \Delta^{\mathcal{I}} \\
(\mathcal{I}, \mathcal{I}') \in \| \pi^* \| &\text{ iff } (\mathcal{I}, \mathcal{I}') \in \bigcup_{k \in \mathbb{N}_0} \| \pi \|^k
\end{aligned}
$$

An interpretation $\mathcal{I}$ is a *model of* a formula φ iff $\varphi^{\mathcal{I}} = \Delta^{\mathcal{I}}$. We say that φ is *(globally) satisfiable* iff there exists a model of φ. We say that φ is *valid* iff every interpretation is a model of φ. We call the logic that is built using this syntax and semantics $\mathsf{dyn}\mathcal{ALCO}$.

The *update of an ABox $\mathcal{A}$ by a consistent set of update actions U* is the set:

$$\mathcal{A} \diamond U = \{\mathcal{I} \diamond U \mid \mathcal{I} \text{ is a model of } \mathcal{A}\}$$

This is a semantic definition. $\mathcal{A} \diamond U$ has also at least one syntactic representation, but it is not unique: there are many ABoxes that can be used to describe it. In particular, the set $\mathcal{A} \diamond U$ equals the set of interpretations satisfying $\langle \pi_U \rangle^{c} \mathcal{A}$, where π_U is the program that applies the update actions of U.

3 Reduction Axioms and Decidability

We now show how to convert any $\mathsf{dyn}\mathcal{ALCO}$ formula to an equivalent *static* formula, i.e., a formula without programs. We first reduce complex programs to atomic programs and then eliminate atomic programs from formulas. This is familiar e.g. from Dynamic Epistemic Logics, see [11]. We start with the reduction axioms for complex programs.

Proposition 1. *The following equivalences are valid:*

$$\begin{aligned}
\langle \pi_1 ; \pi_2 \rangle \varphi &\leftrightarrow \langle \pi_1 \rangle \langle \pi_2 \rangle \varphi \\
\langle \pi_1 \cup \pi_2 \rangle \varphi &\leftrightarrow \langle \pi_1 \rangle \varphi \vee \langle \pi_2 \rangle \varphi \\
\langle \pi^* \rangle \varphi &\leftrightarrow \langle \pi^{\leq 2^n} \rangle \varphi \\
\langle \psi ? \rangle \varphi &\leftrightarrow \mathcal{U}\psi \wedge \varphi
\end{aligned}$$

where $n = \mathsf{card}(\mathsf{Con}(\pi)) \times \mathsf{card}(\mathsf{Ind}(\pi))$.

Observe that, contrarily to PDL and just as in DL-PA [3], the Kleene star can be eliminated. The next proposition shows how to reduce atomic programs.

Proposition 2. *The following equivalences are valid:*[1]

$$\begin{aligned}
\langle +A(a) \rangle B &\leftrightarrow \begin{cases} \{a\} \vee B & \text{if } A = B \\ B & \text{otherwise} \end{cases} \\
\langle -A(a) \rangle B &\leftrightarrow \begin{cases} \neg\{a\} \wedge B & \text{if } A = B \\ B & \text{otherwise} \end{cases} \\
\langle \pm A(a) \rangle \{b\} &\leftrightarrow \{b\} \\
\langle \pm A(a) \rangle \neg\varphi &\leftrightarrow \neg\langle \pm A(a) \rangle \varphi \\
\langle \pm A(a) \rangle (\varphi \vee \psi) &\leftrightarrow \langle \pm A(a) \rangle \varphi \vee \langle \pm A(a) \rangle \psi \\
\langle \pm A(a) \rangle \exists r.\varphi &\leftrightarrow \exists r.\langle \pm A(a) \rangle \varphi \\
\langle \pm A(a) \rangle \mathcal{U}\varphi &\leftrightarrow \mathcal{U}\langle \pm A(a) \rangle \varphi
\end{aligned}$$

where $A, B \in \mathsf{Con}$, $r \in \mathsf{R}$ *and* $a, b \in \mathsf{Ind}$.

Finally, the reduction axioms for the converse operator follow.

[1] Note that we have used the logical connectives $\neg$ and $\vee$ in the place of $\bot$ and $\rightarrow$ since they are easier to present.

Proposition 3. *The following equivalences are valid:*

$$\langle +A(a)\rangle^{c}\varphi \leftrightarrow (a:A) \wedge (\varphi \vee \langle -A(a)\rangle\varphi)$$
$$\langle -A(a)\rangle^{c}\varphi \leftrightarrow (a:\neg A) \wedge (\varphi \vee \langle +A(a)\rangle\varphi)$$
$$\langle \pi_1;\pi_2\rangle^{c}\varphi \leftrightarrow \langle \pi_2\rangle^{c}\langle \pi_1\rangle^{c}\varphi$$
$$\langle \pi_1 \cup \pi_2\rangle^{c}\varphi \leftrightarrow \langle \pi_1\rangle^{c}\varphi \vee \langle \pi_2\rangle^{c}\varphi$$
$$\langle \pi^{*}\rangle^{c}\varphi \leftrightarrow \langle \pi^{\leq 2^n}\rangle^{c}\varphi$$
$$\langle \psi?\rangle^{c}\varphi \leftrightarrow \langle \psi?\rangle\varphi$$

Using now Propositions 1, 2 and 3 we obtain the following theorem.

Theorem 1. *Every formula of* dyn$\mathcal{ALCO}$ *is equivalent to a static formula.*

Through Theorem 1 we can now obtain a unique syntactical representation of the set $\mathcal{A} \diamond U$ by reducing the formula $\langle \pi_U\rangle^{c}\mathcal{A}$ to a static one containing no programs.

Example 1 ([18]). Let $\mathcal{A} = \{\mathsf{John} : \exists \mathsf{hasChild.Happy}, \mathsf{Mary} : \mathsf{Happy} \sqcap \mathsf{Clever}\}$ and $U = \{-\mathsf{Happy(Mary)}\}$. Applying the reduction axioms to:

$$\langle -\mathsf{Happy(Mary)}\rangle^{c}((\mathsf{John} : \exists \mathsf{hasChild.Happy}) \wedge (\mathsf{Mary} : \mathsf{Happy} \wedge \mathsf{Clever}))$$

we obtain the static formula:

$$(\mathsf{John} : \exists \mathsf{hasChild.}(\mathsf{Happy} \vee \{\mathsf{Mary}\})) \wedge (\mathsf{Mary} : \neg\mathsf{Happy} \wedge \mathsf{Clever})$$

which accurately represents $\mathcal{A} \diamond U$.

Last but not least, decidability of global satisfiability in dyn$\mathcal{ALCO}$ follows from, first, the fact that any static formula can be mapped to an $\mathcal{ALCO}(\mathcal{U})$-concept and vice versa, where $\mathcal{ALCO}(\mathcal{U})$ denotes the language $\mathcal{ALCO}$ together with the universal role, and, second, the fact that concept satisfiability in $\mathcal{ALCO}(\mathcal{U})$ is decidable [16]. Thus, (global) satisfiability checking of the static fragment of our logic can be reduced to the satisfiability problem of $\mathcal{ALCO}(\mathcal{U})$. We then obtain the following theorem.

Theorem 2. *Global satisfiability in the logic* dyn$\mathcal{ALCO}$ *is decidable.*

4 Active Inclusion Axioms in $\mathcal{ALC}$ TBoxes

In this section we import the idea behind active integrity constraints into DL and examine dynamic ways an ABox can be repaired via the preferred update actions suggested by the active axioms. To do this we first have to enrich concept inclusions with update actions, indicating the preferred ways to be repaired in case of inconsistency.

We start with the definitions of static and active concept inclusions.

Definition 2. *A concept inclusion of the form* $C_1 \sqcap \cdots \sqcap C_n \sqsubseteq \bot$ *is called a* static *concept inclusion.*

Note that in $\mathcal{ALC}$, any concept inclusion axiom is equivalent to a static concept inclusion.

Definition 3. *Let $\rho = C_1 \sqcap \cdots \sqcap C_n \sqsubseteq \bot$ be a static concept inclusion. An* active *concept inclusion is of the form $\langle \rho, V \rangle$ where V is a set of indexical update actions such that:*

- *if $+A \in V$ then there exists $C_i = \neg A$ with $A \in \mathsf{Con}$.*
- *if $-A \in V$ then there exists $C_i = A$ with $A \in \mathsf{Con}$.*

An active *TBox, denoted by $\mathsf{a}\mathcal{T}$, is a set of active concept inclusions.*

For an active concept inclusion $\eta = \langle \rho, V \rangle$ we let $\mathsf{static}(\eta) = \rho$ and $\mathsf{active}(\eta) = V$. Note that $\mathsf{active}(\eta)$ can be empty. For an active TBox $\mathsf{a}\mathcal{T}$ we let $\mathsf{static}(\mathsf{a}\mathcal{T}) = \{\mathsf{static}(\eta) : \eta \in \mathsf{a}\mathcal{T}\}$ and $\mathsf{active}(\mathsf{a}\mathcal{T}) = \bigcup_{\eta \in \mathsf{a}\mathcal{T}} \mathsf{active}(\eta)$. We say that $\mathsf{a}\mathcal{T}$ extends $\mathcal{T}$ iff $\mathcal{T}$ is equivalent to $\mathsf{static}(\mathsf{a}\mathcal{T})$.

Going back to the example of Section 1, we observe that the active TBox $\mathsf{a}\mathcal{T}$ conforms to the above definitions. Note also that $\mathsf{static}(\mathsf{a}\mathcal{T})$ is equivalent to $\mathcal{T}$.

From now on we consider some fixed finite sets Con of atomic concepts and Ind of individuals. Furthermore, we consider a fixed consistent TBox $\mathcal{T}$ and a fixed active TBox $\mathsf{a}\mathcal{T}$ extending $\mathcal{T}$. Finally, we suppose that all ABoxes we work with are consistent. An ABox $\mathcal{A}$ is inconsistent with respect to $\mathcal{T}$ iff there is no interpretation satisfying both $\mathcal{A}$ and $\mathcal{T}$. Note that in $\mathsf{dyn}\mathcal{ALCO}$ this amounts to unsatisfiability of the formula $\mathcal{T} \wedge \mathcal{A}$.

In the next two subsections, we define the notions of *founded* and *dynamic* repairs of an ABox $\mathcal{A}$ with respect to $\mathsf{a}\mathcal{T}$, which choose among the update actions in $\mathsf{active}(\mathsf{a}\mathcal{T})$ and modify the ABox such that $\mathcal{A}$ is consistent with the concept inclusion axioms of $\mathcal{T}$. The former are based on [7, 8] while the latter are based on [12].

4.1 Founded Weak Repairs and Founded Repairs

We start with the notion of *foundedness*, which is a key condition that a repair should satisfy for its update actions to be supported by the active concept inclusions.

Definition 4. *Let $\mathcal{I}$ be an interpretation. A set of update actions U is founded with respect to $\mathsf{a}\mathcal{T}$ and $\mathcal{I}$ if for every $\pm A(a) \in U$ there exists an $\eta \in \mathsf{a}\mathcal{T}$ such that:*

- $\pm A \in \mathsf{active}(\eta)$
- *$\mathcal{I} \diamond U$ is a model of* $\mathsf{static}(\eta)$
- *$\mathcal{I} \diamond (U \setminus \{\pm A(a)\})$ is not a model of* $\mathsf{static}(\eta)$

Based on this, the definitions of a *founded weak repair* and a *founded repair* follow.

Definition 5. *Let $\mathsf{static}(\mathsf{a}\mathcal{T}) \wedge \mathcal{A}$ be unsatisfiable. A* founded weak repair *of $\mathcal{A}$ is a consistent set of update actions U such that* (1) *$\mathcal{T} \wedge (\mathcal{A} \diamond U)$ is satisfiable and* (2) *there is a model $\mathcal{I}$ of $\mathcal{A}$ such that U is founded with respect to $\mathsf{a}\mathcal{T}$ and $\mathcal{I}$. If moreover $\mathcal{T} \wedge (\mathcal{A} \diamond U')$ is unsatisfiable for all $U' \subset U$, then U is a* founded repair *of $\mathcal{A}$.*

The following simple example showcases this definition.

Example 2. Consider the TBox $\mathcal{T} = \{\mathsf{Born} \sqsubseteq \mathsf{Alive}, \top \sqsubseteq \mathsf{Alive} \sqcup \mathsf{Dead}\}$. Consider also the following active TBox which extends $\mathcal{T}$:

$$\mathsf{a}\mathcal{T} = \{\langle \mathsf{Born} \sqcap \neg\mathsf{Alive} \sqsubseteq \bot, \{+\mathsf{Alive}\}\rangle, \langle \neg\mathsf{Alive} \sqcap \neg\mathsf{Dead} \sqsubseteq \bot, \{+\mathsf{Dead}\}\rangle\}$$

Furthermore, consider the ABox $\mathcal{A} = \{\mathsf{John} : \mathsf{Born} \sqcap \neg\mathsf{Alive} \sqcap \neg\mathsf{Dead}\}$ which is inconsistent with $\mathcal{T}$. The set $\{+\mathsf{Alive}(\mathsf{John})\}$ is the only founded weak repair of $\mathcal{A}$. Indeed, the second update action in $\{+\mathsf{Alive}(\mathsf{John}), +\mathsf{Dead}(\mathsf{John})\}$ cannot be founded on the second active axiom of $\mathsf{a}\mathcal{T}$. It is also the only founded repair.

We now extend the set of atomic concepts Con by new concepts A^+ and A^- uniquely associated with each concept A. They allow us to keep track of the concepts that are added to or removed from an ABox. We use these then to define the following program:

$$\mathsf{toggle}^{\pm}(A(a)) = \big(\neg(a{:}A^+) \wedge \neg(a{:}A^-)\big)\,?\,;\big((+A(a)\,;+A^+(a)) \cup (-A(a)\,;+A^-(a))\big)$$

which is intuitively the program $+A(a) \cup -A(a)$ but enhanced with update operations that keep track of any changes.

Next, in order to find a founded weak repair we have to ensure that the foundedness condition of Definition 4 holds after the update actions of the $\mathsf{toggle}^{\pm}(A(a))$ program. To do this, we define the following formula:

$$\mathsf{Founded} = \bigwedge_{\substack{A\in\mathsf{Con}\\ a\in\mathsf{Ind}}} \Big((a{:}A^+) \vee (a{:}A^-) \rightarrow \bigvee_{\substack{\eta\in\mathsf{a}\mathcal{T}\\ \pm A\in\mathsf{active}(\eta)}} \langle\, +A(a) \cup -A(a)\rangle\neg\mathsf{static}(\eta)\Big)$$

which does exactly that: it checks for all concepts that have been added to or removed from an assertion that they belong to the active part of some concept inclusion and that, without them, the static part of this same concept inclusion is violated. Using the aforementioned we can now define the program that searches for founded weak repairs:

$$\mathsf{foundedWeakRepair} = \Big(\bigcup_{\substack{A\in\mathsf{Con}\\ a\in\mathsf{Ind}}} \mathsf{toggle}^{\pm}(A(a))\Big)^*;\mathcal{T}?;\mathsf{Founded}?$$

The next step is to define the program:

$$\mathsf{undo}(A(a)) = \big((a{:}A^+)\,?\,;-A(a)\,;-A^+(a)\big) \;\cup\; \big((a{:}A^-)\,?\,;+A(a)\,;-A^-(a)\big)$$

which, as the name suggests, will undo any change that was imposed on an assertion.

Now, in order to redo any changes that are stored through A^+ and A^- to an alternative interpretation, we use the program:

$$\mathsf{redo}(A(a)) = \big((a{:}A^+)\,?\,;+A(a)\big) \cup \big((a{:}A^-)\,?\,;-A(a)\big)$$

The last step in checking minimality is to define a program that visits all models of the original ABox $\mathcal{A}$. This will allow us to check whether we can obtain a founded weak repair using less update actions. It is easy to see that this can be achieved through the program:

$$\mathsf{gotoAltInt}(\mathcal{A}) = \Big(\bigcup_{\substack{A\in\mathsf{Con}\\ a\in\mathsf{Ind}}} (\,+A(a) \cup -A(a))\Big)^*\,;\,\mathcal{A}\,?$$

Summing up, we use the above to create the formula:

$$\mathsf{Minimal}(\mathcal{A}) = \neg \langle\, \mathsf{gotoAltInt}(\mathcal{A})\,;\Big(\bigcup_{\substack{A\in\mathsf{Con}\\ a\in\mathsf{Ind}}} \mathsf{redo}(A(a))\Big)^{*};\Big(\bigcup_{\substack{A\in\mathsf{Con}\\ a\in\mathsf{Ind}}} \mathsf{undo}(A(a))\Big)^{+} \rangle\, \mathcal{T}$$

which is the key ingredient in checking if a repair is minimal with respect to other repairs. Note that the $\mathsf{redo}(A(a))$ program may not actually reapply all of the update actions that may have been stored earlier through the $\mathsf{toggle}^{\pm}(A(a))$ program: it does not need to, as reapplying only some of them is equivalent to reapplying all of them and undoing all those that were left out.

We can now define the program that searches for founded repairs:

$$\mathsf{foundedRepair}(\mathcal{A}) = \mathsf{foundedWeakRepair}\,;\, \mathsf{Minimal}(\mathcal{A})\,?$$

Last but not least, let $\mathsf{makeFalse}^{\pm}$ abbreviate the program which falsifies all the assertions of the form $A^{+}(a)$ and $A^{-}(a)$ for all $A \in \mathsf{Con}$ and $a \in \mathsf{Ind}$. The program $\mathsf{makeFalse}^{\pm}$ will be used in the final step to make sure that none of the new concepts that were used for storing purposes survive in the repair.

We showcase all of the above in the following theorem, which completes the picture.

Theorem 3. *Let $\mathcal{A}$ be inconsistent with respect to $\mathsf{static}(\mathsf{a}\mathcal{T})$ and let U be a consistent set of update actions. Furthermore, let no atomic concept A^{+} or A^{-} appear in any of them.*

- *U is a founded weak repair of $\mathcal{A}$ iff there exists an $\mathcal{I}$ such that:*

$$(\mathcal{I}, \mathcal{I}\diamond U) \in \Big\|\Big(\mathcal{A}\wedge \bigwedge_{\substack{A\in\mathsf{Con}\\ a\in\mathsf{Ind}}} (\neg(a:A^{+})\wedge\neg(a:A^{-}))\Big)?;\mathsf{foundedWeakRepair};\mathsf{makeFalse}^{\pm}\Big\|$$

- *U is a founded repair of $\mathcal{A}$ iff there exists an $\mathcal{I}$ such that:*

$$(\mathcal{I}, \mathcal{I}\diamond U) \in \Big\|\Big(\mathcal{A}\wedge \bigwedge_{\substack{A\in\mathsf{Con}\\ a\in\mathsf{Ind}}} (\neg(a:A^{+})\wedge\neg(a:A^{-}))\Big)\,?\,;\,\mathsf{foundedRepair}(\mathcal{A})\,;\mathsf{makeFalse}^{\pm}\,\Big\|$$

Going back to Example 2, we can now witness how, taking $\mathcal{I}$ to be a model of $\{\mathsf{John} : \mathsf{Born}\sqcap\neg\mathsf{Alive}\sqcap\neg\mathsf{Dead}\}$, the set $\{+\mathsf{Alive}(\mathsf{John})\}$ satisfies both of the conditions of Theorem 3. On the other hand, the set $\{+\mathsf{Alive}(\mathsf{John}), +\mathsf{Dead}(\mathsf{John})\}$ doesn't satisfy them for any interpretation $\mathcal{I}$.

4.2 Dynamic Weak Repairs and Dynamic Repairs

We will now investigate dynamic repairs which exploit even better the programs and the setting of Dynamic Logic in order to provide a different view of repairs. We begin with some definitions.

For every active concept inclusion $\eta \in \mathsf{a}\mathcal{T}$ and individual a, the programs π^{a}_{η} and ${}^{\pm}\pi^{a}_{\eta}$ are defined as follows:

$$\pi^{a}_{\eta} = \neg\,(a:\mathsf{static}(\eta))\,?\,;\bigcup_{\pm A\in\mathsf{active}(\eta)} \pm A(a)$$

$$
\begin{aligned}
{}^{+}\pi^{a}_{\eta} &= \bigcup_{+A \in \mathsf{active}(\eta)} (\,+A(a)\,;+A^{+}(a)) \\
{}^{-}\pi^{a}_{\eta} &= \bigcup_{-A \in \mathsf{active}(\eta)} (\,-A(a)\,;+A^{-}(a)) \\
{}^{\pm}\pi^{a}_{\eta} &= \neg\,(a:\mathsf{static}(\eta))\,?\,;({}^{+}\pi^{a}_{\eta} \;\cup\; {}^{-}\pi^{a}_{\eta})
\end{aligned}
$$

Intuitively, the program π^{a}_{η} will check for each active concept inclusion η and individual a whether the static concept inclusion in η is violated at a, and if so, will try to repair it using only update actions that are specified by η. The program ${}^{\pm}\pi^{a}_{\eta}$ furthermore stores the concepts that have been changed.

Using the program π^{a}_{η} we can now formally define *dynamic weak repairs* and *dynamic repairs*.

Definition 6. *Let* $\mathsf{static}(\mathsf{a}\mathcal{T}) \wedge \mathcal{A}$ *be unsatisfiable. A consistent set of update actions* U *is a* dynamic weak repair *of* $\mathcal{A}$ *iff there exists an* $\mathcal{I}$ *such that:*

$$
(\mathcal{I}, \mathcal{I} \diamond U) \in \Big\| \mathcal{A}\,?\,; \textbf{while } \big(\neg\mathcal{T}\big) \textbf{ do } \Big(\bigcup_{\substack{a \in \mathsf{Ind} \\ \eta \in \mathsf{a}\mathcal{T}}} \pi^{a}_{\eta} \Big) \Big\|
$$

If $\mathcal{T} \wedge (\mathcal{A} \diamond U')$ *is unsatisfiable for all* $U' \subset U$*, then* U *is a* dynamic repair *of* $\mathcal{A}$*.*

It is worth noting that dynamic weak repairs are not necessarily founded. The next example showcases this.

Example 3 (Example 2, ctd.). Consider again the active TBox:

$$
\mathsf{a}\mathcal{T} = \{\langle \mathsf{Born} \sqcap \neg\mathsf{Alive} \sqsubseteq \bot, \{+\mathsf{Alive}\}\rangle, \langle \neg\mathsf{Alive} \sqcap \neg\mathsf{Dead} \sqsubseteq \bot, \{+\mathsf{Dead}\}\rangle\}
$$

and the ABox $\mathcal{A} = \{\mathsf{John} : \mathsf{Born} \sqcap \neg\mathsf{Alive} \sqcap \neg\mathsf{Dead}\}$, whose only founded weak repair was $\{+\mathsf{Alive}(\mathsf{John})\}$. There are two dynamic weak repairs of $\mathcal{A}$, namely $U_1 = \{+\mathsf{Alive}(\mathsf{John})\}$ and $U_2 = \{+\mathsf{Alive}(\mathsf{John}), +\mathsf{Dead}(\mathsf{John})\}$. Only U_1 is a dynamic repair.

We will use now the program ${}^{\pm}\pi^{a}_{\eta}$ together with the formula $\mathsf{Minimal}(\mathcal{A})$ to show how a dynamic repair can be extracted using the familiar procedure of the previous theorem. Defining:

$$
\mathsf{dynamicRepair}(\mathcal{A}) = \textbf{while } \big(\neg\mathcal{T}\big) \textbf{ do } \Big(\bigcup_{\substack{a \in \mathsf{Ind} \\ \eta \in \mathsf{a}\mathcal{T}}} {}^{\pm}\pi^{a}_{\eta} \Big)\,;\; \mathsf{Minimal}(\mathcal{A})\,?
$$

we have the following theorem.

Theorem 4. *Let* $\mathcal{A}$ *be inconsistent with respect to* $\mathsf{static}(\mathsf{a}\mathcal{T})$ *and let* U *be a consistent set of update actions. Furthermore, let no atomic concept* A^{+} *or* A^{-} *appear in any of them.* U *is a dynamic repair of* $\mathcal{A}$ *iff there exists an* $\mathcal{I}$ *such that:*

$$
(\mathcal{I}, \mathcal{I} \diamond U) \in \Big\| \Big(\mathcal{A} \wedge \bigwedge_{\substack{A \in \mathsf{Con} \\ a \in \mathsf{Ind}}} (\neg(a:A^{+}) \wedge \neg(a:A^{-}))\Big)\,?\,;\; \mathsf{dynamicRepair}(\mathcal{A})\,;\mathsf{makeFalse}^{\pm} \Big\|
$$

Going back to Example 3 this time we can witness how, taking $\mathcal{I}$ to be a model of $\{\mathsf{John} : \mathsf{Born} \sqcap \neg\mathsf{Alive} \sqcap \neg\mathsf{Dead}\}$, only the set $\{+\mathsf{Alive}(\mathsf{John})\}$ satisfies the condition of Theorem 4 whereas the set $\{+\mathsf{Alive}(\mathsf{John}), +\mathsf{Dead}(\mathsf{John})\}$ does not, for any $\mathcal{I}$.

5 Discussion and Conclusion

We have seen how different repairing methods based on preferred update actions of the database literature translate into Description Logics. We have defined a *repair* to be a set of update actions U such that, when applied to an ABox, the outcome is a repaired ABox that follows the active axioms of—and is consistent with—the active TBox. The way to find such a U is the following: starting with a model $\mathcal{I}$ of an ABox $\mathcal{A}$, we can extract a founded or dynamic repair U of $\mathcal{A}$ by finding an $\mathcal{I}'$ such that $(\mathcal{I}, \mathcal{I}') \in \| \mathsf{foundedRepair}(\mathcal{A}) \|$ or $(\mathcal{I}, \mathcal{I}') \in \| \mathsf{dynamicRepair}(\mathcal{A}) \|$, respectively, and setting $U = \{+A(a) : (a : A^+)^{\mathcal{I}'} = \Delta^{\mathcal{I}'}\} \cup \{-A(a) : (a : A^-)^{\mathcal{I}'} = \Delta^{\mathcal{I}'}\}$. The dynamic logic framework on which we rely is useful for not only representing the various repairs but also for constructing them. Constructing the repaired ABox then would amount to: either extract $\mathcal{A} \diamond U$ by reducing the formula $\langle \pi_U \rangle^{\mathsf{c}} \mathcal{A}$ as in Example 1 or immediately apply the update U in the initial $\mathcal{ALCO}$ ABox and characterize $\mathcal{A} \diamond U$ via a semantic update of $\mathcal{A}$ by U.

A slightly different route, more in line with Description Logic, would be to define a *repair* to be any repaired ABox $\mathcal{A}'$ and, given an ABox $\mathcal{A}$, evaluate if $\mathcal{A}'$ is indeed a repair of $\mathcal{A}$ by checking if the formula $\mathcal{A} \wedge \langle \mathsf{repair} \rangle \mathcal{A}'$ is satisfiable, where repair is any of the repair programs defined in the previous section and used in Theorems 3 and 4. Of course we then would have to *guess* such a repair $\mathcal{A}'$, but the satisfiability check could also be used for assertions instead of whole ABoxes, in order to check if something more specific follows from repairing the initial ABox under the active axioms of an active TBox.

Deciding the existence of founded and dynamic repairs also amounts to satisfiability checking. More specifically, taking repair to be once again any of the repair programs, we can decide the existence of each one by checking whether the formula $\langle \mathsf{repair} \rangle^{\mathsf{c}} \mathcal{A}$ is satisfiable, where $\mathcal{A}$ is the initial ABox. Indeed, any interpretation that satisfies the formula $\langle \mathsf{repair} \rangle^{\mathsf{c}} \mathcal{A}$ will be a model of at least one repaired ABox. Our approach also comes close to the work presented in [1] where dynamic problems on graph-structured data are reduced to static DL reasoning. Apart from the difference in the Dynamic Logic framework presented here and the fact that we are not restricted to finite interpretations, the main differentiation between the two works is the fact that we introduce and work mainly with active TBoxes as well as adapt the various repairs found in the database literature of active integrity constraints to the DL setting.

A limitation of our approach comes from the boolean nature of the active concept inclusions and the fact that complex concepts cannot be paired with a preferred update action in the active part of a concept inclusion. For instance, it is easy to see that according to Definition 3, the TBox $\mathcal{T} = \{\exists \mathsf{hasFather}.\mathsf{Female} \sqsubseteq \bot\}$ can be only extended to the active TBox $\mathsf{a}\mathcal{T} = \{\langle \exists \mathsf{hasFather}.\mathsf{Female} \sqsubseteq \bot, \emptyset \rangle\}$. At first sight, going through roles to apply the atomic programs or add/remove roles between individuals seems to make the logic undecidable, mainly because of the interplay between these more powerful programs and the Kleene star, but the fine details are left for future work.

Summing up, we have taken a semantic approach to repairing ABoxes with regard to active TBoxes, the latter being an extension of regular TBoxes with update actions. To that end we have exploited a dynamic logic framework on which various repairing procedures are introduced and discussed.

References

1. Ahmetaj, S., Calvanese, D., Ortiz, M., Simkus, M.: Managing change in graph-structured data using description logics. ACM Trans. Comput. Log. **18**(4), 27:1–27:35 (2017), http://doi.acm.org/10.1145/3143803
2. Baader, F., Calvanese, D., McGuinness, D.L., Nardi, D., Patel-Schneider, P.F. (eds.): The Description Logic Handbook: Theory, Implementation, and Applications. Cambridge University Press, New York, NY, USA (2003)
3. Balbiani, P., Herzig, A., Troquard, N.: Dynamic logic of propositional assignments: a well-behaved variant of PDL. In: Kupferman, O. (ed.) Logic in Computer Science (LICS). IEEE (2013)
4. Bertossi, L.: Database Repairing and Consistent Query Answering. Morgan & Claypool Publishers (2011)
5. Bienvenu, M., Bourgaux, C., Goasdoué, F.: Querying inconsistent description logic knowledge bases under preferred repair semantics. In: Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 -31, 2014, Québec City, Québec, Canada. pp. 996–1002 (2014), http://www.aaai.org/ocs/index.php/AAAI/AAAI14/paper/view/8231
6. Bienvenu, M., Bourgaux, C., Goasdoué, F.: Query-driven repairing of inconsistent dl-lite knowledge bases. In: Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016. pp. 957–964 (2016), http://www.ijcai.org/Abstract/16/140
7. Caroprese, L., Greco, S., Zumpano, E.: Active integrity constraints for database consistency maintenance. IEEE Trans. Knowl. Data Eng. **21**(7), 1042–1058 (2009)
8. Caroprese, L., Truszczynski, M.: Active integrity constraints and revision programming. TPLP **11**(6), 905–952 (2011)
9. Ceri, S., Fraternali, P., Paraboschi, S., Tanca, L.: Automatic generation of production rules for integrity maintenance. ACM Trans. Database Syst. **19**(3), 367–422 (Sep 1994), http://doi.acm.org/10.1145/185827.185828
10. Chomicki, J., Marcinkowski, J.: Minimal-change integrity maintenance using tuple deletions. Inf. Comput. **197**(1-2), 90–121 (Feb 2005), http://dx.doi.org/10.1016/j.ic.2004.04.007
11. Ditmarsch, H.v., van der Hoek, W., Kooi, B.: Dynamic Epistemic Logic. Springer Publishing Company, Incorporated, 1st edn. (2007)
12. Feuillade, G., Herzig, A.: A dynamic view of active integrity constraints. In: Fermé, E., Leite, J. (eds.) Logics in Artificial Intelligence - 14th European Conference, JELIA 2014, Funchal, Madeira, Portugal, September 24-26, 2014. Proceedings. Lecture Notes in Computer Science, vol. 8761, pp. 486–499. Springer (2014), https://doi.org/10.1007/978-3-319-11558-0
13. Flesca, S., Greco, S., Zumpano, E.: Active integrity constraints. In: Moggi, E., Warren, D.S. (eds.) PPDP. pp. 98–107. ACM (2004)
14. Harel, D.: Dynamic logic. In: Gabbay, D.M., Günthner, F. (eds.) Handbook of Philosophical Logic, vol. II, pp. 497–604. D. Reidel, Dordrecht (1984)
15. Harel, D., Kozen, D., Tiuryn, J.: Dynamic Logic. MIT Press (2000)
16. Horrocks, I., Kutz, O., Sattler, U.: The even more irresistible SROIQ. In: Doherty, P., Mylopoulos, J., Welty, C.A. (eds.) Proceedings, Tenth International Conference on Principles of Knowledge Representation and Reasoning, Lake District of the United Kingdom, June 2-5, 2006. pp. 57–67. AAAI Press (2006), http://www.aaai.org/Library/KR/2006/kr06-009.php

17. 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), http://dx.doi.org/10.1007/978-3-642-15918-3_9
18. Liu, H., Lutz, C., Milicic, M., Wolter, F.: Foundations of instance level updates in expressive description logics. Artificial Intelligence **175**(18), 2170–2197 (2011)
19. Motik, B., Horrocks, I., Sattler, U.: Bridging the gap between OWL and relational databases. J. Web Sem. **7**(2), 74–89 (2009), https://doi.org/10.1016/j.websem.2009.02.001
20. Rantsoudis, C., Feuillade, G., Herzig, A.: Repairing ABoxes through active integrity constraints. In: Artale, A., Glimm, B., Kontchakov, R. (eds.) Proceedings of the 30th International Workshop on Description Logics, Montpellier, France, July 18-21, 2017. CEUR Workshop Proceedings, vol. 1879. CEUR-WS.org (2017), http://ceur-ws.org/Vol-1879/paper41.pdf
21. Tao, J., Sirin, E., Bao, J., McGuinness, D.L.: Integrity constraints in OWL. In: Fox, M., Poole, D. (eds.) Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2010, Atlanta, Georgia, USA, July 11-15, 2010. AAAI Press (2010), http://www.aaai.org/ocs/index.php/AAAI/AAAI10/paper/view/1931