A Dynamic Logic Account of Active Integrity Constraints

Guillaume Feuillade, Andreas Herzig, Christos Rantsoudis

DOI 10.3233/FI-2019-1843

A Dynamic Logic Account of Active Integrity Constraints

Guillaume Feuillade*

Institut de Recherche en Informatique de Toulouse (IRIT)

Université Toulouse, France

Guillaume.Feuillade@irit.fr

Andreas Herzig

Institut de Recherche en Informatique de Toulouse (IRIT)

CNRS, France

Andreas.Herzig@irit.fr

Christos Rantsoudis

Institut de Recherche en Informatique de Toulouse (IRIT)

Université Toulouse, France

Christos.Rantsoudis@irit.fr

Abstract. Active integrity constraints have been introduced in the database community as a way to restore integrity based on a set of preferred update actions. We view active integrity constraints as dynamic logic programs and show how several semantics of database repair that were proposed in the literature can be characterised in Dynamic Logic of Propositional Assignments DL-PA. We moreover propose a new definition of repair which makes use of the programs of Dynamic Logic to provide repair solutions based on an iterating procedure. After an analysis of their properties and a comparison to the previous approaches, we provide complexity results for the problem of existence of these new repairs. Furthermore, an extension on databases with history is explored and the behavior of the various repairs is adjusted to work in this setting. For all these definitions we provide DL-PA counterparts of reasoning and decision problems, such as the existence of a repair or the existence of a unique repair.

Keywords: active integrity constraints, dynamic logic, propositional assignments

*Address for correspondence: IRIT, Univ. Paul Sabatier, 118 route de Narbonne, F-31062 Toulouse Cedex 9, France.

1. Introduction

One of the most important (but notoriously difficult) issues in the database and AI literature is the problem of updating a database under a set of *integrity constraints*. The latter are usually expressed by logical formulas and their role is to impose conditions that every state of the database must satisfy. In the course of database maintenance several changes are applied to the databases and checking whether these constraints are no longer satisfied is of the highest priority. When a database fails to satisfy the integrity constraints, it has to be repaired in order to *restore integrity*. Given a database, the procedure of repairing and restoring its consistency with respect to a set of integrity constraints has been extensively studied in the last decades [1, 2, 3]. Apart from the problem of finding a repair though, a significant part of the research has focused on the distinction between which types of repairs are more suitable for integrity maintenance, given the fact that the number of all possible repairs can be remarkably large. A number of different types of repairs came into fruition, with one of the most prevalent being the ones based on the *minimality of change* principle [4, 5, 6]. Despite this, however, the need to have 'more informed' ways of maintaining database integrity arose.

In light of this, *active integrity constraints* were proposed as an extension of integrity constraints (or *static* constraints) with update actions, each one suggesting the preferred update method when an inconsistency arises between the database and a constraint [7, 8, 9, 10]. For example, the integrity constraint $(\forall X)[A(X) \rightarrow B(X)]$ can be extended to the active constraint $(\forall X)[A(X) \wedge \neg B(X) \rightarrow -A(X)]$, indicating that any violation of the formula $A(X) \rightarrow B(X)$ should be reacted to by making $A(X)$ false (instead of making $B(X)$ true). In the propositional case, an active integrity constraint can be represented as a couple $r = \langle \mathsf{C}(r), \mathsf{R}(r) \rangle$ where $\mathsf{C}(r)$ is a Boolean formula (called the static part of r and denoting a static constraint) and $\mathsf{R}(r)$ is a set of update actions, each of which is of the form $+p$ or $-p$ for some atomic formula p. The idea is that (1) when $\mathsf{C}(r)$ is false then the constraint r is violated and (2) a violated constraint can be repaired by performing one or more of the update actions in $\mathsf{R}(r)$. The two most prevalent types of repairs w.r.t. a set of active integrity constraints are the *founded* and the *justified* repairs. Note that while these approaches can greatly reduce the number of possible repairs, the choice between different repair procedures can still lead to different sets of possible repairs or even no repairs at all (for example when $\mathsf{R}(r)$ is the empty set).

Now, although the setting of active integrity constraints is well established and thoroughly explored, there exist cases where even founded and justified repairs cannot provide a satisfactory solution. Recent approaches have also exhibited that the debate about the 'right' semantics is not yet entirely settled [11]. In our opinion, a more *dynamic* procedure could provide solutions to inconsistencies that arise between a database and a set of active constraints that build upon and extend one another. We showcase such an example in the following scenario. Consider a company with two departments, D_1 and D_2, where temporary employees (e.g. interns) are assigned to D_1 and permanent employees work at D_2 (so D_1 has no permanent members and D_2 has no temporary members). Every employee must be in a department, i.e., we have the integrity constraint $D_1 \vee D_2$. Furthermore, consider that every person working on D_2 has previously worked on D_1, i.e., D_1 is like a 'training ground' for becoming a permanent member. Consider now a database for permanent members which keeps track of their status. Every employee should be declared in the database as starting to work on D_1 and we can express this by the active constraint $\langle D_1 \vee D_2, +D_1 \rangle$ which declares that if $D_1 \vee D_2$

is violated and an employee is not assigned to any department then s/he should be assigned to D_1. Furthermore, since the database is for permanent employees, if an employee is assigned only to D_1 then this should be rectified by assigning them to D_2 as well. This is expressed by the active constraint $\langle \neg D_1 \vee D_2, +D_2 \rangle$. Finally, consider that the database loses track of an employee, i.e., a permanent employee is declared as working at neither D_1 nor D_2. How would the status of this employee, which is inconsistent w.r.t. the active constraints, be repaired according to the available repair procedures? Whereas founded and justified repairs cannot provide a solution to this problem (see Section 2.3.2 for more details), a dynamic procedure which would check each active constraint at a time and apply an update action before repeating seems able to do so. Indeed, as we will witness in Section 5.2, the set $\{+D_1, +D_2\}$ will be the only solution using such a dynamic procedure. Note also that a repair which conforms to the minimal change principle would suggest that $\{+D_2\}$ should be the only repair. While this is indeed the only minimal solution, it can be argued that it should not be the case that this employee was assigned to D_2 directly, without having worked on D_1 first.

In this paper we examine active integrity constraints in the framework of dynamic logic and argue that they can be viewed as particular programs: the sequential composition of the test of $\mathsf{C}(r)$ and the nondeterministic choice of an action in $\mathsf{R}(r)$. Repairing a database can then be done by means of a complex program that combines active integrity constraints. We use a simple yet powerful dialect of dynamic logic: Dynamic Logic of Propositional Assignments, abbreviated DL-PA [12, 13, 14]. The latter is a simple instantiation of Propositional Dynamic Logic PDL [15, 16]. Instead of PDL's abstract atomic programs, the atomic programs of DL-PA are update actions: assignments of propositional variables to either true or false, written $p{\leftarrow}\top$ and $p{\leftarrow}\bot$. Just as in PDL, these atomic programs can be combined by means of program operators: sequential and nondeterministic composition, finite iteration and test. The language of DL-PA has not only programs, but also formulas. While DL-PA programs describe the evolution of the world, DL-PA formulas describe the state of the world. In particular, formulas of the form $\langle \pi \rangle \varphi$ express that φ is true after *some* possible execution of π, and $[\pi]\varphi$ expresses that φ is true after *every* possible execution of π. The models of DL-PA are considerably simpler than PDL's Kripke models: valuations of classical propositional logic are enough. The assignment $p{\leftarrow}\top$ inserts p, while the assignment $p{\leftarrow}\bot$ deletes p. Apart from being simple yet quite expressive, its biggest computational advantage over PDL comes in the form of the elimination of the Kleene star: it is shown in [12, 13] that every DL-PA formula can be reduced to an equivalent Boolean formula (something that is not possible in PDL). This is an important attribute and a very useful tool, as it will allow us to construct repaired databases syntactically. The most significant advantage of using a dynamic logic framework though is the fact that we can easily study extensions (like the history-based repairs of Section 7) that are expressible in the language by simply extending the formulas in the appropriate ways. Last but not least, just as in [9, 10, 11] we only consider ground constraints in the current paper, i.e., we work with a propositional language.

The paper is organized as follows. In Section 2 we provide a thorough background on all the aforementioned points: the logic DL-PA, static and active constraints, as well as the associated repairs for both (weak repairs, PMA repairs, founded and justified repairs). In Section 3 we provide an embedding of the associated repairs of static constraints (weak repairs and PMA repairs) into DL-PA. In Section 4 we do the same for the associated repairs of active constraints (founded and justified repairs). Section 5 comprises the main contribution of the paper: we propose some new definitions of

repairs in terms of **while** programs and compare them with the aforementioned founded and justified repairs in various ways. Their computational complexity is investigated in Section 6. In Section 7 we push the envelope of the active constraint paradigm and examine databases with history as well as how the various repairs are integrated in their framework. Finally, Section 8 concludes with some examples of related reasoning problems and a brief discussion on future work.

This paper extends [17] by the analysis of justified repairs, a thorough discussion on the dynamic repairs introduced as well as how they compare with the available repair procedures from the literature, a complexity analysis and a look into databases with history.

2. Background

We consider languages of classical propositional logic, built from a countable set of propositional variables (alias atomic formulas) $\mathbb{P} = \{p, q, \ldots\}$. *Boolean formulas* are built from $\mathbb{P}$ by means of the Boolean operators $\top$, $\bot$, $\neg$, and $\vee$ and are denoted by A, B, etc. The other Boolean connectives $\wedge$, $\rightarrow$, and $\leftrightarrow$ are abbreviated in the usual way. A *literal* is an element of $\mathbb{P}$ or the negation of an element of $\mathbb{P}$ and a *clause* is a disjunction of literals.

Valuations are subsets of $\mathbb{P}$ and are denoted by V, V_1, V_2, etc. The set of all valuations is therefore $\mathbb{V} = 2^{\mathbb{P}}$. It will sometimes be convenient to write $V(p) = \top$ instead of $p \in V$ and $V(p) = \bot$ instead of $p \notin V$. A valuation determines the truth value of every Boolean formula. The set of valuations where A is true is denoted by $\|A\|$. We sometimes write $V \models A$ when $V \in \|A\|$. In the context of the present article a valuation is called a *database*.

An *update action* is of the form $p{\leftarrow}\top$ and $p{\leftarrow}\bot$, for $p \in \mathbb{P}$. The former is the insertion of p and the latter is the deletion of p. We denote the set of all update actions by $\mathbb{U}$. We sometimes use X as a metavariable for $\top$ and $\bot$ and write $p{\leftarrow}\mathsf{X}$. For subsets P of $\mathbb{P}$ it will be convenient to write $P{\leftarrow}\top$ to denote the set of update actions $\{p{\leftarrow}\top \ : \ p \in P\}$, and likewise for $P{\leftarrow}\bot$. A set of update actions $U \subseteq \mathbb{U}$ is *consistent* if it does not contain both $p{\leftarrow}\top$ and $p{\leftarrow}\bot$, for some p. The *update* of a valuation V by a set of update actions U is defined as:

$$V \diamond U = (V \setminus \{p \ : \ p{\leftarrow}\bot \in U\}) \cup \{p \ : \ p{\leftarrow}\top \in U\}$$

So all the deletions are applied first, followed by the application of all insertions. We could as well have chosen another order of application. When U is consistent then all of them lead to the same result. In particular:

Proposition 2.1. Let $\{\alpha_1, \ldots, \alpha_n\}$ be a consistent set of update actions. Let $\langle k_1 \ldots k_n \rangle$ be some permutation of $\langle 1 \ldots n \rangle$. Then $V \diamond \{\alpha_1, \ldots, \alpha_n\} = (\ldots(V \diamond \{\alpha_{k_1}\})\ldots) \diamond \{\alpha_{k_n}\}$.

In the following subsections, we recall Dynamic Logic of Propositional Assignments and the definitions of the various repair procedures w.r.t. static and active integrity constraints.

2.1. Dynamic Logic of Propositional Assignments

The first studies of assignments in the context of dynamic logic are due, among others, to Tiomkin and Makowski and van Eijck [18, 19]. Dynamic Logic of Propositional Assignments DL-PA was

introduced in [12] and was further studied in [13, 14][1]. Evidence for its widespread applicability was provided in several recent publications, including belief update and belief revision, argumentation, planning and reasoning about knowledge [20, 21, 22, 23, 24, 25]. We briefly recall syntax and semantics of DL-PA.

2.1.1. Language

The language of DL-PA is defined by the following grammar:

$$\varphi ::= p \mid \top \mid \bot \mid \neg\varphi \mid \varphi \vee \varphi \mid \langle\pi\rangle\varphi$$
$$\pi ::= \alpha \mid \pi;\pi \mid \pi \cup \pi \mid \pi^* \mid \pi^- \mid \varphi?$$

where φ are the formulas of DL-PA, π are the programs of DL-PA, p ranges over the set of atomic formulas $\mathbb{P}$ and α ranges over the set of update actions $\mathbb{U}$. In DL-PA, update actions are singletons and are called *atomic assignments*. The operators of sequential composition ";", nondeterministic composition "$\cup$", finite iteration (the so-called Kleene star) "$(.)^*$" and test "$(.)?$" are the familiar operators of PDL. The operator "$(.)^-$" is the converse operator. The formula $\langle\pi\rangle\varphi$ is read "there is an execution of π after which φ is true". So e.g. $\langle p{\leftarrow}\bot^-\rangle(p \wedge q)$ expresses that $p \wedge q$ is true after some execution of $p{\leftarrow}\bot^-$, i.e., $p \wedge q$ was true before p was set to false. The star-free fragment of DL-PA is the subset of the language made up of formulas without the Kleene star.

We define $\mathbb{P}_\varphi$ to be the set of variables from $\mathbb{P}$ occurring in formula φ, and we define $\mathbb{P}_\pi$ to be the set of variables from $\mathbb{P}$ occurring in program π. For example, $\mathbb{P}_{p\leftarrow q\cup p\leftarrow\neg q} = \{p, q\} = \mathbb{P}_{\langle p\leftarrow\bot\rangle q}$.

Several program abbreviations are familiar from PDL. First, **skip** abbreviates $\top?$ and **fail** abbreviates $\bot?$. Second, **if** φ **then** π_1 **else** π_2 is expressed by $(\varphi?;\pi_1)\cup(\neg\varphi?;\pi_2)$. Third, the loop **while** φ **do** π is expressed by $(\varphi?;\pi)^*;\neg\varphi?$. The nondeterministic composition $\bigcup_{\alpha\in U}\alpha$ equals **fail** when U is empty. We also define π^+ to be $\pi;\pi^*$. Let us moreover introduce assignments of literals to variables by means of the following two abbreviations:

$$p{\leftarrow}q = \textbf{if } q \textbf{ then } p{\leftarrow}\top \textbf{ else } p{\leftarrow}\bot \qquad\qquad p{\leftarrow}\neg q = \textbf{if } q \textbf{ then } p{\leftarrow}\bot \textbf{ else } p{\leftarrow}\top$$

The former assigns to p the truth value of q, while the latter assigns to p the truth value of $\neg q$. In particular, the program $p{\leftarrow}\neg p$ flips the truth value of p. Note that both abbreviations have constant length, namely 14. Finally and as usual in modal logic, $[\pi]\varphi$ abbreviates $\neg\langle\pi\rangle\neg\varphi$.

2.1.2. Semantics

DL-PA programs are interpreted as relations between valuations. The atomic programs α update valuations in the usual way (see the beginning of the section) and complex programs are interpreted just as in PDL by mutual recursion. Table 1 gives the interpretation of formulas and programs, where $\circ$ is relation composition and $(.)^{-1}$ is relation inverse.

A formula φ is DL-PA *valid* iff $\|\varphi\| = 2^{\mathbb{P}} = \mathbb{V}$. It is DL-PA *satisfiable* iff $\|\varphi\| \neq \emptyset$. For example, the formulas $\langle p{\leftarrow}\bot\rangle\top$, $\langle p{\leftarrow}\top\rangle\varphi \leftrightarrow \neg\langle p{\leftarrow}\top\rangle\neg\varphi$, $\langle p{\leftarrow}\top\rangle p$ and $\langle p{\leftarrow}\bot\rangle\neg p$ are all valid. Observe that

[1]Note that [14] fixes some complexity results that were erroneously established in [13].

Table 1. Interpretation of formulas and programs.

$\|\|p\|\|$	$= \{V : p \in V\}$	$\|\|\alpha\|\|$	$= \{\langle V_1, V_2\rangle : V_2 = V_1 \diamond \{\alpha\}\}$
$\|\|\top\|\|$	$= \mathbb{V} = 2^{\mathbb{P}}$	$\|\|\pi;\pi'\|\|$	$= \|\|\pi\|\| \circ \|\|\pi'\|\|$
$\|\|\bot\|\|$	$= \emptyset$	$\|\|\pi \cup \pi'\|\|$	$= \|\|\pi\|\| \cup \|\|\pi'\|\|$
$\|\|\neg\varphi\|\|$	$= 2^{\mathbb{P}} \setminus \|\|\varphi\|\|$	$\|\|\pi^*\|\|$	$= (\|\|\pi\|\|)^*$
$\|\|\varphi \vee \psi\|\|$	$= \|\|\varphi\|\| \cup \|\|\psi\|\|$	$\|\|\pi^-\|\|$	$= (\|\|\pi\|\|)^{-1}$
$\|\|\langle\pi\rangle\varphi\|\|$	$= \{V : \exists V_1$ s.t. $\langle V, V_1\rangle \in \|\|\pi\|\|$ and $V_1 \in \|\|\varphi\|\|\}$	$\|\|\varphi?\|\|$	$= \{\langle V, V\rangle : V \in \|\|\varphi\|\|\}$

if p does not occur in φ then $\varphi \to \langle p{\leftarrow}\top\rangle\varphi$ and $\varphi \to \langle p{\leftarrow}\bot\rangle\varphi$ are valid. This is due to the following semantical property that is instrumental in the proof of several results in the rest of the paper.

Proposition 2.2. Let P be a subset of $\mathbb{P}$. Suppose that $\mathbb{P}_\varphi \cap P = \emptyset$, i.e., none of the variables of P occurs in φ. Then $V \cup P \in \|\|\varphi\|\|$ iff $V \setminus P \in \|\|\varphi\|\|$.

As already mentioned in the introductory section, a distinguishing feature of DL-PA is that its dynamic operators can be eliminated (which is not possible in PDL). Just as for QBF, the resulting formula may be exponentially longer than the original formula.

Theorem 2.3. ([13], Theorem 1)
For every DL-PA formula there is an equivalent Boolean formula.

For example, the DL-PA formula $\langle p{\leftarrow}\bot\rangle(\neg p \wedge \neg q)$ is equivalent to the formula $\langle p{\leftarrow}\bot\rangle\neg p \wedge \langle p{\leftarrow}\bot\rangle\neg q$, which is in turn equivalent to $\top \wedge \neg q$. Therefore, the DL-PA formula $\langle p{\leftarrow}\bot\rangle(\neg p \wedge \neg q)$ reduces to the Boolean formula $\neg q$.

Every assignment sequence $\alpha_1;\ldots;\alpha_n$ is a deterministic program that is always executable: for a given V, there is exactly one V' such that $\langle V, V'\rangle \in \|\|\alpha_1;\ldots;\alpha_n\|\|$. Moreover, the order of the α_i in a sequential composition is irrelevant when the set of update actions $\{\alpha_1,\ldots,\alpha_n\}$ is consistent. The following can be viewed as a reformulation of Proposition 2.1 in terms of the DL-PA operator of sequential composition.

Proposition 2.4. Let $\{\alpha_1,\ldots,\alpha_n\}$ be a consistent set of update actions. Let $\langle k_1 \ldots k_n\rangle$ be some permutation of $\langle 1 \ldots n\rangle$. Then $V \diamond \{\alpha_1,\ldots,\alpha_n\}$ equals the unique V' such that $\langle V, V'\rangle \in \|\|\alpha_{k_1};\ldots;\alpha_{k_n}\|\|$.

This entitles us to use sets of consistent update actions as programs: one may suppose that this stands for a sequential composition in some predefined order (based e.g. on the enumeration of the set of propositional variables).

2.2. Static constraints and the associated repairs

In this subsection we consider the classical notion of database integrity that is defined in terms of *static integrity constraints*, or *static constraints* for short. In our propositional language they are nothing but

Boolean formulas. Two ways of repairing databases based on such constraints can be found in the literature [9]. Both consist in first finding an appropriate set of update actions U and then building the update $V \diamond U$ of V by U as defined in the beginning of the section. We relate them to well-known operations in belief revision and update [26], which allows us to reuse their embeddings into DL-PA [20].

2.2.1. Weak repairs and drastic updates

Let V be a database, U a set of update actions and C a set of static constraints, i.e., a set of Boolean formulas. In the rest of the paper we will only consider finite sets of static constraints. We say that U is *relevant* w.r.t. V iff $p{\leftarrow}\top \in U$ implies $p \notin V$ and $p{\leftarrow}\bot \in U$ implies $p \in V$. The definition of a *weak repair* immediately follows.

Definition 2.5. Let V be a database and let C be a set of static constraints. A weak repair of V achieving C is a consistent set of update actions $U \subseteq \mathbb{U}$ such that $V \diamond U \models \bigwedge C$ and U is relevant w.r.t. V.

The next example illustrates that weak repairs are indeed very weak.

Example 2.6. Let $V = \emptyset$ and $C = \{p \vee q\}$. The weak repairs of V achieving C are all those subsets of the set of positive update actions $\{r{\leftarrow}\top \ : \ r \in \mathbb{P}\}$ that contain either $p{\leftarrow}\top$, or $q{\leftarrow}\top$, or both.

As the following result shows, if we consider what is true in all possible weak repairs then we obtain what is called a *drastic update* in the literature on belief revision and update.[2]

Proposition 2.7. Let V be a database and let C be a set of static constraints. Then:

$$\{V \diamond U \ : \ U \text{ is a weak repair of } V \text{ achieving } C\} = \Big\| \bigwedge C \Big\|$$

Note that a weak repair may contain assignments of variables that do not occur in C. To witness, in the above example $\{p{\leftarrow}\top, q{\leftarrow}\top, r{\leftarrow}\top\}$ is a weak repair of V achieving C. To remedy this we consider every weak repair U from now on to be such that if $p{\leftarrow}\top$ or $p{\leftarrow}\bot$ occurs in U then $p \in \mathbb{P}_C$, where $\mathbb{P}_C$ is the set of variables from $\mathbb{P}$ occurring in C. This corresponds to a very basic update semantics that is sometimes called Winslett's standard semantics [4, 5].

2.2.2. Repairs *tout court* and their relation to Winslett's PMA

We now present the definition of a *repair* which, as we already mentioned, uses the principle of minimal change to produce repair solutions that are considered more practical in contrast to the more general weak repairs.

Definition 2.8. Let V be a database and let C be a set of static constraints. A repair of V achieving C is a weak repair of V achieving C that is minimal w.r.t. set inclusion: there is no weak repair of V achieving C that is strictly contained in it.

[2] It is actually also a drastic revision because V is a complete database and update and revision coincide in that case [27].

The next example is a follow-up to Example 2.6.

Example 2.9. Let $V = \emptyset$ and $C = \{p \vee q\}$. There are exactly two repairs of V achieving C, viz. $\{p \leftarrow \top\}$ and $\{q \leftarrow \top\}$.

We are now going to relate repairs to Winslett's possible models approach PMA [28, 4]. Remember that the update of a database V by a Boolean formula A according to the PMA is the set of valuations V' such that $V' \models A$ and such that the symmetric difference between V and V' is minimal w.r.t. set inclusion. Formally, the symmetric difference is defined as $\mathsf{D}(V, V') = \{p \ : \ V(p) \neq V'(p)\}$ and the PMA update of V by A is the set:

$$\{V' \ : \ V' \models A \text{ and there is no } V'' \in \|A\| \text{ such that } \mathsf{D}(V, V'') \subset \mathsf{D}(V, V')\}$$

For example, the PMA update of $\emptyset$ by $p \vee q$ is $\{\{p\}, \{q\}\}$ and the PMA update of $\emptyset$ by $(p \wedge q) \vee r$ is $\{\{p, q\}, \{r\}\}$.

Proposition 2.10. Let V be a database and let C be a set of static constraints. Then:

$$\{V \diamond U \ : \ U \text{ is a repair of } V \text{ by } C\} \text{ is the PMA update of } V \text{ by } \Big(\bigwedge C\Big)$$

The above result justifies the term *PMA repair* that we are going to employ henceforth (because the mere term 'repair' might lead to confusions).

2.3. Active constraints and the associated repairs

Active integrity constraints were proposed more than ten years ago [7] and various ways of repairing a database V by such constraints were studied in the literature. We refer to [9] for an overview. Just as for static constraints, all definitions are based on the notion of a *repair set*: an appropriate set of update actions U such that $V \diamond U$ no longer violates the integrity constraints. $V \diamond U$ is once again the result of updating V by U as defined in the beginning of the section and is called the *repaired database*.

In the present subsection we recall syntax and semantics of the two main routes that have been explored in the literature.

2.3.1. Active integrity constraints

An *active integrity constraint* (or *active constraint* for short) combines a static integrity constraint with a set of preferred repair actions.

Definition 2.11. An active constraint is a couple $r = \langle \mathsf{C}(r), \mathsf{R}(r) \rangle$, where $\mathsf{C}(r)$ is a Boolean formula and $\mathsf{R}(r)$ is a finite set of update actions that is consistent.

As before, $\mathsf{C}(r)$ is a static integrity constraint that is violated when $\mathsf{C}(r)$ is false. If this is the case and $\mathsf{R}(r) \neq \emptyset$ then r is *applicable* and $\mathsf{R}(r)$ indicates how to get rid of the violation and restore integrity. The elements of $\mathsf{R}(r)$ are viewed as *permitted* update actions: when $\mathsf{C}(r)$ is violated then each of the actions in $\mathsf{R}(r)$ gets a 'license to update'. This is a rather imprecise description of the job

the update actions in $\mathsf{R}(r)$ are expected to do and in the literature various semantics were proposed. One of the most prominent of them are *founded* repairs which make use of the *foundedness* condition in order to apply the correct update actions, while *justified* repairs build upon and refine this condition in order to tackle the so-called *circularity of support* issue that can be witnessed by founded repairs.

We say that an active constraint $r = \langle \mathsf{C}(r), \mathsf{R}(r) \rangle$ is *standard* if $\mathsf{C}(r)$ is a clause and each update action in $\mathsf{R}(r)$ makes one of the literals of $\mathsf{C}(r)$ true: if $p \leftarrow \top \in \mathsf{R}(r)$ then p has to be one of the literals of $\mathsf{C}(r)$ and if $p \leftarrow \bot \in \mathsf{R}(r)$ then $\neg p$ has to be one of the literals of $\mathsf{C}(r)$.

Remark 2.12. The definition in the literature differs in several respects from ours here. First, $\mathsf{C}(r)$ is usually not viewed as a static integrity constraint but as the negation of a static integrity constraint: r is violated when the first component of r is true. Second, active constraints are denoted by $\mathsf{C}(r) \rightarrow \mathsf{R}(r)$, which makes them look like formulas. However, "$\rightarrow$" is different from material implication as the right hand side of the implication is not a formula but a set of programs. So their semantics remains to be given: in the literature this is typically done by means of disjunctive logic programs under a non-monotonic semantics. Third, all active constraints have to be standard.

We denote finite sets of active constraints by η, η_1, etc. The set of static integrity constraints associated with η is defined as $\mathsf{C}(\eta) = \{\mathsf{C}(r) : r \in \eta\}$. Furthermore, the *size* of $\mathsf{C}(\eta)$, denoted by $|\,\mathsf{C}(\eta)\,|$, is the sum of the lengths of each $\mathsf{C}(r)$ for all $r \in \eta$, i.e., $|\,\mathsf{C}(\eta)\,| = \sum_{r \in \eta} |\,\mathsf{C}(r)\,|$, where $|\,\mathsf{C}(r)\,|$ is the length of the Boolean formula $\mathsf{C}(r)$ as defined in propositional logic.

It remains to give a semantics to active constraints. In the rest of this subsection we discuss the two main existing directions, viz. founded and justified repairs. We later propose a new one in Section 5 using the programs of DL-PA.

2.3.2. Founded weak repairs and founded repairs

In the literature, founded repairs are considered to be a basic semantics of active constraints. They provide a basis for further refinements. The key notion they rely on is the *foundedness* condition.[3]

Definition 2.13. Let V be a database and let η be a set of active constraints. A consistent set of update actions U is founded w.r.t. V and η if for every $\alpha \in U$ there is an $r \in \eta$ such that:

(a) $\alpha \in \mathsf{R}(r)$

(b) $V \diamond U \models \mathsf{C}(r)$

(c) $V \diamond (U \setminus \{\alpha\}) \not\models \mathsf{C}(r)$

Given this condition, the definitions of a *founded weak repair* and a *founded repair* immediately follow.

Definition 2.14. Let V be a database and let η be a set of active constraints. A set of update actions U is a founded weak repair of V by η if U is a weak repair of V achieving $\mathsf{C}(\eta)$ and U is founded w.r.t. V and η. Moreover, if U is also a PMA repair of V achieving $\mathsf{C}(\eta)$, then U is a founded repair of V by η.

[3]We have reformulated the original definition so that it applies to our more general definition of active constraints. Both are equivalent as far as standard active constraints are concerned.

The following simple example showcases this definition.

Example 2.15. Let $V = \emptyset$ and $\eta = \{\langle p, \{p \leftarrow \top\}\rangle, \langle p \vee q, \{q \leftarrow \top\}\rangle\}$. The set $\{p \leftarrow \top\}$ is the only founded weak repair of V by η. Indeed, the second update action in $\{p \leftarrow \top, q \leftarrow \top\}$ cannot be founded on the second active constraint of η. It is also the only founded repair.

There are sets of active constraints for which there is no founded repair, although there is a founded weak repair [9, Example 2]. The next example, which is adapted from the example of the introductory section, shows that there are sets of active constraints for which there is not even a founded weak repair.

Example 2.16. Let $V = \emptyset$ and $\eta = \{\langle p \vee q, \{p \leftarrow \top\}\rangle, \langle \neg p \vee q, \{q \leftarrow \top\}\rangle\}$. The set $\{q \leftarrow \top\}$ is a PMA repair of V achieving $\mathsf{C}(\eta)$. However, there is no founded weak repair (and thus no founded repair either).

Last but not least, the next example illustrates *circularity of support*: each update action that is individually founded owes this to the presence of the other update actions in the repair.

Example 2.17. ([9], Example 3)
Let $V = \emptyset$ and $\eta = \{\langle p \vee q, \{p \leftarrow \top\}\rangle, \langle \neg p \vee q, \{q \leftarrow \top\}\rangle, \langle p \vee \neg q, \{p \leftarrow \top\}\rangle\}$. The set $\{p \leftarrow \top, q \leftarrow \top\}$ is the only founded weak repair of V by η: $p \leftarrow \top$ is founded on $\langle p \vee \neg q, \{p \leftarrow \top\}\rangle$ and $q \leftarrow \top$ is founded on $\langle \neg p \vee q, \{q \leftarrow \top\}\rangle$. It is also a founded repair.

Such repairs were considered to be unintended in [9] and the notion of *justified repair* was proposed to overcome the problem. We discuss this issue further in Section 5.3.

2.3.3. Justified weak repairs and justified repairs

Justified repairs use a stronger condition than foundedness in order to avoid the aforementioned circular dependencies. We start with the definition of a *closed* set of update actions.

Definition 2.18. Let η be a set of standard active constraints. For $r \in \eta$, all the literals in $\mathsf{C}(r)$ which cannot be affected by an update action in $\mathsf{R}(r)$ are called *non-updatable*. A set of update actions U is closed under an $r \in \eta$ when the following holds: if the update actions in U *falsify* all the non-updatable literals in $\mathsf{C}(r)$, then U must contain an update action from $\mathsf{R}(r)$. Furthermore, U is closed under η if it is closed under every $r \in \eta$.

For example, $\{p \leftarrow \top, q \leftarrow \top\}$ is closed under $\langle p \vee \neg q, \{p \leftarrow \top\}\rangle, \langle \neg p \vee q, \{q \leftarrow \top\}\rangle$ and $\langle p \vee q, \{p \leftarrow \top\}\rangle$, while it is neither closed under $\langle \neg p \vee \neg q, \{p \leftarrow \bot\}\rangle$ nor under $\langle \neg p \vee r, \{r \leftarrow \top\}\rangle$. The second step is to define the *no-effect actions* associated with an initial database V and an updated database V'.

Definition 2.19. Let V and V' be two databases. The update action $p \leftarrow \top$ is a no-effect action of (V, V') if $p \in V \cap V'$ and the update action $p \leftarrow \bot$ is a no-effect action of (V, V') if $p \notin V \cup V'$. The set $\mathsf{ne}(V, V')$ denotes the set of all no-effect actions of (V, V').

Clearly, for given V and U, we have that $V \diamond U = V \diamond (U \setminus U')$ for every $U' \subseteq \mathsf{ne}(V, V \diamond U)$. Returning to our initial aim now, the definitions of a *justified weak repair* and a *justified repair* are the following.

Definition 2.20. Let V be a database and let η be a set of standard active constraints. A consistent set of update actions U is a justified weak repair of V by η iff:[4]

(a) $U \cap \mathsf{ne}(V, V \diamond U) = \emptyset$ (no 'no-effect' actions)

(b) $U \cup \mathsf{ne}(V, V \diamond U)$ is closed under η

(c) there is no $U' \subset U \cup \mathsf{ne}(V, V \diamond U)$ such that:

(1) U' contains $\mathsf{ne}(V, V \diamond U)$

(2) U' is closed under η

Finally, if U is also a PMA repair of V achieving $\mathsf{C}(\eta)$, then U is a justified repair of V by η.

The next theorem shows the relationship between founded and justified repairs.

Theorem 2.21. ([9], Corollaries 1 and 2)
Let V be a database, U a consistent set of update actions and η a set of standard active constraints. If U is a justified weak repair of V by η, then U is also a founded weak repair of V by η (and likewise, if it is a justified repair of V by η, then it is also a founded repair of V by η).

The next example shows that the converse does not hold. Furthermore, it illustrates that for justified repairs circularity of support is no longer an issue.

Example 2.22. ([9], Example 5)
Consider again $V = \emptyset$ and $\eta = \{\langle p \vee q, \{p \leftarrow \top\}\rangle, \langle \neg p \vee q, \{q \leftarrow \top\}\rangle, \langle p \vee \neg q, \{p \leftarrow \top\}\rangle\}$. In contrast with Example 2.17 and its founded repair, there is no justified weak repair of V by η. As justified weak repairs are also founded weak repairs, we only have to check whether $U = \{p \leftarrow \top, q \leftarrow \top\}$ is a justified weak repair of V by η. Supposing that $\mathbb{P} = \{p, q\}$, we have $\mathsf{ne}(V, V \diamond U) = \mathsf{ne}(\emptyset, \{p, q\}) = \emptyset$ and $U \cup \mathsf{ne}(V, V \diamond U) = U$ is not a minimal set of update actions containing $\mathsf{ne}(V, V \diamond U)$ and closed under η, as $\emptyset$ also has these properties.

However if we replace η with $\eta' = \{\langle p \vee q, \{p \leftarrow \top, q \leftarrow \top\}\rangle, \langle \neg p \vee q, \{q \leftarrow \top\}\rangle, \langle p \vee \neg q, \{p \leftarrow \top\}\rangle\}$ then the set $\{p \leftarrow \top, q \leftarrow \top\}$ is both a justified and a founded repair of V by η' (and both are the only ones). The reason that the set $\{p \leftarrow \top, q \leftarrow \top\}$ is a justified repair now is because $\emptyset$ is not closed under η anymore (it trivially falsifies all the non-updatable literals in the first constraint, since now there aren't any, but it contains no update actions).

In the next two sections, we show that weak, PMA, founded and justified repairs can be captured in DL-PA.

[4]The original definition of justified weak repairs is slightly different than the one given here. However it is shown that the two are equivalent in [9, Theorem 1].

3. Repairs and weak repairs in DL-PA

We now embed Winslett's standard semantics (and thereby weak repairs) and the PMA (and thereby repairs *tout court*) into DL-PA. This was already done in [20], but our embeddings are slightly more elegant and are presented in a more uniform and streamlined way. We start with some auxiliary definitions.

To each propositional variable p we associate a *fresh* propositional variable $p^{\pm}$. Each $p^{\pm}$ will register whether or not the proposition p has been modified along the update.[5] This is necessary to ensure that every variable is modified at most once during a repair. We extend the definition to sets of variables $P \subseteq \mathbb{P}$: $P^{\pm} = \{p^{\pm} \mid p \in P\}$. With the information stored in the fresh variables $\mathbb{P}^{\pm}$, we can retrieve the initial valuation from a valuation $V \subseteq \mathbb{P} \cup \mathbb{P}^{\pm}$ through the set:

$$\{p \in V \ : \ p^{\pm} \notin V\} \cup \{p \notin V \ : \ p^{\pm} \in V\}$$

First, we need a program that sets all the propositions in a given set P to $\bot$: $P{\leftarrow}\bot$ is the sequence of assignments $p{\leftarrow}\bot$ for all $p \in P$ (whose order does not matter, see Proposition 2.4). Second, the following two DL-PA programs (1) modify a single proposition and store this and (2) undo that modification:

$$\begin{array}{lcl} \mathsf{toggle}(p) & = & \textbf{if } \neg p^{\pm} \textbf{ then } p{\leftarrow}\neg p; p^{\pm}{\leftarrow}\top \textbf{ else fail} \ = \ \neg p^{\pm}?; p{\leftarrow}\neg p; p^{\pm}{\leftarrow}\top \\ \mathsf{undo}(p) & = & \textbf{if } p^{\pm} \textbf{ then } p{\leftarrow}\neg p; p^{\pm}{\leftarrow}\bot \textbf{ else fail} \ = \ p^{\pm}?; p{\leftarrow}\neg p; p^{\pm}{\leftarrow}\bot \end{array}$$

As announced above, $p^{\pm}$ keeps track of the modifications of p: we are going to ensure that it is true only once p has been modified during the current update. The program $\mathsf{toggle}(p)$ flips the truth value of p if this value has not been modified yet and records the modification by setting $p^{\pm}$ to $\top$; if p has already been made true then $\mathsf{toggle}(p)$ fails. The program $\mathsf{undo}(p)$ undoes this.

It is easy to see then that starting from a database V that contains none of the variables $p^{\pm}$, a weak repair of V achieving C can be obtained through the following DL-PA program:

$$\mathsf{weakRepair}(C) = \left(\bigcup_{p \in \mathbb{P}_C} \mathsf{toggle}(p) \right)^* ; \Big(\bigwedge C \Big)?$$

Since each variable can be updated at most once and since the order of the updates does not matter, this can be rewritten without the Kleene star as a sequence:

$$(\mathsf{toggle}(p_1) \cup \textbf{skip}) ; \ldots ; (\mathsf{toggle}(p_k) \cup \textbf{skip}) ; \Big(\bigwedge C \Big)?$$

where $p_1, \ldots, p_k$ are the variables in $\mathbb{P}_C$. Furthermore, given that none of the variables $p^{\pm}$ occur in the database, the program $\mathsf{toggle}(p)$ simplifies to just: $p{\leftarrow}\neg p; p^{\pm}{\leftarrow}\top$.

Finally, we define the following DL-PA formula:

$$\mathsf{Minimal}(C) = \neg \left\langle \left(\bigcup_{p \in \mathbb{P}_C} \mathsf{undo}(p) \right)^+ \right\rangle \bigwedge C$$

[5] The difference with [20] is that our programs memorise that a variable has been flipped instead of storing its previous value.

The program in this formula undoes a nonempty set of $\mathsf{toggle}(p)$ actions (and nondeterministically so, failing when there was no change at all). Therefore the formula $\mathsf{Minimal}(C)$ says that there is no execution of that program leading to a database closer to the actual database that satisfies the constraints. Hence the actual database corresponds to a minimal change of the initial database. We sum up all the above in the following theorem.

Theorem 3.1. Let C be a set of static integrity constraints in the language of $\mathbb{P}$ and let $V \subseteq \mathbb{P}$ be a database (i.e., no $p^{\pm}$ occurs in either of them). Let U be a consistent set of update actions that is relevant w.r.t. V.

- U is a weak repair of V achieving C iff:

$$\langle V, V \diamond U \rangle \in \| \, \mathsf{weakRepair}(C) \; ; \; {\mathbb{P}_C}^{\pm} \leftarrow \bot \, \|$$

- U is a PMA repair of V achieving C iff:

$$\langle V, V \diamond U \rangle \in \| \, \mathsf{weakRepair}(C) \; ; \; \mathsf{Minimal}(C)? \; ; {\mathbb{P}_C}^{\pm} \leftarrow \bot \, \|$$

4. Founded and justified repairs in DL-PA

We will now move on to the embedding of the notions of founded and justified repairs into DL-PA. For this, we will re-use the programs defined in the previous section for finding a weak repair and checking minimality, as well as the set of fresh variables $\mathbb{P}^{\pm}$ we had at our disposal in order to keep track of modifications. Moreover, we will need to define a program for checking the foundedness condition in order to generate the founded repairs as well as programs for adding the no-effect actions to a database and checking if a set of update actions is closed under a set of active constraints.

We start with the embedding of founded repairs into DL-PA, for which we will need the following formula:

$$\mathsf{Founded}(\eta) = \bigwedge_{p \in \mathbb{P}_{\mathsf{C}(\eta)}} \left(p^{\pm} \rightarrow \bigvee_{\substack{r \in \eta \\ p \leftarrow \mathsf{X} \in \mathsf{R}(r)}} \langle p \leftarrow \neg p \rangle \neg \mathsf{C}(r) \right)$$

where X ranges over $\{\top, \bot\}$. We can see then that this formula is true if and only if all current update actions (encoded in the current valuation by means of the fresh variables $p^{\pm}$) are founded w.r.t. the initial valuation and η.

The embedding of justified repairs into DL-PA is a bit more complex. Firstly, we use the set $\mathsf{nup}(r)$ of all the *non-updatable* literals in r that we saw in Definition 2.18, i.e., all the literals in $\mathsf{C}(r)$ for which there is no preferred update action in $\mathsf{R}(r)$. By $\mathsf{nup}(r)^+$ we denote the set of propositional variables of the form p in $\mathsf{nup}(r)$, i.e., $\mathsf{nup}(r)^+ = \mathsf{nup}(r) \cap \mathbb{P}$. Similarly, $\mathsf{nup}(r)^-$ comprises the propositional variables of the form $\neg p$ in $\mathsf{nup}(r)$, i.e., $\mathsf{nup}(r)^- = \{\neg p \mid p \in \mathsf{nup}(r) \setminus \mathbb{P}\}$. Furthermore, we introduce two new sets of *fresh* propositional variables, P^+ and P^-, defined similarly to $P^{\pm}$ as follows: $P^+ = \{p^+ \mid p \in P\}$ and $P^- = \{p^- \mid p \in P\}$. Intuitively, the proposition p^+ will keep track of

the no-effect action of the form $p \leftarrow \top$, while the proposition p^- will keep track of the no-effect action of the form $p \leftarrow \bot$. This is realized by the following two programs:

$$\mathsf{ne}^+(p) = (p \wedge \neg p^{\pm})\,?\,;\, p^+ \leftarrow \top$$
$$\mathsf{ne}^-(p) = (\neg p \wedge \neg p^{\pm})\,?\,;\, p^- \leftarrow \top$$

Moreover, we will need a program that skips when neither $p \leftarrow \top$ nor $p \leftarrow \bot$ is a no-effect action and fails otherwise (where 'ea' stands for 'effect action'):

$$\mathsf{ea}(p) = p^{\pm}\,?$$

Then we associate with the current database all the no-effect actions between the initial database and the current state through the following program:

$$\mathsf{ne}(\eta) = (\mathsf{ne}^+(p_1) \cup \mathsf{ne}^-(p_1) \cup \mathsf{ea}(p_1))\,;\ldots;\,(\mathsf{ne}^+(p_k) \cup \mathsf{ne}^-(p_k) \cup \mathsf{ea}(p_k))$$

where $p_1, \ldots, p_k$ are the variables in $\mathbb{P}_{\mathsf{C}(\eta)}$ (so the no-effect actions that are added are only those that are relevant w.r.t. η). Next, we define $\mathsf{A}(\mathsf{r})$ and $\mathsf{B}(\mathsf{r})$ to be the following formulas:

$$\mathsf{A}(\mathsf{r}) = \Big(\bigwedge_{p \in \mathsf{nup}(r)^+} (\neg p \wedge p^{\pm}) \vee (\neg p \wedge p^-)\Big) \wedge \Big(\bigwedge_{p \in \mathsf{nup}(r)^-} (p \wedge p^{\pm}) \vee (p \wedge p^+)\Big)$$
$$\mathsf{B}(\mathsf{r}) = \Big(\bigvee_{p \leftarrow \top \in \mathsf{R}(r)} p\Big) \vee \Big(\bigvee_{p \leftarrow \bot \in \mathsf{R}(r)} \neg p\Big)$$

Using these then we define:

$$\mathsf{Closed}(\eta) = \bigwedge_{r \in \eta} (\mathsf{A}(\mathsf{r}) \rightarrow \mathsf{B}(\mathsf{r}))$$
$$\mathsf{MinimallyClosed}(\eta) = \neg \left\langle \left(\bigcup_{p \in \mathbb{P}_{\mathsf{C}(\eta)}} \mathsf{undo}(p) \right)^+ \right\rangle \mathsf{Closed}(\eta)$$

Given a set of standard active constraints η, the first formula is true exactly when all current update actions (again, encoded in the current valuation through the set $P^{\pm}$) plus all the no-effect actions (encoded in the current valuation through the sets P^+ and P^-) are closed under η, while the second formula is true if and only if the set comprising of all current update actions is the minimal set of update actions that (together with the no-effect actions) has this property. Finally, we define the following abbreviations:

$$\mathsf{Justified}(\eta)? = \mathsf{ne}(\eta)\ ;\ \mathsf{Closed}(\eta)?\ ;\ \mathsf{MinimallyClosed}(\eta)?$$
$$\mathsf{ClearAll} = \mathbb{P}_C{}^{\pm} \leftarrow \bot\ ;\ \mathbb{P}_C{}^{+} \leftarrow \bot\ ;\ \mathbb{P}_C{}^{-} \leftarrow \bot$$

The following two theorems now give a complete characterisation of founded and justified repairs in terms of DL-PA programs.

Theorem 4.1. Let η be a set of active integrity constraints in the language of $\mathbb{P}$ and let $V \subseteq \mathbb{P}$ be a database (i.e., no $p^{\pm}$ occurs in either of them). Let U be a consistent set of update actions that is relevant w.r.t. V.

- U is a founded weak repair of V by η iff:

$$\langle V, V \diamond U \rangle \in \left\| \mathsf{weakRepair}(\mathsf{C}(\eta)) \,;\, \mathsf{Founded}(\eta)? \,;\, {\mathbb{P}_C}^{\pm} \leftarrow \bot \right\|$$

- U is a founded repair of V by η iff:

$$\langle V, V \diamond U \rangle \in \left\| \mathsf{weakRepair}(\mathsf{C}(\eta)) \,;\, \mathsf{Founded}(\eta)? \,;\, \mathsf{Minimal}(\mathsf{C}(\eta))? \,;\, {\mathbb{P}_C}^{\pm} \leftarrow \bot \right\|$$

Theorem 4.2. Let η be a set of standard active constraints in the language of $\mathbb{P}$ and let $V \subseteq \mathbb{P}$ be a database (so no $p^{\pm}$, p^{+} and p^{-} occurs in either of them). Let U be a consistent set of update actions that is relevant w.r.t. V.

- U is a justified weak repair of V by η iff:

$$\langle V, V \diamond U \rangle \in \left\| \mathsf{weakRepair}(\mathsf{C}(\eta)) \,;\, \mathsf{Justified}(\eta)? \,;\, \mathsf{ClearAll} \right\|$$

- U is a justified repair of V by η iff:

$$\langle V, V \diamond U \rangle \in \left\| \mathsf{weakRepair}(\mathsf{C}(\eta)) \,;\, \mathsf{Justified}(\eta)? \,;\, \mathsf{Minimal}(\mathsf{C}(\eta))? \,;\, \mathsf{ClearAll} \right\|$$

5. A new definition of repair in DL-PA

We now propose some new definitions that take advantage of the resources of DL-PA. More precisely, we make use of **while** loops in order to iterate the application of active constraints. We start by discussing how databases can be repaired by applying active constraints in sequence. This will lead us to the definition of dynamic repair. We show that it is incomparable with both founded and justified repairs and discuss its properties and some variants.

5.1. Repairing a database: a dynamic view

Suppose there is only one active constraint r that is standard. Then it is clear how to proceed: either $V \models \mathsf{C}(r)$ and there is nothing to do, or $V \not\models \mathsf{C}(r)$ and we have to apply r. In the second case, each $\alpha_i \in \mathsf{R}(r)$ provides a PMA repair of V achieving $\mathsf{C}(r)$.[6] What about the case where $\mathsf{R}(r)$ is empty? Well, then V cannot be repaired and we are stuck.

So far so good. The situation may get way more intricate when the set of active constraints η contains two or more elements that can interact. Firstly, the example of the introductory section and Example 2.16 illustrated an instance of active constraints which intuitively should have a repair (and

[6]For our more general active constraints where there is no syntactical link between $\mathsf{C}(r)$ and $\mathsf{R}(r)$ we have to compute all possible minimal subsets $U \subseteq \mathsf{R}(r)$ such that $V \diamond U \models \mathsf{C}(r)$. All of them will be PMA repairs.

it does, in the case of PMA repairs) but for which there is no founded or justified weak repair. We would like to find a definition of a repair which depends only on the preferred update actions and always provides a repaired database, as long as there are update actions from each $\mathsf{C}(r)$ to choose from. Moreover, even for standard active constraints it might not be enough to apply an update action $\alpha_i \in \bigcup_{r\in\eta} \mathsf{R}(r)$ only once: some of the active constraints might have to be applied several times in order to obtain integrity. The following active constraints that are inspired by an $(n{+}1)$-bit counter highlight this.

Suppose for $n \geq 0$ we represent binary numbers up to $2^{n+1}{-}1$ by means of $n{+}1$ propositional variables $p_0, \ldots, p_n$: $\neg p_n \wedge \cdots \wedge \neg p_0$ represents the integer zero and $p_n \wedge \cdots \wedge p_0$ represents $2^{n+1}{-}1$. For each bit we also need an auxiliary variable x_i. Let:

$$
\begin{aligned}
r_1 &= \langle p_0 \vee x_1 \vee \cdots \vee x_n, \{p_0 \leftarrow \top\}\rangle \\
r_{2_k} &= \langle p_k \vee \neg p_{k-1} \vee \cdots \vee \neg p_0 \vee x_k, \{x_k \leftarrow \top\}\rangle, \quad \text{for } 1 \leq k \leq n \\
r_{3_{k_i}} &= \langle \neg p_i \vee \neg x_k, \{p_i \leftarrow \bot\}\rangle, \quad \text{for } 1 \leq k \leq n \text{ and } 0 \leq i \leq k-1 \\
r_{3_{k_k}} &= \langle p_k \vee \neg x_k, \{p_k \leftarrow \top\}\rangle, \quad \text{for } 1 \leq k \leq n \\
r_{4_k} &= \langle \neg p_k \vee p_{k-1} \vee \cdots \vee p_0 \vee \neg x_k, \{x_k \leftarrow \bot\}\rangle, \quad \text{for } 1 \leq k \leq n
\end{aligned}
$$

The idea is that when $\neg p_k \wedge p_{k-1} \wedge \cdots \wedge p_0$ is true, i.e., when the number $011\ldots1$ has to be incremented to $100\ldots0$, then x_k is made true by r_{2_k} and remains so until $100\ldots0$ has been attained. This involves flipping the k digits in the conjunction $\neg p_k \wedge p_{k-1} \wedge \cdots \wedge p_0$: with active constraints this is done one-by-one by $r_{3_{k_i}}$ and $r_{3_{k_k}}$. Then x_k is set to false again by r_{4_k}. Let η_n be the set of all the above rules, for a given n:

$$
\eta_n = \{r_1\} \cup \{r_{2_1}, \ldots, r_{2_n}\} \cup \{r_{3_{1_0}}, r_{3_{1_1}}\} \cup \{r_{3_{2_0}}, r_{3_{2_1}}, r_{3_{2_2}}\} \cup \ldots \cup \{r_{3_{n_0}}, \ldots, r_{3_{n_n}}\} \cup \{r_{4_1}, \ldots, r_{4_n}\}
$$

Successive repairing steps implement an $(n{+}1)$-bit counter counting from the initial database $\emptyset$ to the database $\{p_n, \ldots, p_0\}$.

The computation takes a number of steps that is exponential in n, while the number of update actions is $\frac{1}{2}(n^2{+}7n){+}1$, demonstrating that sometimes atomic repairs must be performed an exponential number of times: for example r_1 needs to be applied 2^n times in order to repair $V_0 = \emptyset$ by η_n. Let us illustrate by the 3-bit counter how the repairs are done.

Example 5.1. Let's take $n = 2$ and try to obtain the integer 111 starting from 000. In Figure 1 we can see the steps needed through which we will reach the set of update actions $U = \{p_0 \leftarrow \top, p_1 \leftarrow \top, p_2 \leftarrow \top\}$ that will update the database $\emptyset$ in order for it to satisfy the active integrity constraints in η_2. The first column represents the current database (starting from $\emptyset$), the second column shows the constraint that was applied in order to reach it and in the third we see the current integer in the counter. Last but not least, the last column shows when an x_k is kept *true* in order for the procedure to reach the integer 10 from 01 and the integer 100 from 011 when needed.

$\neg p_2 \wedge \neg p_1 \wedge \neg p_0 \wedge \neg x_1 \wedge \neg x_2$		000	–
$\neg p_2 \wedge \neg p_1 \wedge \ p_0 \wedge \neg x_1 \wedge \neg x_2$	r_1	001	–
$\neg p_2 \wedge \neg p_1 \wedge \ p_0 \wedge \ x_1 \wedge \neg x_2$	r_{2_1}	001	✓
$\neg p_2 \wedge \neg p_1 \wedge \neg p_0 \wedge \ x_1 \wedge \neg x_2$	$r_{3_{1_0}}$	000	✓
$\neg p_2 \wedge \ p_1 \wedge \neg p_0 \wedge \ x_1 \wedge \neg x_2$	$r_{3_{1_1}}$	010	✓
$\neg p_2 \wedge \ p_1 \wedge \neg p_0 \wedge \neg x_1 \wedge \neg x_2$	r_{4_1}	010	–
$\neg p_2 \wedge \ p_1 \wedge \ p_0 \wedge \neg x_1 \wedge \neg x_2$	r_1	011	–
$\neg p_2 \wedge \ p_1 \wedge \ p_0 \wedge \neg x_1 \wedge \ x_2$	r_{2_2}	011	✓
$\neg p_2 \wedge \ p_1 \wedge \neg p_0 \wedge \neg x_1 \wedge \ x_2$	$r_{3_{2_0}}$	010	✓
$\neg p_2 \wedge \neg p_1 \wedge \neg p_0 \wedge \neg x_1 \wedge \ x_2$	$r_{3_{2_1}}$	000	✓
$\ p_2 \wedge \neg p_1 \wedge \neg p_0 \wedge \neg x_1 \wedge \ x_2$	$r_{3_{2_2}}$	100	✓
$\ p_2 \wedge \neg p_1 \wedge \neg p_0 \wedge \neg x_1 \wedge \neg x_2$	r_{4_2}	100	–
$\ p_2 \wedge \neg p_1 \wedge \ p_0 \wedge \neg x_1 \wedge \neg x_2$	r_1	101	–
$\ p_2 \wedge \neg p_1 \wedge \ p_0 \wedge \ x_1 \wedge \neg x_2$	r_{2_1}	101	✓
$\ p_2 \wedge \neg p_1 \wedge \neg p_0 \wedge \ x_1 \wedge \neg x_2$	$r_{3_{1_0}}$	100	✓
$\ p_2 \wedge \ p_1 \wedge \neg p_0 \wedge \ x_1 \wedge \neg x_2$	$r_{3_{1_1}}$	110	✓
$\ p_2 \wedge \ p_1 \wedge \neg p_0 \wedge \neg x_1 \wedge \neg x_2$	r_{4_1}	110	–
$\ p_2 \wedge \ p_1 \wedge \ p_0 \wedge \neg x_1 \wedge \neg x_2$	r_1	111	–

Figure 1: Incrementing 000 to 111 through η_2

We can see how some constraints need to be applied many times in order to succeed in repairing the original database. This calls for a dynamic way through which a database is updated in order for it to be repaired: a procedure that modifies the database according to the integrity constraints step by step, until it reaches a satisfactory form (i.e., satisfies the integrity constraints). Founded and justified repairs cannot do the job in this and other scenarios of that kind, as an active constraint can only be used once: indeed, in the example of the $(n+1)$-bit counter, no repair can be obtained by means of founded and justified repairs. That's why we introduce dynamic repairs.

5.2. Dynamic weak repairs and dynamic repairs

We associate with every active constraint r the DL-PA programs:

$$\pi_r = \neg \mathsf{C}(r)\,?\,; \bigcup_{\alpha \in \mathsf{R}(r)} \alpha$$

$$\pi_r^{\pm} = \neg \mathsf{C}(r)\,?\,; \bigcup_{p \leftarrow \mathsf{X} \in \mathsf{R}(r)} (p \leftarrow \mathsf{X}; p^{\pm} \leftarrow \top)$$

Remember that $\bigcup_{\alpha \in \mathsf{R}(r)} \alpha$ equals **fail** when $\mathsf{R}(r)$ is empty. This matches the intuitive reading that we have given to active constraints in Section 2.3.1: the repair program π_r checks whether the static

integrity constraint associated with r is violated, and if so nondeterministically applies one of the update actions from $\mathsf{R}(r)$. The program $\pi_r^{\pm}$ moreover stores that p has been changed. These intuitions are also supported by the following proposition, which tells us that applicability of an active constraint r (the fact that $\mathsf{C}(r)$ is violated) is matched by the DL-PA notion of executability of the program π_r.

Proposition 5.2. Let r be an active constraint and let V be a database. Then applicability of r at V is equivalent to both $V \models \langle \pi_r \rangle \top$ and $V \models \langle \pi_r^{\pm} \rangle \top$.

Proof:
It suffices to observe that when π is a nondeterministic composition of update actions then the equivalence $\neg \mathsf{C}(r) \leftrightarrow \langle \neg \mathsf{C}(r)?; \pi \rangle \top$ is DL-PA valid for every $\mathsf{C}(r)$. □

Based on these, the definitions of a *dynamic weak repair* and a *dynamic repair* are the following.

Definition 5.3. Let V be a database and let η be a set of active constraints. A dynamic weak repair of V by η is a consistent set of update actions U such that U is relevant w.r.t. V and:

$$\langle V, V \diamond U \rangle \in \left\| \textbf{while } \neg \left(\bigwedge \mathsf{C}(\eta) \right) \textbf{ do } \left(\bigcup_{r \in \eta} \pi_r \right) \right\|$$

Moreover, if U is also a PMA repair of V achieving $\mathsf{C}(\eta)$, then U is a dynamic repair of V by η.

In the following example we see that dynamic repairs sometimes coincide with founded repairs.

Example 5.4. (Example 2.17, ctd.)
Consider again $V = \emptyset$ and $\eta = \{\langle p \vee q, \{p \leftarrow \top\}\rangle, \langle \neg p \vee q, \{q \leftarrow \top\}\rangle, \langle p \vee \neg q, \{p \leftarrow \top\}\rangle\}$. There is a single dynamic weak repair (and also dynamic repair) of V by η, viz. $\{p \leftarrow \top, q \leftarrow \top\}$. Remember by Example 2.22 that there is no justified repair.

As we have already witnessed with the $(n{+}1)$-bit counter though, dynamic weak repairs are not necessarily founded. The next example is simpler.

Example 5.5. (Example 2.15, ctd.)
Consider again $V = \emptyset$ and $\eta = \{\langle p, \{p \leftarrow \top\}\rangle, \langle p \vee q, \{q \leftarrow \top\}\rangle\}$, whose only founded weak repair was $\{p \leftarrow \top\}$. There are two dynamic weak repairs of V by η, namely $\{p \leftarrow \top\}$ and $\{p \leftarrow \top, q \leftarrow \top\}$. Only the former is a dynamic repair.

Remember also that at the beginning of Section 5.1 we argued against founded and justified repairs using Example 2.16 (itself an adaptation of the example discussed in the introductory section), for which we would like to have a way to repair V by η. The next example shows that dynamic weak repairs solve this problem. Let us also note that just like founded and justified repairs, dynamic weak repairs do not necessarily coincide with dynamic repairs.

Example 5.6. (Example 2.16, ctd.)
Consider again $V = \emptyset$ and $\eta = \{\langle p \vee q, \{p \leftarrow \top\}\rangle, \langle \neg p \vee q, \{q \leftarrow \top\}\rangle\}$. The only dynamic weak repair of V by η is the set of update actions $\{p \leftarrow \top, q \leftarrow \top\}$. But $\{q \leftarrow \top\}$ is the PMA repair of V achieving $\mathsf{C}(\eta)$, so there is no dynamic repair.

Finally, in a similar manner as in the previous sections, the next theorem characterises dynamic repairs in terms of DL-PA programs.

Theorem 5.7. Let η be a set of active integrity constraints in the language of $\mathbb{P}$ and let $V \subseteq \mathbb{P}$ be a database (i.e., no $p^{\pm}$ occurs in either of them). Let U be a consistent set of update actions that is relevant w.r.t. V. U is a dynamic repair of V by η iff:

$$\langle V, V \diamond U \rangle \in \left\| \textbf{while } \neg \Big(\bigwedge \mathsf{C}(\eta) \Big) \textbf{ do } \Big(\bigcup_{r \in \eta} \pi_r^{\pm} \Big) ; \mathsf{Minimal}\Big(\mathsf{C}(\eta)\Big)? ; {\mathbb{P}_C}^{\pm} \leftarrow \bot \right\|$$

Proof:
The proof is quite trivial and based on the definitions. Given a database $V \subseteq \mathbb{P}$ and a set of active integrity constraints η, a dynamic repair of V by η is both a dynamic weak repair of V by η and a PMA repair of V achieving $\mathsf{C}(\eta)$. In DL-PA terms then, this is given by the sequential composition of the programs $\textbf{while } \neg (\bigwedge \mathsf{C}(\eta)) \textbf{ do } \big(\bigcup_{r \in \eta} \pi_r^{\pm} \big)$ and $\mathsf{Minimal}\big(\mathsf{C}(\eta)\big)?$, with the DL-PA program $\pi_r^{\pm}$ keeping track of which propositions have been modified along the update so that they can be checked again in the latter. Finally, as before the program ${\mathbb{P}_C}^{\pm} \leftarrow \bot$ ensures that no $p^{\pm}$ exists in U. □

5.3. Some interesting properties

In this subsection we present some interesting properties of this dynamic procedure that distinguishes it from the other main repairs which have been studied and prevailed in the literature, viz. weak repairs, PMA repairs, founded and justified repairs. Our goal is to provide a concrete argument that repairs produced in this way are an interesting kind of repairs, possessing the advantages of the others while not comprising some of their disadvantages.

The main problem with founded repairs is the so called *circularity of support* which has been already mentioned at the end of Section 2.3.2. This undesirable property is what ultimately led to the definition of justified repairs, which are way more complex and difficult to understand, at least at first sight. Dynamic repairs on the other hand provide a solution to this problem without straying too far from the initial definition, making it far less intricate. In contrast with founded repairs which need the *foundedness* property to give priority to the "preferred" repairs, dynamic repairs simply select an update action from the set that they have access to (the set of preferred ones) without checking for any other condition. This leads to no *circular support* between any set of preferred actions and can also be seen in Example 5.4 where $\{p \leftarrow \top, q \leftarrow \top\}$ remains a dynamic repair of V by η even if the constraint '$\langle p \vee \neg q, \{p \leftarrow \top\} \rangle$' is absent (something that cannot be said for founded repairs, as this constraint is required for the foundedness of 'p'). Furthermore, although justified repairs solve this problem, they often do not exist, as can be seen by Example 2.22. Through dynamic repairs we can provide a solution to cases like this, avoiding the *circularity of support* found in founded repairs, while still being able to provide a repaired database. So not only are dynamic repairs more intuitive, but they also comprise the best of both situations.

Despite this however, one could still argue that they are too "strict", sometimes requiring that every integrity constraint in $\mathsf{C}(\eta)$ has a way to be repaired (i.e., an update action in $\mathsf{R}(r)$ should exist for all $r \in \eta$ in order to make $\mathsf{C}(r)$ true). If $\mathsf{R}(r) = \emptyset$ for some $r \in \eta$, then the whole dynamic repairing

procedure could collapse and a dynamic weak repair never occur. The next example illustrates exactly this.

Example 5.8. Let $V = \{q\}$ and $\eta = \{\langle p \vee q, \{p \leftarrow \top\}\rangle, \langle p, \emptyset\rangle\}$. The set $\{p \leftarrow \top\}$ is a PMA repair of V achieving $\mathsf{C}(\eta)$. However, there are no dynamic weak repairs (and thus no dynamic repairs either).

As we can see, the problem arises when an integrity constraint has no preferred update actions and cannot be satisfied by the application of some other constraint when repairing the database. One could differentiate this behavior into three classes of repairs, based on the level of "strictness" of the preference that is involved in the active constraints. The more strict repairs are those conforming to the idea that every integrity constraint $\mathsf{C}(r)$ should be repaired only through an update action in $\mathsf{R}(r)$, the less strict allow any update actions in $\bigcup_{r\in\eta} \mathsf{R}(r)$ to be used for any integrity constraint $\mathsf{C}(r)$, while the middle ground is to keep a balance between the two. As we can see, the passing from the more strict class to the others changes the meaning of the update actions in $\mathsf{R}(r)$ from *permitted* repair actions to *preferred* ones, a distinction that is not always made clear in the literature. As Example 5.8 shows, dynamic repairs and dynamic weak repairs possess some of this "strict" nature: an update action will only arise while updating a database if it helps to repair some constraint. This forbids repairs in cases where all clauses apart from those having no preferred actions are already satisfied. In contrast, in such cases solutions with founded weak repairs can occur, as shown in the next example.

Example 5.9. ([9], Example 2)
Let $V = \emptyset$ and $\eta = \{\langle p \vee \neg q \vee \neg r, \{p \leftarrow \top\}\rangle, \langle \neg p \vee q \vee \neg r, \{q \leftarrow \top\}\rangle, \langle \neg p \vee \neg q \vee r, \{r \leftarrow \top\}\rangle, \langle p, \emptyset\rangle\}$. The set $\{p \leftarrow \top, q \leftarrow \top, r \leftarrow \top\}$ is the only founded weak repair of V by η. Furthermore, the set $\{p \leftarrow \top\}$ is a PMA repair of V achieving $\mathsf{C}(\eta)$. There are no dynamic weak repairs (and thus no dynamic repairs either).

So this leads to the following question: should we require a repair to exist in such cases or not? Are we willing to agree with the fact that such databases do not and should not have a repair, or is repairing the database in order to satisfy the integrity constraints in $\mathsf{C}(\eta)$ of the highest priority?

If the answer to the last question is positive then dynamic repairs could be less interesting. We can however tweak the definition slightly and define *global-dynamic* weak repairs to be a kind of dynamic repairs with the same intuitive behavior as before but belonging to the least strict of the aforementioned classes of repairs. More precisely, the reason that dynamic repairs cannot repair a constraint using update actions found in the other clauses is the *local* nature of the **do** part in the **while** loop. Before trying to repair the whole set of active integrity constraints in η, a dynamic repair *locally* checks if every clause (integrity constraint) is satisfied. If we drop this requirement and allow the dynamic procedure to *globally* choose update actions found in the other clauses, then we will have a solution in Examples 5.8, 5.9 and more generally the cases we have discussed.

Definition 5.10. Let V be a database and let η be a set of active constraints. A global-dynamic weak repair of V by η is a consistent set of update actions U such that U is relevant w.r.t. V and:

$$\langle V, V \diamond U\rangle \in \left\| \textbf{while } \neg\Big(\bigwedge \mathsf{C}(\eta)\Big) \textbf{ do } \Big(\bigcup_{\substack{r\in\eta \\ \alpha\in\mathsf{R}(r)}} \alpha\Big) \right\|$$

In the same vein as before, if U is also a PMA repair of V achieving $\mathsf{C}(\eta)$, then U is a global-dynamic repair of V by η.

It is easy to see now that this tweaked definition provides us with the desired repaired database in Examples 5.8 and 5.9. Specifically, the set $\{p\leftarrow\top\}$ is a global-dynamic repair of V by η in both examples. Furthermore, a dynamic weak repair is always a global-dynamic weak repair, as it can be created by the same procedure using one step less, namely by not checking the condition "$\neg\mathsf{C}(r)$?" in the **do** part of the program. This makes dynamic weak repairs a subset of global-dynamic weak repairs (and also dynamic repairs a subset of global-dynamic repairs).

It is our intention to use the global-dynamic repairs mainly as a tool of comparison between the different classes of repairs and less as a practical repairing technique that would replace the others. As we will see, the most important attribute of global-dynamic repairs is that they are exactly the global-dynamic weak repairs that are minimal w.r.t. set inclusion (i.e., if U is a global-dynamic repair of V then there is no global-dynamic weak repair U' of V such that $U' \subset U$). The recipe when defining repairs till now is to first give the definition of their *weak* versions and then state that they also have to be PMA repairs. This is of course different from saying that these repairs are the weak repairs that are minimal w.r.t. set inclusion, as in this case they would always exist if at least one of their weak counterparts existed. But it is not always the case that they may coincide with PMA repairs and usually may not exist altogether. This can be witnessed in Example 5.9 for founded repairs and in Example 5.6 for dynamic repairs.

However, when minimality w.r.t. set inclusion and coincidence with PMA repairs is the same, we have a much more powerful and reliable tool in our hands that avoids the main disadvantage of other repairs, namely that they may not exist (even if their weak versions do). This is shown in Theorem 5.12. Before this, a small lemma characterises this global-dynamic nature.

Lemma 5.11. Let η be a set of active integrity constraints in the language of $\mathbb{P}$ and let $V \subseteq \mathbb{P}$ be a database. Let U_1 be a global-dynamic weak repair of V by η and U_2 be a weak repair of V achieving $\mathsf{C}(\eta)$ such that $U_2 \subset U_1$. Then U_2 is also a global-dynamic weak repair of V by η.

Proof:
By hypothesis, U_1 and U_2 are both consistent sets of update actions that are relevant w.r.t. V such that $V \diamond U_1 \models \bigwedge \mathsf{C}(\eta)$ and $V \diamond U_2 \models \bigwedge \mathsf{C}(\eta)$ with $U_2 \subset U_1$. This means that V can be updated with less update actions than U_1 in order to satisfy the integrity constraints in η. But U_1 was constructed by iteration on checking the satisfaction of $\bigwedge \mathsf{C}(\eta)$ and applying update actions from $\bigcup_{r\in\eta} \mathsf{R}(r)$. Then U_2 can be constructed in exactly the same way, since it doesn't comprise any update actions outside of U_1, with the difference of taking less update actions into account: namely, by restricting the nondeterministic choice to updates from U_2 and leaving the update actions in $U_1 \setminus U_2$ out of consideration. This will also lead to a repaired database. So U_2 is a global-dynamic weak repair of V by η as well. □

Theorem 5.12. Let η be a set of active integrity constraints in the language of $\mathbb{P}$ and let $V \subseteq \mathbb{P}$ be a database. Let U be a consistent set of update actions that is relevant w.r.t. V. U is a global-dynamic repair of V by η iff U is a global-dynamic weak repair of V by η that is minimal w.r.t. set inclusion.

Proof:
Let the set GDR consist of all the global-dynamic repairs of V by η and let $MGDWR$ consist of all the global-dynamic weak repairs of V by η that are minimal w.r.t. set inclusion. For convenience we also define in the same way R as the set of all PMA repairs and WR as the set of all weak repairs of V achieving $\mathsf{C}(\eta)$. Finally, let $GDWR$ be the set of all global-dynamic weak repairs of V by η. First of all, observe that $GDR = GDWR \cap R$ (1) and $GDWR \subseteq WR$ (2). Let us show that $GDR = MGDWR$.

- $GDR \subseteq MGDWR$: let $U_1 \in GDR$. By (1) then, $U_1 \in GDWR$ and $U_1 \in R$. Let $U_2 \in GDWR$ such that $U_2 \subset U_1$. By (2) we also have that $U_2 \in WR$. This means that $U_1 \in R$ and $U_2 \in WR$ with $U_2 \subset U_1$. But this cannot be the case, as a PMA repair is a minimal weak repair w.r.t. set inclusion. So there is no $U_2 \in GDWR$ such that $U_2 \subset U_1$, where $U_1 \in GDWR$. Thus $U_1 \in MGDWR$.
  Note that this also applies to founded, justified and dynamic repairs. The difference is in the other direction.

- $MGDWR \subseteq GDR$: let $U_1 \in MGDWR$. By definition then, $U_1 \in GDWR$ and there is no $U' \in GDWR$ such that $U' \subset U_1$ (3). Let $U_2 \in WR$ such that $U_2 \subset U_1$. By Lemma 5.11 it is also the case then that $U_2 \in GDWR$. But this cannot be the case by (3). So there is no $U_2 \in WR$ such that $U_2 \subset U_1$. Since by (2) we also have $U_1 \in WR$, this means that $U_1 \in R$. Thus $U_1 \in GDWR \cap R$ and using (1) we get $U_1 \in GDR$.

□

So there is enough evidence to support the idea of using global-dynamic repairs as our repairs of choice when we want to update a database taking into account active integrity constraints in the cases where other repairs don't work. They are the closest thing to a PMA repair as shown by the next proposition, with the only limitation of being non-existent if the set of update actions $\bigcup_{r\in\eta} \mathsf{R}(r)$ is empty or if an integrity constraint can't be repaired even through update actions existing in other clauses (in both cases a solution to this problem shouldn't exist intuitively).

Proposition 5.13. Let η be a set of active integrity constraints in the language of $\mathbb{P}$ and let $V \subseteq \mathbb{P}$ be a database. Let U be a consistent set of update actions that is relevant w.r.t. V. U is a global-dynamic repair of V by η iff U is a PMA repair of V achieving $\mathsf{C}(\eta)$ and $U \subseteq \bigcup_{r\in\eta} \mathsf{R}(r)$.

Proof:
The left-to-right direction is trivial by the definition of global-dynamic repairs, whereas the right-to-left direction follows immediately by Theorem 5.12. □

But ultimately the choice between dynamic and global-dynamic repairs is traced back to what the answer should be regarding the repairing or not of a database in all of these cases. The former use a more restrictive procedure that makes it more *local*, while the latter do not.

Remark 5.14. In dynamic weak repairs and dynamic repairs we can use the $\bigcup_{p\in\mathbb{P}_{\mathsf{R}(r)}} \mathsf{toggle}(p)$ program in the place of $\bigcup_{\alpha\in\mathsf{R}(r)} \alpha$ in π_r without any change in the dynamic behavior of the repairs. In the

case of global-dynamic weak repairs, at first sight the programs:

$$\textbf{while } \neg\Big(\bigwedge \mathsf{C}(\eta)\Big) \textbf{ do } \Big(\bigcup_{\substack{r\in\eta\\ \alpha\in\mathsf{R}(r)}} \alpha\Big) \quad \text{and} \quad \textbf{while } \neg\Big(\bigwedge \mathsf{C}(\eta)\Big) \textbf{ do } \Big(\bigcup_{\substack{r\in\eta\\ p\in\mathbb{P}_{\mathsf{R}(r)}}} \mathsf{toggle}(p)\Big)$$

seem to once again bring the same results. This bears the question of whether we could instead use the second definition which abbreviates:

$$\left(\neg\Big(\bigwedge \mathsf{C}(\eta)\Big)?\,;\bigcup_{\substack{r\in\eta\\ p\in\mathbb{P}_{\mathsf{R}(r)}}} \mathsf{toggle}(p)\right)^{*};\Big(\bigwedge \mathsf{C}(\eta)\Big)?$$

and which is highly reminiscent of the program $\mathsf{weakRepair}(\mathsf{C}(\eta))$, showing its close relationship with weak and PMA repairs. It should be clarified why $\mathsf{toggle}(p)$ doesn't work anymore. The reason is that *toggling* a propositional variable in this case is not the same as choosing the respective update action, as $\mathsf{toggle}(p)$ can bring the opposite results. We can see this when $V = \emptyset$ and $\eta = \{\langle \neg p \vee q, \{p \leftarrow \bot\}\rangle, \langle \neg q \vee p, \{q \leftarrow \bot\}\rangle, \langle r, \{r \leftarrow \top\}\rangle\}$. A global-dynamic weak repair of V by η using the alternative definition with $\mathsf{toggle}(p)$ is the set of update actions $\{p \leftarrow \top, q \leftarrow \top, r \leftarrow \top\}$, which is obviously absurd.

As already mentioned, this does not happen with dynamic weak repairs and dynamic repairs. It is another aspect of their "strict" and *local* nature, as they check if a clause needs repairing before toggling any propositional variable, thus making any update action chosen to be exactly the intended one from the set of preferred ones.

6. Complexity of dynamic repairs

In this section we provide tight complexity bounds for the problems of existence of a dynamic weak repair and a dynamic repair. It is known that deciding the existence of a repair is NP-complete for weak repairs, PMA repairs and founded weak repairs, while it is Σ_P^2-complete for founded repairs, justified weak repairs and justified repairs [9]. As we will see, the same problem proves to be more difficult for dynamic weak repairs and dynamic repairs: deciding their existence is PSPACE-complete. For the lower bound (hardness) we provide a reduction from the problem of checking whether a fully quantified Boolean formula is true, whereas for the upper bound (membership) a reduction to the model checking problem of DL-PA will suffice. The result follows from the fact that checking whether a fully quantified Boolean formula is true and DL-PA model checking are both PSPACE-complete problems [29, 14].

6.1. Lower complexity bound

The hardness result is articulated in the following theorem, for which we only provide a proof sketch.

Theorem 6.1. The problems of existence of a dynamic weak repair and a dynamic repair are both PSPACE-hard.

Proof Sketch:
In order to show that the existence of dynamic weak repairs and dynamic repairs is PSPACE-hard we will provide a reduction from the following problem: given a fully quantified Boolean formula G, decide whether G is true. We suppose w.l.o.g. that G is in prenex normal form, with the variables in the prefix being all different and the matrix containing only the Boolean connectives $\neg$ and $\wedge$. Let $\text{subf}(G)$ be the set comprising all the subformulas of G and let $\text{subf}^v(G)$ be the set comprising all the variables in G. We define the set $\mathbb{P}_G$ of propositional variables to be composed of:

- all x, $x^?$ and $x^!$ such that $x \in \text{subf}^v(G)$
- $A^{+?}, A^{-?}, A^{+!}$ and $A^{-!}$ for each $A \in \text{subf}(G)$

Intuitively, the elements of $\mathbb{P}_G$ play the following roles: x stores the truth value of x in G, $x^?$ indicates that a value for x must be chosen and $x^!$ indicates that the value for x has been chosen. Similarly, $A^{+?}$ indicates that we check if A is true and $A^{+!}$ indicates that A has been proved to be true. The same goes for $A^{-?}$ and $A^{-!}$, for checking and proving that A is false.

The idea now is to start from the initial formula G and to compute whether it is true by asking whether there is a dynamic procedure that repairs the database $\{G^{+?}\}$ under a set of active constraints. This set can be defined by first assigning a set of active constraints to each $A \in \text{subf}(G)$, indicating the required steps for checking whether or not A can be proved true or false, and taking their union. For each A, the goal is to reach a state satisfying $\neg A^{+?} \wedge A^{+!}$ if we want to prove that A is true and $\neg A^{-?} \wedge A^{-!}$ if we want to prove that A is false. Indeed, each $A \in \text{subf}(G)$ can be associated with a set of active constraints which repair any database satisfying $A^{+?} \wedge \neg A^{+!}$ (respectively $A^{-?} \wedge \neg A^{-!}$) to one satisfying $\neg A^{+?} \wedge A^{+!}$ (respectively $\neg A^{-?} \wedge A^{-!}$). So, starting from the database $\{G^{+?}\}$ which satisfies $G^{+?} \wedge \neg G^{+!}$, if there is a successful dynamic repair procedure using these sets of active constraints then $\neg G^{+?} \wedge G^{+!}$ will be reached and the initial formula G will be proved to be true. On the other hand, if there is no dynamic weak repair of $\{G^{+?}\}$ by these active constraints then G will be false.

In the following we define the set of active integrity constraints for the case when $A = \exists x.B$ that encode the truth conditions that are used in the evaluation of A. The underlined literals highlight the differences between the static constraints while the rightmost column explains the taken action.

a_1: $\langle \neg A^{+?} \vee A^{+!} \vee B^{+?} \vee B^{+!} \vee x^? \vee x^! \vee x, \{x^? \leftarrow \top\} \rangle$, ask for a truth value for x;

a_2: $\langle \neg A^{+?} \vee A^{+!} \vee B^{+?} \vee B^{+!} \vee \underline{\neg x^?} \vee x^! \vee x, \{x \leftarrow \top, x^! \leftarrow \top\} \rangle$, toggle x or set it to false;

a_3: $\langle \neg A^{+?} \vee A^{+!} \vee B^{+?} \vee B^{+!} \vee \neg x^? \vee x^! \vee \underline{\neg x}, \{x^! \leftarrow \top\} \rangle$, set x to true;

a_4: $\langle \neg A^{+?} \vee A^{+!} \vee B^{+?} \vee B^{+!} \vee \neg x^? \vee \underline{\neg x^!}, \{x^? \leftarrow \bot\} \rangle$, end the choice of x;

a_5: $\langle \neg A^{+?} \vee A^{+!} \vee B^{+?} \vee B^{+!} \vee \underline{x^?} \vee \neg x^!, \{B^{+?} \leftarrow \top\} \rangle$, ask for B to be true;

a_6: $\langle \neg A^{+?} \vee A^{+!} \vee B^{+?} \vee \underline{\neg B^{+!}} \vee x^? \vee \neg x^!, \{A^{+!} \leftarrow \top\} \rangle$, A is now proved true;

a_7: $\langle \neg A^{+?} \vee \underline{\neg A^{+!}} \vee B^{+?} \vee \neg B^{+!} \vee x^? \vee \neg x^!, \{x^! \leftarrow \bot\} \rangle$, free x;

a_8: $\langle \neg A^{+?} \vee \neg A^{+!} \vee B^{+?} \vee \neg B^{+!} \vee x^? \vee \underline{x^!}, \{B^{+!} \leftarrow \bot\} \rangle$, remove the result for B;

a_9: $\langle \neg A^{+?} \vee \neg A^{+!} \vee B^{+?} \vee \underline{B^{+!}} \vee x^? \vee x^! \vee \neg x, \{x \leftarrow \bot\} \rangle$, remove x if it is set to true;

a_{10}: $\langle \neg A^{+?} \vee \neg A^{+!} \vee B^{+?} \vee B^{+!} \vee x^? \vee x^! \vee \underline{x}, \{A^{+?} \leftarrow \bot\} \rangle$, end the request for A;

For $A = \exists x.B$ and $V \subseteq \text{subf}^{\vee}(G)$ then the following holds: $V \models A$ iff there is a dynamic weak repair of $V \cup \{A^{+?}\}$ by $\{a_1, \ldots, a_{10}\}$. Similar sets of active constraints then can be assigned to the rest of the formulas $A \in \text{subf}(G)$. Finally, we define the active constraint $g = \langle G^{+?} \vee \neg G^{+!}, \{G^{+!} \leftarrow \bot\}\rangle$ which is used in the final step to ensure minimality of the repair procedure, i.e., in order to ensure that only the update action $G^{+?} \leftarrow \bot$ survives at the end. We then define the set of active constraints η_G to be the union of the aforementioned active constraints for each $A \in \text{subf}(G)$ together with g. Due to the active constraint g, the set $\{G^{+?} \leftarrow \bot\}$ will always be the only dynamic weak repair of $\{G^{+?}\}$ by η_G and, since $\emptyset$ is not a weak repair of $\{G^{+?}\}$ by η_G, it is also a PMA repair. So if a dynamic weak repair of $\{G^{+?}\}$ by η_G exists, it will be the set $\{G^{+?} \leftarrow \bot\}$ which is also a dynamic repair of $\{G^{+?}\}$ by η_G. This gives the following: (1) if there is a dynamic weak repair of $\{G^{+?}\}$ by η_G then there is also a dynamic repair of $\{G^{+?}\}$ by η_G. It follows that:

(2) G is true iff there is a dynamic weak repair of $\{G^{+?}\}$ by η_G

(3) G is true iff there is a dynamic repair of $\{G^{+?}\}$ by η_G

Note that (3) is an immediate outcome of (1) and (2). Now, since it is known that checking whether a fully quantified Boolean formula is true is a PSPACE-complete problem, the hardness result for both dynamic weak repairs and dynamic repairs follows from (2) and (3) and the fact that, given a formula G, the cardinality of the set η_G is linear in the length of G. □

6.2. Upper complexity bound

Next, in order to show that deciding the existence of dynamic weak repairs and dynamic repairs is in PSPACE we just need to reduce the problem into the model checking problem of DL-PA, since the latter is known to be PSPACE-complete [14]. The reduction is easy and is based on the following proposition.

Proposition 6.2. Let η be a set of active integrity constraints in the language of $\mathbb{P}$ and let $V \subseteq \mathbb{P}$ be a database.

- A dynamic weak repair of V by η exists iff $V \models \langle \pi_\eta \rangle \top$, where π_η is the program **while** $\neg (\bigwedge \mathsf{C}(\eta))$ **do** $(\bigcup_{r \in \eta} \pi_r)$
- A dynamic repair of V by η exists iff $V \models \langle \pi_\eta \rangle \top$, where π_η is the program **while** $\neg (\bigwedge \mathsf{C}(\eta))$ **do** $(\bigcup_{r \in \eta} \pi_r^{\pm})\,;\mathsf{Minimal}(\mathsf{C}(\eta))?\,;\mathbb{P}_C{}^{\pm} \leftarrow \bot$

Proof:
In both cases, for the left to right direction consider that a dynamic weak repair (respectively, dynamic repair) of V by η exists. By Definition 5.3 and Theorem 5.7 then, this means that there exists a set of update actions U such that U is relevant w.r.t. V and $\langle V, V \diamond U\rangle \in \|\pi_\eta\|$. Since $V \diamond U \models \top$, the result follows.

Again in both cases, for the right to left direction let $V \models \langle \pi_\eta \rangle \top$. This means that there exists a V' such that $\langle V, V'\rangle \in \|\pi_\eta\|$. It is easy to see that, if we set $U = \{p \leftarrow \top \ : \ p \in V' \text{ and } p \notin V\} \cup \{p \leftarrow \bot \ : \ p \in V \text{ and } p \notin V'\}$ then $V' = V \diamond U$ and U is relevant w.r.t. V. So, in other words, there exists a U

such that U is relevant w.r.t. V and $\langle V, V \diamond U\rangle \in \|\pi_\eta\|$. By Definition 5.3 and Theorem 5.7 then, this means that there exists a dynamic weak repair (respectively, dynamic repair) of V by η. □

The theorem for membership then follows immediately.

Theorem 6.3. The problems of existence of a dynamic weak repair and a dynamic repair are both in PSPACE.

Proof:
It is known that model checking in DL-PA is a PSPACE-complete problem. The result follows from Proposition 6.2 and the fact that, given a set of active integrity constraints η, the length of the program π_η is linear in the size of the set $\mathsf{C}(\eta)$. □

Using Theorems 6.1 and 6.3 then, we have the following corollary.

Corollary 6.4. The problems of existence of a dynamic weak repair and a dynamic repair are both PSPACE-complete.

7. History-based repairs

In this section we would like to discuss *history-based repairs*, an extension of the repairs seen so far taking into account databases with *history*. What we mean by *history* is the consistent set of transactions that took place from the last time the database satisfied the integrity constraints up until its current form. So let's say that apart from the initial database, we are also provided with a consistent set of update actions: these are what we refer to as *history*, the extra information of the route taken from an earlier point in time (more specifically, the last time the integrity constraints were satisfied) until the current state of affairs.

So given this extra information, how should we make use of it? A starting point would be to make sure that the update actions which took place in order to reach the current database will not appear again in the future. Imagine, for instance, that we are provided with the database V, a set of active integrity constraints η and an update action that took place in order to reach it, $p{\leftarrow}\bot$. If there exist two repairs of V by η, namely $U_1 = \{p{\leftarrow}\top\}$ and $U_2 = \{q{\leftarrow}\top\}$, then we would like to disregard U_1 as it would repair V by adding p and V would return to an "earlier state" (from which it was updated by removing p) thus violating the 'priority of the new information' principle that was widely considered in the update literature [26]. Furthermore, we should consider what happens in the case where, although repairs exist, there is no repair that updates the database without returning it to an "earlier state". Should we make use of them and disregard the given history or not? Intuitively, the repair actions of active integrity constraints are of the highest priority when repairing a database, whereas the aforementioned *history* is based on a set of update actions which was used to repair a previous database into the current one, but can be undone if needed. We can see this in the previous example as well, where if U_1 was the only repair of V by η then it should be applied regardless of $p{\leftarrow}\bot$ being used to reach V.

But perhaps a more concrete example is the following, based on the "an employee cannot be in 2 departments" constraint: let $\mathbb{P} = \{d_1, d_2\}$, where d_1 and d_2 denote departments 1 and 2 respectively, and the integrity constraint $r = \langle \neg d_1 \vee \neg d_2, \{d_1 \leftarrow \bot, d_2 \leftarrow \bot\} \rangle$ which says that no employee should work in both departments at the same time; if this is the case, then they should be removed from either one, without any specific preference. Assume now that $H = \langle \{d_1, d_2\}, \{d_1 \leftarrow \top\} \rangle$ is our history-based database, where $\{d_1, d_2\}$ is our actual database V (saying that there is someone working on both departments) and the set $\{d_1 \leftarrow \top\}$ represents the history, i.e., that the last department which they joined was d_1. In this case, we would prefer the repairing of V by $\{d_2 \leftarrow \bot\}$, instead of $\{d_1 \leftarrow \bot\}$, considering the latest action of putting the specific employee recently in department 1 to be of higher priority. This is done by repairing V by $r_H = \langle \neg d_1 \vee \neg d_2, \{d_2 \leftarrow \bot\} \rangle$ instead of r and actually disregarding the "preferred" update action $d_1 \leftarrow \bot$ in $\mathsf{R}(r)$ which conflicts with the provided history.

With that in mind, we call $H = \langle V, U \rangle$ a history-based database when $V \subseteq \mathbb{P}$ is a database and $U \subseteq \mathbb{U}$ a consistent set of update actions such that there is $V' \subseteq \mathbb{P}$ with $V' \diamond U = V$. We also define U^{-1} as the set comprising the opposite update actions in U: $U^{-1} = \{p \leftarrow \bot : p \leftarrow \top \in U\} \cup \{p \leftarrow \top : p \leftarrow \bot \in U\}$.

According to what has been said so far, we would like U' to be a repair of a history-based database $H = \langle V, U \rangle$ by a set of active integrity constraints η, only if $U' \cap U^{-1} = \emptyset$. In order for this to happen we have to repair V by η_H instead of η, where η_H has:

$$r_H = \langle \mathsf{C}(r), \mathsf{R}(r) \setminus U^{-1} \rangle \quad \text{, for } r \in \eta$$

In this way we disregard update actions which have the risk of returning our current database to a previous state and give priority to the new ones. As already mentioned, by choosing to ignore update actions based on the history U that we have, we may risk reducing a set $\mathsf{R}(r)$ to be empty for some $r \in \eta$, thus leading to no repairs occurring. In this case, choosing to not take U into consideration is the only option and we return to the repairing of V by η, instead of H by η. The following example highlights everything that's been said so far.

Example 7.1. Let $H = \langle \{d_1, d_2\}, \{d_1 \leftarrow \top\} \rangle$ and $\eta_1 = \langle \neg d_1 \vee \neg d_2, \{d_1 \leftarrow \bot, d_2 \leftarrow \bot\} \rangle$. Both $U_1 = \{d_1 \leftarrow \bot\}$ and $U_2 = \{d_2 \leftarrow \bot\}$ are founded, justified, dynamic and global-dynamic repairs of $V = \{d_1, d_2\}$ by η_1. Only the second is a founded, justified, dynamic and global-dynamic repair of H by η_1. Similarly, consider $\eta_2 = \langle \neg d_1 \vee \neg d_2, \{d_1 \leftarrow \bot\} \rangle$. Then $U_1 = \{d_1 \leftarrow \bot\}$ is a founded, justified, dynamic and global-dynamic repair of $V = \{d_1, d_2\}$ by η_2. There are no repairs of H by η_2, however, making us reduce H to just V and use U_1 once again.

Finally, let us define:

$$\mathsf{weakRepairH}(\mathsf{C}(\eta), U) = \left(\bigcup_{p \in A} \mathsf{toggle}(p) \right)^* ; \left(\bigwedge \mathsf{C}(\eta) \right) ?$$

$$(\pi^{\pm}_{r,U})' = \neg \mathsf{C}(r) \, ? \, ; \bigcup_{p \leftarrow \mathsf{X} \in B} (p \leftarrow \mathsf{X}; p^{\pm} \leftarrow \top) \qquad \text{and} \qquad (\pi^{\pm}_{r,U})'' = \bigcup_{p \leftarrow \mathsf{X} \in B} (p \leftarrow \mathsf{X}; p^{\pm} \leftarrow \top)$$

where $A = \mathbb{P}_{\mathsf{C}(\eta) \setminus U^{-1}}$ and $B = \mathsf{R}(r) \setminus U^{-1}$.

The next theorem characterises founded, justified, dynamic and global-dynamic history-based repairs in terms of DL-PA programs.

Theorem 7.2. Let η be a set of active integrity constraints in the language of $\mathbb{P}$ and let $V \subseteq \mathbb{P}$ be a database (so no $p^{\pm}, p^{+}$ and p^{-} occurs in either of them). Let U and U' be consistent sets of update actions such that U is relevant w.r.t. V.

- U is a founded repair of $H = \langle V, U' \rangle$ by η iff:

$$\langle V, V \diamond U \rangle \in \Big\| \mathsf{weakRepairH}(\mathsf{C}(\eta), U'); \mathsf{Founded}(\eta)? \,; \mathsf{Minimal}(\mathsf{C}(\eta))? \,; \mathbb{P}_C{}^{\pm} \leftarrow \bot \Big\|$$

- U is a justified repair of $H = \langle V, U' \rangle$ by η iff η is a set of standard active constraints and:

$$\langle V, V \diamond U \rangle \in \Big\| \mathsf{weakRepairH}(\mathsf{C}(\eta), U'); \mathsf{Justified}(\eta)?; \mathsf{Minimal}(\mathsf{C}(\eta))?; \mathsf{ClearAll} \Big\|$$

- U is a dynamic repair of $H = \langle V, U' \rangle$ by η iff:

$$\langle V, V \diamond U \rangle \in \Big\| \textbf{while } \neg \Big(\bigwedge \mathsf{C}(\eta) \Big) \textbf{ do } \Big(\bigcup_{r \in \eta} (\pi^{\pm}_{r,U'})' \Big); \mathsf{Minimal}\Big(\mathsf{C}(\eta)\Big)? \,; \mathbb{P}_C{}^{\pm} \leftarrow \bot \Big\|$$

- U is a global-dynamic repair of $H = \langle V, U' \rangle$ by η iff:

$$\langle V, V \diamond U \rangle \in \Big\| \textbf{while } \neg \Big(\bigwedge \mathsf{C}(\eta) \Big) \textbf{ do } \Big(\bigcup_{r \in \eta} (\pi^{\pm}_{r,U'})'' \Big); \mathsf{Minimal}\Big(\mathsf{C}(\eta)\Big)? \,; \mathbb{P}_C{}^{\pm} \leftarrow \bot \Big\|$$

Proof:
Note that the construction of all history-based repairs in DL-PA terms are identical to their counterparts in the previous sections, with the difference being in the choice of p's and update actions $p \leftarrow \mathsf{X}$ in the nondeterministic composition place of the programs. By removing the set U^{-1} from each nondeterministic choice imposed by the programs $\mathsf{weakRepairH}(\mathsf{C}(\eta), U)$, $(\pi^{\pm}_{r,U})'$ and $(\pi^{\pm}_{r,U})''$ we get exactly the active integrity constraints r_H (instead of r) needed in order to repair a history-based database by the corresponding set of active integrity constraints η_H. □

8. Conclusion

We have shown how several definitions of database repair via active integrity constraints can be expressed in DL-PA, including new proposals in terms of their iterated application. More specifically, we have introduced a new, dynamic way of handling database repair under a set of active integrity constraints and have shown some interesting properties and alternatives by means of history-based repairs, all through the use of the quite simple but expressive Dynamic Logic of Propositional Assignments DL-PA. This allows us to claim that DL-PA is a nice integrated framework for database updates: it not only provides operators $p \leftarrow \top$ of insertion and $p \leftarrow \bot$ of deletion and more generally sets U of such assignments that can be applied to a database V; it also provides a means to reason about the repair of the resulting $V \diamond U$ when some element of the set of integrity constraints is violated.

In the following, the program *repair* denotes one of the repair programs of Theorems 3.1, 4.1, 4.2, 5.7, 7.2, as well as Definitions 5.3 and 5.10. We can witness the aforementioned treatment of DL-PA as a means of reasoning between repairs in the following two instances:

- V' is a possible repair of the update of the database V by the insertion or deletion of p if and only if the couple $\langle V, V' \rangle$ belongs to the interpretation of the DL-PA programs $p{\leftarrow}\top$; *repair* or $p{\leftarrow}\bot$; *repair* respectively.

- The set of candidate repaired databases is the interpretation of the DL-PA formula $\langle repair^{-} \rangle \varphi_V$, where φ_V is a conjunction of literals describing V syntactically.

But beyond identifying possible repaired databases, what is even more interesting is that our programs *repair* also allow to solve decision problems. Some notable examples follow:

- We may check whether it is possible at all to repair V by model checking in DL-PA whether $V \models \langle repair \rangle \top$.

- We can check whether there is a unique repair of V by model checking whether the set of databases V' such that $\langle V, V' \rangle \in \|repair\|$ is a singleton. This amounts to model check for each of the variables p occurring in the constraints whether $V \models [repair]p \vee [repair]\neg p$.

- We might as well wish to check possibility or unicity of the repairs independently of a specific database V. For instance, we can check whether η can repair any database by checking whether the formula $\langle repair \rangle \top$ is DL-PA valid.

- A further interesting reasoning task is to check whether two sets of active constraints η_1 and η_2 are equivalent under a given semantics by checking whether $\|repair_{\eta_1}\| = \|repair_{\eta_2}\|$.

All of the above demonstrate the usefulness of DL-PA as a logic dealing with database repair. The related decision problems also provide a hint of the variety of applications it provides. Furthermore, our way of handling active integrity constraints of the form $r = \langle \mathsf{C}(r), \mathsf{R}(r) \rangle$ allows us to generalise the condition $\mathsf{C}(r)$ from clauses to arbitrary formulas (that could actually even be DL-PA formulas). This opens up two perspectives. First, our definition also covers revision programs [9]; we leave it to future work to establish the exact relationship. Second, we could further generalise the action $\mathsf{R}(r)$ from a set of update actions to arbitrary DL-PA programs. Dynamic repairs would then still make sense, while it is not clear how founded and justified repairs would have to be defined.

Last but not least, although we have argued that there are real world problems, such as the one in the introductory section, where dynamic repairs are preferable over founded or justified repairs, we did see that this comes at a cost: the computational complexity of dynamic repairs is higher. We leave it to future work to explore possible avenues of improving this dynamic behavior, especially in terms of computational resources. A possible extension to a first order setting is also a good avenue for future research, seeing that the nature of the procedure is independent of the propositional setting that we worked on and that it could easily be adapted on higher level formalisms.

Acknowledgements

The work presented in this paper has been partially supported by a Centre International de Mathmatiques et d'Informatique de Toulouse (CIMI Toulouse) doctoral fellowship. We also wish to thank the two anonymous reviewers for their constructive comments which resulted in improving the presentation of the paper.

References

[1] Abiteboul S. Updates, A New Frontier. In: Gyssens M, Paredaens J, Gucht DV (eds.), ICDT'88, 2nd International Conference on Database Theory, Bruges, Belgium, August 31 - September 2, 1988, Proceedings, volume 326 of *Lecture Notes in Computer Science*. Springer. ISBN 3-540-50171-1, 1988 pp. 1–18. doi:10.1007/3-540-50171-1_1.

[2] Ceri S, Fraternali P, Paraboschi S, Tanca L. Automatic Generation of Production Rules for Integrity Maintenance. *ACM Trans. Database Syst.*, 1994. **19**(3):367–422. doi:10.1145/185827.185828.

[3] Bertossi LE. Database Repairing and Consistent Query Answering. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2011. doi:10.2200/S00379ED1V01Y201108DTM020.

[4] Winslett M. Updating Logical Databases. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 1990.

[5] Herzig A, Rifi O. Propositional Belief Base Update and Minimal Change. *Artif. Intell.*, 1999. **115**(1):107–138. doi:10.1016/S0004-3702(99)00072-7.

[6] Chomicki J, Marcinkowski J. Minimal-change integrity maintenance using tuple deletions. *Inf. Comput.*, 2005. **197**(1-2):90–121. doi:10.1016/j.ic.2004.04.007.

[7] Flesca S, Greco S, Zumpano E. Active integrity constraints. In: Moggi E, Warren DS (eds.), Proceedings of the 6th International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming, 24-26 August 2004, Verona, Italy. ACM. ISBN 1-58113-819-9, 2004 pp. 98–107. doi:10.1145/1013963.1013977.

[8] Caroprese L, Greco S, Zumpano E. Active Integrity Constraints for Database Consistency Maintenance. *IEEE Trans. Knowl. Data Eng.*, 2009. **21**(7):1042–1058. doi:10.1109/TKDE.2008.226.

[9] Caroprese L, Truszczynski M. Active integrity constraints and revision programming. *TPLP*, 2011. **11**(6):905–952. doi:10.1017/S1471068410000475.

[10] Cruz-Filipe L. Optimizing Computation of Repairs from Active Integrity Constraints. In: Beierle C, Meghini C (eds.), Foundations of Information and Knowledge Systems - 8th International Symposium, FoIKS 2014, Bordeaux, France, March 3-7, 2014. Proceedings, volume 8367 of *Lecture Notes in Computer Science*. Springer. ISBN 978-3-319-04938-0, 2014 pp. 361–380. doi:10.1007/978-3-319-04939-7_18.

[11] Bogaerts B, Cruz-Filipe L. Fixpoint semantics for active integrity constraints. *Artif. Intell.*, 2018. **255**:43–70. doi:10.1016/j.artint.2017.11.003.

[12] Herzig A, Lorini E, Moisan F, Troquard N. A Dynamic Logic of Normative Systems. In: Walsh T (ed.), IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, 2011. IJCAI/AAAI. ISBN 978-1-57735-516-8, 2011 pp. 228–233. doi:10.5591/978-1-57735-516-8/IJCAI11-049.

[13] Balbiani P, Herzig A, Troquard N. Dynamic Logic of Propositional Assignments: A Well-Behaved Variant of PDL. In: 28th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2013, New Orleans, LA, USA, June 25-28, 2013. IEEE Computer Society. ISBN 978-1-4799-0413-6, 2013 pp. 143–152. doi:10.1109/LICS.2013.20.

[14] Balbiani P, Herzig A, Schwarzentruber F, Troquard N. DL-PA and DCL-PC: model checking and satisfiability problem are indeed in PSPACE. *CoRR*, 2014. **abs/1411.7825**. URL `http://arxiv.org/abs/1411.7825`.

[15] Harel D. Dynamic Logic. In: Gabbay DM, Günthner F (eds.), Handbook of Philosophical Logic, volume II, pp. 497–604. Springer Netherlands, Dordrecht. ISBN 978-94-009-6259-0, 1984. doi:10.1007/978-94-009-6259-0_10.

[16] Harel D, Kozen D, Tiuryn J. Dynamic Logic. MIT Press, 2000.

[17] 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, volume 8761 of *Lecture Notes in Computer Science*. Springer. ISBN 978-3-319-11557-3, 2014 pp. 486–499. doi:10.1007/978-3-319-11558-0_34.

[18] Tiomkin ML, Makowsky JA. Propositional Dynamic Logic with Local Assignments. *Theor. Comput. Sci.*, 1985. **36**:71–87. doi:10.1016/0304-3975(85)90031-3.

[19] van Eijck J. Making Things Happen. *Studia Logica*, 2000. **66**(1):41–58. doi:10.1023/A:1026792711025.

[20] Herzig A. Belief Change Operations: A Short History of Nearly Everything, Told in Dynamic Logic of Propositional Assignments. In: Baral C, Giacomo GD, Eiter T (eds.), Principles of Knowledge Representation and Reasoning: Proceedings of the Fourteenth International Conference, KR 2014, Vienna, Austria, July 20-24, 2014. AAAI Press. ISBN 978-1-57735-657-8, 2014 .

[21] Doutre S, Herzig A, Perrussel L. A Dynamic Logic Framework for Abstract Argumentation. In: Baral C, Giacomo GD, Eiter T (eds.), Principles of Knowledge Representation and Reasoning: Proceedings of the Fourteenth International Conference, KR 2014, Vienna, Austria, July 20-24, 2014. AAAI Press. ISBN 978-1-57735-657-8, 2014 .

[22] Herzig A, de Menezes MV, de Barros LN, Wassermann R. On the revision of planning tasks. In: Schaub T, Friedrich G, O'Sullivan B (eds.), ECAI 2014 - 21st European Conference on Artificial Intelligence, 18-22 August 2014, Prague, Czech Republic - Including Prestigious Applications of Intelligent Systems (PAIS 2014), volume 263 of *Frontiers in Artificial Intelligence and Applications*. IOS Press. ISBN 978-1-61499-418-3, 2014 pp. 435–440. doi:10.3233/978-1-61499-419-0-435.

[23] Herzig A, Lorini E, Maffre F, Schwarzentruber F. Epistemic Boolean Games Based on a Logic of Visibility and Control. In: Kambhampati S (ed.), Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016. IJCAI/AAAI Press. ISBN 978-1-57735-770-4, 2016 pp. 1116–1122.

[24] Cooper MC, Herzig A, Maffre F, Maris F, Régnier P. A Simple Account of Multi-Agent Epistemic Planning. In: Kaminka GA, Fox M, Bouquet P, Hüllermeier E, Dignum V, Dignum F, van Harmelen F (eds.), ECAI 2016 - 22nd European Conference on Artificial Intelligence, 29 August-2 September 2016, The Hague, The Netherlands - Including Prestigious Applications of Artificial Intelligence (PAIS 2016), volume 285 of *Frontiers in Artificial Intelligence and Applications*. IOS Press. ISBN 978-1-61499-671-2, 2016 pp. 193–201. doi:10.3233/978-1-61499-672-9-193.

[25] Charrier T, Schwarzentruber F. A Succinct Language for Dynamic Epistemic Logic. In: Larson K, Winikoff M, Das S, Durfee EH (eds.), Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems, AAMAS 2017, São Paulo, Brazil, May 8-12, 2017. ACM, 2017 pp. 123–131.

[26] Katsuno H, Mendelzon AO. On the difference between updating a knowledge base and revising it. In: Gärdenfors P (ed.), Belief revision, pp. 183–203. Cambridge University Press, 1992. (preliminary version in Allen, J.A., Fikes, R., and Sandewall, E., eds., Principles of Knowledge Representation and Reasoning: Proc. 2nd Int. Conf., pages 387–394. Morgan Kaufmann Publishers, 1991).

[27] Peppas P, Nayak AC, Pagnucco M, Foo NY, Kwok RBH, Prokopenko M. Revision vs. Update: Taking a Closer Look. In: Wahlster W (ed.), 12th European Conference on Artificial Intelligence, Budapest, Hungary, August 11-16, 1996, Proceedings. John Wiley and Sons, Chichester, 1996 pp. 95–99.

[28] Winslett M. Reasoning about Action Using a Possible Models Approach. In: Shrobe HE, Mitchell TM, Smith RG (eds.), Proceedings of the 7th National Conference on Artificial Intelligence, St. Paul, MN, USA, August 21-26, 1988. AAAI Press / The MIT Press. ISBN 0-262-51055-3, 1988 pp. 89–93.

[29] Stockmeyer LJ, Meyer AR. Word Problems Requiring Exponential Time: Preliminary Report. In: Aho AV, Borodin A, Constable RL, Floyd RW, Harrison MA, Karp RM, Strong HR (eds.), Proceedings of the 5th Annual ACM Symposium on Theory of Computing, April 30 - May 2, 1973, Austin, Texas, USA. ACM, 1973 pp. 1–9. doi:10.1145/800125.804029.