Database Repair via Event-Condition-Action Rules in Dynamic Logic

Guillaume Feuillade, Andreas Herzig, Christos Rantsoudis

Database Repair via Event-Condition-Action Rules in Dynamic Logic

Guillaume Feuillade, Andreas Herzig, Christos Rantsoudis

IRIT, Univ. Paul Sabatier, CNRS

Abstract. Event-condition-action (ECA) rules equip a database with information about preferred ways to restore integrity. They face problems of non-terminating executions and only procedural semantics had been given to them up to now. Declarative semantics however exist for a particular class of ECA rules lacking the event argument, called active integrity constraints. We generalise one of these semantics to ECA rules and couch it in a simple dynamic logic with deterministic past.

Keywords: ECA rules, database repair, dynamic logic

1 Introduction

To restore the integrity of a database violating some constraints is an old and notoriously difficult problem. One of the main difficulties is that there are typically several possible repairs: the integrity constraints alone do not provide enough clues which the 'right' repair is. A natural idea is to add more information to the integrity constraints. The most influential proposal was to move from classical, static integrity constraints to so-called Event-Condition-Action (ECA) rules. Such rules indicate how integrity should be restored: when the *event* occurs and the *condition* is satisfied then the *action* is triggered which, intuitively, makes the violating condition false [13,15,4]. A relational database together with a set of ECA rules make up an *active database* [13,24]. Such databases were studied intensely in the database literature in the last four decades. The existing semantics are mainly procedural and chain rule applications: the actions of some ECA rule trigger other ECA rules, and so on. As argued in [7], *"their lack of declarative semantics makes it difficult to understand the behavior of multiple ECAs acting together and to evaluate rule-processing algorithms in a principled way"*. They are moreover plagued by termination problems.

Up to now only few declarative, logical semantics were given to ECA rules. Most of them adopted the logic programming paradigm [23,22,6,19,14]; Bertossi and Pinto considered the Situation Calculus where ECA rules are described in a first-order language plus one second-order axiom and where repairs are performed by means of auxiliary actions [5]. Our aim in this paper is to associate a semantics to ECA rules that is based in dynamic logic. A main reason for our choice is that the latter has built-in constructions allowing us to reason about terminating executions. Another reason is that this allows us to start

from existing declarative semantics for a simplified version of ECA rules. The latter lack the event argument and are simply condition-action couples and were investigated in the database and AI literature since more than 15 years under the denomination *active integrity constraints*, abbreviated AIC [20,10,11,12,7]. AICs *"encode explicitly both an integrity constraint and preferred basic actions to repair it, if it is violated"* [12]. Their semantics notably avoids problems with non-terminating executions that plague ECA rules. Syntactically, an AIC is a couple of the form

$$r = \langle \mathbf{C}(r), \mathbf{A}(r) \rangle$$

where the condition $\mathbf{C}(r)$ is a conjunction of literals $L_1 \wedge \cdots \wedge L_n$ (that we can think of as the negation of a classical, static integrity constraint) and the action $\mathbf{A}(r)$ is a set of assignments of the form either $+p$ or $-p$. It is supposed that each of the elements of $\mathbf{A}(r)$ makes one of the literals false: $+p \in \mathbf{A}(r)$ implies $\neg p \in \mathbf{C}(r)$, and $-p \in \mathbf{A}(r)$ implies $p \in \mathbf{C}(r)$. For instance, the AIC $\langle \texttt{Bachelor} \wedge \texttt{Married}, -\texttt{Bachelor} \rangle$ says that when $\texttt{Bachelor} \wedge \texttt{Married}$ is true then the action $-\texttt{Bachelor}$ should be executed, that is, Bachelor should be deleted. Given a set of AICs, a *repair* of a database $\mathcal{D}$ is a set of AIC actions whose execution produces a database that is consistent with the integrity constraints. Their semantics is clear when there is only one AIC: if the set of AICs is the singleton $R = \{\langle \mathbf{C}(r), \mathbf{A}(r) \rangle\}$ and $\mathcal{D} \models \mathbf{C}(r)$ then each assignment $\alpha \in \mathbf{A}(r)$ is a possible repair of $\mathcal{D}$. When there are several AICs then their actions may interfere in complex ways, just as ECA rules do: the execution of $\mathbf{A}(r)$ makes $\mathbf{C}(r)$ false but may have the side effect that the constraint part $\mathbf{C}(r')$ of another AIC r' that was false before the execution becomes satisfied, and so on. The consistency restoring actions should therefore be arranged in a way such that overall consistency is obtained. Moreover, the database should be changed minimally only. Several semantics achieving this were designed, among which preferred, founded, and justified repairs [12] as well as more recent prioritised versions [8]. Implementations of such repairs were also studied, and the account was extended in order to cope with various applications [17,16,18].

In this paper we examine whether and how the existing logical semantics of AICs can be generalised to ECA rules. For that reason (and also because the term 'rule' has a procedural connotation) we will henceforth use the term *ECA constraints* instead of ECA rules. Syntactically, we consider ECA constraints of the form

$$\kappa = \langle \mathbf{E}(\kappa), \mathbf{C}(\kappa), \mathbf{A}(\kappa) \rangle$$

where $\langle \mathbf{C}(\kappa), \mathbf{A}(\kappa) \rangle$ is an AIC and $\mathbf{E}(\kappa)$ is a boolean formula built from assignments of the form either $+p$ or $-p$. The latter describes the assignments that must have occurred for the AIC to trigger. Our convention is to call *events* assignments that happened in the past and led to a violation state, and to call *actions* assignments to be performed in the future in order to repair the violation.

Example 1. Let $\mathtt{em}_{e,d}$ stand for "e is an employee of department d". The set of ECA constraints

$$\mathcal{K}_{\mathtt{em}} = \big\{ \langle +\mathtt{em}_{e,d_1}, \mathtt{em}_{e,d_1} \wedge \mathtt{em}_{e,d_2}, \{-\mathtt{em}_{e,d_2}\}\rangle, \\ \langle +\mathtt{em}_{e,d_2}, \mathtt{em}_{e,d_1} \wedge \mathtt{em}_{e,d_2}, \{-\mathtt{em}_{e,d_1}\}\rangle \big\}$$

implements a 'priority to the input' policy repairing the functionality constraint for the employee relation: when the integrity constraint $\neg\mathtt{em}_{e,d_1} \vee \neg\mathtt{em}_{e,d_2}$ is violated and $\mathtt{em}_{e,d_1}$ was made true in the last update action then the latter is retained and $\mathtt{em}_{e,d_2}$ is made false in order to enforce the constraint; symmetrically, $\mathtt{em}_{e,d_1}$ is made false when $+\mathtt{em}_{e,d_2}$ is part of the the last update action.

Observe that if the last update action of our example made neither $\mathtt{em}_{e,d_1}$ nor $\mathtt{em}_{e,d_2}$ true, i.e., if $\neg{+}\mathtt{em}_{e,d_1} \wedge \neg{+}\mathtt{em}_{e,d_2}$ holds, then the database already violated the integrity constraints before the last update action. In such cases ECA constraints are of no help: the violation should have been repaired earlier. Moreover, if the last update action of our example made $\mathtt{em}_{e,d_1}$ and $\mathtt{em}_{e,d_2}$ simultaneously true, i.e., if $+\mathtt{em}_{e,d_1} \wedge +\mathtt{em}_{e,d_2}$ holds, then the ECA constraints of $\mathcal{K}_{\mathtt{em}}$ authorise both $-\mathtt{em}_{e,d_1}$ and $-\mathtt{em}_{e,d_2}$ as legal repairs.

A logical account of ECA constraints not only requires reasoning about integrity constraints and actions to repair them when violated, but also reasoning about the event bringing about the violation. Semantically, we have to go beyond the models of AIC-based repairs, which are simply classical valuations (alias databases): in order to account for the last update action we have to add information about the events that led to the present database state. Our models are therefore couples made up of a classical valuation $\mathcal{D}$ and a set of assignments $\mathcal{H}$. The intuition is that the assignments in $\mathcal{H}$ are among the set of assignments that led to $\mathcal{D}$; that is, $\mathcal{H}$ is part of the last update action that took place and brought the database into its present state $\mathcal{D}$. For these models, we show that the AIC definitions of founded repair and well-founded repair can be generalised in a natural way to ECA constraints. We then provide a dynamic logic analysis of ECA constraints. More precisely, we resort to a dialect of dynamic logic: Dynamic Logic of Propositional Assignments DL-PA [3,2]. In order to take events into account we have to extend the basic logic by connectives referring to a deterministic past. We are going to show that this can be done without harm, i.e., without modifying the formal properties of the logic; in particular, satisfiability, validity and model checking stay in PSPACE. Our extension is appropriate to reason about ECA-based repairs: for several definitions of ECA repairs we give DL-PA programs whose executability characterises the existence of a repair (theorems 1-4). Furthermore, several other interesting reasoning problems can be expressed in DL-PA, such as uniqueness of a repair or equivalence of two different sets of constraints.

Just as most of the papers in the AIC literature we rely on grounding and restrict our presentation to the propositional case. This simplifies the presentation and allows us to abstract away from orthogonal first-order aspects. A full-fledged analysis would require a first-order version of DL-PA, which is something that has not been studied yet.

The paper is organised as follows. In Section 2 we recall AICs and their semantics. We then define ECA constraints (Section 3) and their semantics in terms of databases with histories (Section 4). In Section 5 we introduce a version of dynamic logic with past and in Section 6 we capture well-founded ECA repairs in DL-PA. In Section 7 we show that other interesting decision problems can be expressed. Section 8 concludes. Proofs are contained in the long version.[1]

2 Background: AICs and their Semantics

We suppose given a set of propositional variables $\mathbb{P}$ with typical elements $p, q, \ldots$ Just as most of the papers in the AIC literature we suppose that $\mathbb{P}$ is finite. A *literal* is an element of $\mathbb{P}$ or a negation thereof. A *database*, alias a *valuation*, is a set of propositional variables $\mathcal{D} \subseteq \mathbb{P}$.

An *assignment* is of the form either $+p$ or $-p$, for $p \in \mathbb{P}$. The former sets p to true and the latter sets p to false. We use α, α',... for assignments. For every α, the assignment $\bar{\alpha}$ is the opposite assignment, formally defined by $\overline{+p} = -p$ and $\overline{-p} = +p$. The set of all assignments is

$$\mathbb{A} = \{+p \ : \ p \in \mathbb{P}\} \cup \{-p \ : \ p \in \mathbb{P}\}.$$

An *update action* is some subset of $\mathbb{A}$. For every update action $A \subseteq \mathbb{A}$ we define the set of variables it makes true and the set of variables it makes false:

$$A^+ = \{p \ : \ +p \in A\},$$
$$A^- = \{p \ : \ -p \in A\}.$$

An update action A is *consistent* if $A^+ \cap A^- = \emptyset$. It is *relevant w.r.t. a database* $\mathcal{D}$ if for every $p \in \mathbb{P}$, if $+p \in A$ then $p \notin \mathcal{D}$ and if $-p \in A$ then $p \in \mathcal{D}$. Hence relevance implies consistency. The *update* of a database $\mathcal{D}$ by an update action A is a partial function $\circ$ that is defined if and only if A is consistent, and if so then

$$\mathcal{D} \circ A = (\mathcal{D} \setminus A^-) \cup A^+.$$

Proposition 1. *Let $A = \{\alpha_1, \ldots, \alpha_n\}$ be a consistent update action. Then $\mathcal{D} \circ A = \mathcal{D} \circ \{\alpha_1\} \circ \cdots \circ \{\alpha_n\}$, for every ordering of the α_i.*

An *active integrity constraint* (AIC) is a couple $r = \langle \mathbf{C}(r), \mathbf{A}(r) \rangle$ where $\mathbf{C}(r)$ is a conjunction of literals and $\mathbf{A}(r) \subseteq \mathbb{A}$ is an update action such that if $+p \in \mathbf{A}(r)$ then $\neg p \in \mathbf{C}(r)$ and if $-p \in \mathbf{A}(r)$ then $p \in \mathbf{C}(r)$. The condition $\mathbf{C}(r)$ can be thought of as the negation of an integrity constraint. For example, in $r = \langle \texttt{Bachelor} \wedge \texttt{Married}, -\texttt{Bachelor} \rangle$ the first element is the negation of the integrity constraint $\neg\texttt{Bachelor} \vee \neg\texttt{Married}$ and the second element indicates the way its violation should be repaired, namely by deleting `Bachelor`.

[1] https://www.irit.fr/~Andreas.Herzig/P/Foiks22Long.pdf

In the rest of the section we recall several definitions of repairs via a given set of AICs R. The formula

$$\mathsf{OK}(R) = \bigwedge_{r \in R} \neg \mathbf{C}(r)$$

expresses that none of the AICs in R is applicable: all the static constraints hold. When $\mathsf{OK}(R)$ is false then the database has to be repaired.

The first two semantics do not make any use of the active part of AICs. First, an update action A is a *weak repair* of a database $\mathcal{D}$ via a set of AICs R if it is relevant w.r.t. $\mathcal{D}$ and $\mathcal{D} \circ A \models \mathsf{OK}(R)$. The latter means that A makes the static constraints true. Second, A is a *minimal repair* of $\mathcal{D}$ via R if it is a weak repair that is set inclusion minimal: there is no weak repair A' of $\mathcal{D}$ via R such that $A' \subset A$.[2] The remaining semantics all require that each of the assignments of an update action is supported by an AIC in some way.

First, an update action A is a *founded weak repair* of a database $\mathcal{D}$ via a set of AICs R if it is a weak repair of $\mathcal{D}$ via R and for every $\alpha \in A$ there is an $r \in R$ such that

- $\alpha \in \mathbf{A}(r)$,
- $\mathcal{D} \circ (A \setminus \{\alpha\}) \models \mathbf{C}(r)$.

Second, A is a *founded repair* if it is both a founded weak repair and a minimal repair [10].[3]

A weak repair A of $\mathcal{D}$ via R is *well-founded* [7] if there is a sequence $\langle \alpha_1, \ldots, \alpha_n \rangle$ such that $A = \{\alpha_1, \ldots, \alpha_n\}$ and for every α_i, $1 \leq i \leq n$, there is an $r_i \in R$ such that

- $\alpha_i \in \mathbf{A}(r_i)$,
- $\mathcal{D} \circ \{\alpha_1, \ldots, \alpha_{i-1}\} \models \mathbf{C}(r_i)$.

A *well-founded repair* is a well-founded weak repair that is also a minimal repair.[4]

For example, for $\mathcal{D} = \emptyset$ and $R = \{\langle \neg p \wedge \neg q, \{+q\} \rangle\}$ any update action $A \subseteq \{+p \ : \ p \in \mathbb{P}\}$ containing $+p$ or $+q$ is a weak repair of $\mathcal{D}$ via R; the minimal repairs of $\mathcal{D}$ via R are $\{+p\}$ and $\{+q\}$; and the only founded and well-founded repair of $\mathcal{D}$ via R is $\{+q\}$. An example where founded repairs and well-founded repairs behave differently is $\mathcal{D} = \emptyset$ and $R = \{r_1, r_2, r_3\}$ with

$$r_1 = \langle \neg p \wedge \neg q, \emptyset \rangle,$$
$$r_2 = \langle \neg p \wedge q, \{+p\} \rangle,$$
$$r_3 = \langle p \wedge \neg q, \{+q\} \rangle.$$

[2] Most papers in the AIC literature call these simply *repairs*, but we prefer our denomination because it avoids ambiguities.

[3] Note that a founded repair is different from a minimal founded weak repair: whereas the latter is guaranteed to exist in case a founded weak repair does, the former is not; in other words, it may happen that founded weak repairs exist but founded repairs do not (because none of the founded weak repairs happens to also be a minimal repair in the traditional sense).

[4] Again, the existence of a well-founded weak repair does not guarantee the existence of a well-founded repair.

Then $A = \{+p, +q\}$ is a founded repair: the third AIC supports $+q$ because $\mathcal{D} \circ \{+p\} \models \mathbf{C}(r_3)$; and the second AIC supports $+p$ because $\mathcal{D} \circ \{+q\} \models \mathbf{C}(r_2)$. However, there is no well-founded weak repair because the only applicable rule from $\mathcal{D}$ (viz. the first one) has an empty active part.

Two other definitions of repairs are prominent in the literature. *Grounded repairs* generalise founded repairs [7]. As each of them is both a founded and well-founded repair, we skip this generalisation. We also do not present *justified repairs* [12]: their definition is the most complex one and it is not obvious how to transfer it to ECA constraints. Moreover, as argued in [7] they do not provide the intuitive repairs at least in some cases.

3 ECA Constraints

We consider two kinds of *boolean formulas*. The first, static language $\mathcal{L}_{\mathbb{P}}$ is built from the variables in $\mathbb{P}$ and describes the condition part of ECAs. The second, dynamic language $\mathcal{L}_{\mathbb{A}}$ is built from the set of assignments $\mathbb{A}$ and describes the event part of ECAs, i.e., the last update action that took place. The grammars for the two languages are therefore

$$\mathcal{L}_{\mathbb{P}} : \varphi ::= p \mid \neg\varphi \mid \varphi \vee \varphi,$$
$$\mathcal{L}_{\mathbb{A}} : \varphi ::= \alpha \mid \neg\varphi \mid \varphi \vee \varphi,$$

where p ranges over $\mathbb{P}$ and α over the set of all assignments $\mathbb{A}$. We call the elements of $\mathbb{A}$ *history atoms*.

An *event-condition-action (ECA) constraint* combines an event description in $\mathcal{L}_{\mathbb{A}}$, a condition description in $\mathcal{L}_{\mathbb{P}}$, and an update action in $2^{\mathbb{A}}$: it is a triple

$$\kappa = \langle \mathbf{E}(\kappa), \mathbf{C}(\kappa), \mathbf{A}(\kappa) \rangle$$

where $\mathbf{E}(\kappa) \in \mathcal{L}_{\mathbb{A}}$, $\mathbf{C}(\kappa) \in \mathcal{L}_{\mathbb{P}}$, and $\mathbf{A}(\kappa) \subseteq \mathbb{A}$. The event part of an ECA constraint describes the last update action. It does so only partially: only events that are relevant for the triggering of the rule are described.[5]

Sets of ECA constraints are noted $\mathcal{K}$. We take over from AICs the formula $\mathsf{OK}(\mathcal{K})$ expressing that none of the ECA constraints is applicable:

$$\mathsf{OK}(\mathcal{K}) = \bigwedge_{\kappa \in \mathcal{K}} \neg \mathbf{C}(\kappa).$$

Example 2. [20, Example 4.6] Every manager of a project carried out by a department must be an employee of that department; if employee e just became the manager of project p or if the project was just assigned to department d_1 then the constraint should be repaired by making e a member of d_1. Moreover, if e has just been removed from d_1 then the project should either be removed from d_1, too, or should get a new manager. Together with the ECA version of

[5] We observe that in the literature, $\mathbf{C}(\kappa)$ is usually restricted to conjunctions of literals. Our account however does not require this.

the functionality constraint on the $\mathtt{em}$ relation of Example 1 we obtain the set of ECA constraints

$$\begin{aligned}\mathcal{K}_{\mathtt{mg}} = \big\{\ & \langle +\mathtt{em}_{e,d_1}, \mathtt{em}_{e,d_1} \wedge \mathtt{em}_{e,d_2}, \{-\mathtt{em}_{e,d_2}\}\rangle,\\ & \langle +\mathtt{em}_{e,d_2}, \mathtt{em}_{e,d_1} \wedge \mathtt{em}_{e,d_2}, \{-\mathtt{em}_{e,d_1}\}\rangle,\\ & \langle +\mathtt{mg}_{e,p} \vee +\mathtt{pr}_{p,d_1}, \mathtt{mg}_{e,p} \wedge \mathtt{pr}_{p,d_1} \wedge \neg\mathtt{em}_{e,d_1}, \{+\mathtt{em}_{e,d_1}\}\rangle,\\ & \langle -\mathtt{em}_{e,d_1}, \mathtt{mg}_{e,p} \wedge \mathtt{pr}_{p,d_1} \wedge \neg\mathtt{em}_{e,d_1}, \{-\mathtt{mg}_{e,p}, -\mathtt{pr}_{p,d_1}\}\rangle\ \big\}.\end{aligned}$$

Example 3. Suppose some IoT device functions ($\mathtt{fun}$) if and only if the battery is loaded ($\mathtt{bat}$) and the wiring is in order ($\mathtt{wire}$); this is captured by the equivalence $\mathtt{fun} \leftrightarrow (\mathtt{bat} \wedge \mathtt{wire})$, or, equivalently, by the three implications $\mathtt{fun} \rightarrow \mathtt{bat}$, $\mathtt{fun} \rightarrow \mathtt{wire}$, and $(\mathtt{bat} \wedge \mathtt{wire}) \rightarrow \mathtt{fun}$. If we assume a 'priority to the input' repair strategy then the set of ECA constraints has to be

$$\begin{aligned}\mathcal{K}_{\mathtt{iot}} = \big\{\ & \langle +\mathtt{fun}, \mathtt{fun} \wedge \neg\mathtt{bat}, \{+\mathtt{bat}\}\rangle,\\ & \langle -\mathtt{bat}, \mathtt{fun} \wedge \neg\mathtt{bat}, \{-\mathtt{fun}\}\rangle,\\ & \langle +\mathtt{fun}, \mathtt{fun} \wedge \neg\mathtt{wire}, \{+\mathtt{wire}\}\rangle,\\ & \langle -\mathtt{wire}, \mathtt{fun} \wedge \neg\mathtt{wire}, \{-\mathtt{fun}\}\rangle,\\ & \langle +\mathtt{bat}, \mathtt{bat} \wedge \mathtt{wire} \wedge \neg\mathtt{fun}, \{+\mathtt{fun}\}\rangle,\\ & \langle +\mathtt{wire}, \mathtt{bat} \wedge \mathtt{wire} \wedge \neg\mathtt{fun}, \{+\mathtt{fun}\}\rangle,\\ & \langle -\mathtt{fun}, \mathtt{bat} \wedge \mathtt{wire} \wedge \neg\mathtt{fun}, \{-\mathtt{bat}, -\mathtt{wire}\}\rangle\ \big\}.\end{aligned}$$

Observe that when the last update action made $\mathtt{bat}$ and $\mathtt{wire}$ simultaneously true then $\mathcal{K}_{\mathtt{iot}}$ allows to go either way. We can exclude this and force the repair to fail in that case by refining the last three ECA constraints to

$$\begin{aligned}&\langle +\mathtt{bat} \wedge +\mathtt{wire} \wedge -\mathtt{fun}, \mathtt{bat} \wedge \mathtt{wire} \wedge \neg\mathtt{fun}, \emptyset\rangle,\\ &\langle +\mathtt{bat} \wedge \neg -\mathtt{fun}, \mathtt{bat} \wedge \mathtt{wire} \wedge \neg\mathtt{fun}, \{+\mathtt{fun}\}\rangle,\\ &\langle +\mathtt{wire} \wedge \neg -\mathtt{fun}, \mathtt{bat} \wedge \mathtt{wire} \wedge \neg\mathtt{fun}, \{+\mathtt{fun}\}\rangle,\\ &\langle +\mathtt{bat} \wedge \neg +\mathtt{wire} \wedge -\mathtt{fun}, \mathtt{bat} \wedge \mathtt{wire} \wedge \neg\mathtt{fun}, \{-\mathtt{wire}\}\rangle,\\ &\langle +\mathtt{wire} \wedge \neg +\mathtt{bat} \wedge -\mathtt{fun}, \mathtt{bat} \wedge \mathtt{wire} \wedge \neg\mathtt{fun}, \{-\mathtt{bat}\}\rangle.\end{aligned}$$

4 Semantics for ECA Constraints

As ECA constraints refer to the past, their semantics requires more than just valuations, alias databases: we have to add the immediately preceding update action that caused the present database state. Based on such models, we examine several definitions of AIC-based repairs in view of extending them to ECA constraints.

4.1 Databases with History

A *database with history*, or h-database for short, is a couple $\Delta = \langle \mathcal{D}, \mathcal{H}\rangle$ made up of a valuation $\mathcal{D} \subseteq \mathbb{P}$ and an update action $\mathcal{H} \subseteq \mathbb{A}$ such that $\mathcal{H}^+ \subseteq \mathcal{D}$ and

$\mathcal{H}^- \cap \mathcal{D} = \emptyset$. The intuition is that $\mathcal{H}$ is the most recent update action that brought the database into the state $\mathcal{D}$. This explains the two above conditions: they guarantee that if $+p$ is among the last assignments in $\mathcal{H}$ then $p \in \mathcal{D}$ and if $-p$ is among the last assignments in $\mathcal{H}$ then $p \notin \mathcal{D}$. Note that thanks to this constraint $\mathcal{H}$ is consistent: $\mathcal{H}^+$ and $\mathcal{H}^-$ are necessarily disjoint.

The *update* of a history $\mathcal{H}$ by an event A is defined as

$$\mathcal{H} \circ A = A \cup \{\alpha \in \mathcal{H} \ : \ A \cup \{\alpha\} \text{ is consistent}\}.$$

For example, $\{+p, +q\} \circ \{-p\} = \{-p, +q\}$. The history $\mathcal{H} \circ A$ is consistent if and only if A and $\mathcal{H}$ are both consistent.

Let $\mathbb{M}$ be the class of all couples $\Delta = \langle \mathcal{D}, \mathcal{H} \rangle$ with $\mathcal{D} \subseteq \mathbb{P}$ and $\mathcal{H} \subseteq \mathbb{A}$ such that $\mathcal{H}^+ \subseteq \mathcal{D}$ and $\mathcal{H}^- \cap \mathcal{D} = \emptyset$. We interpret the formulas of $\mathcal{L}_{\mathbb{P}}$ in $\mathcal{D}$ and those of $\mathcal{L}_{\mathbb{A}}$ in $\mathcal{H}$, in the standard way. For example, consider the h-database $\langle \mathcal{D}, \mathcal{H} \rangle$ with $\mathcal{D} = \{\mathtt{em}_{e,d_1}, \mathtt{em}_{e,d_2}\}$ and $\mathcal{H} = \{+\mathtt{em}_{e,d_1}\}$. Then we have $\mathcal{D} \models \mathtt{em}_{e,d_1} \wedge \neg \mathtt{em}_{e,d_3}$ and $\mathcal{H} \models +\mathtt{em}_{e,d_1} \wedge \neg +\mathtt{em}_{e,d_2}$.

Now that we can interpret the event part and the condition part of an ECA constraint, it remains to interpret the action part. This is more delicate and amounts to designing the semantics of repairs based on ECA constraints. In the rest of the section we examine several possible definitions. For a start, we take over the two most basic definitions of repairs from AICs: weak repairs and minimal repairs of an h-database $\langle \mathcal{D}, \mathcal{H} \rangle$ via a set of ECA constraints $\mathcal{K}$.

- An update action A is a *weak repair* of $\langle \mathcal{D}, \mathcal{H} \rangle$ via $\mathcal{K}$ if A is relevant w.r.t. $\mathcal{D}$ and $\mathcal{D} \circ A \models \mathsf{OK}(\mathcal{K})$;
- An update action A is a *minimal repair* of $\langle \mathcal{D}, \mathcal{H} \rangle$ via $\mathcal{K}$ if A is a weak repair that is set inclusion minimal: there is no weak repair A' of $\langle \mathcal{D}, \mathcal{H} \rangle$ via $\mathcal{K}$ such that $A' \subset A$.

4.2 Founded and Well-Founded ECA Repairs

A straightforward adaption of founded repairs to ECA constraints goes as follows. Suppose given a candidate repair $A \subseteq \mathbb{A}$. For an ECA constraint κ to support an assignment $\alpha \in A$ given an h-database $\langle \mathcal{D}, \mathcal{H} \rangle$, the constraint $\mathbf{E}(\kappa)$ about the immediately preceding event should be satisfied by $\mathcal{H}$ together with the rest of changes imposed by A, i.e., we should also have $\mathcal{H} \circ (A \setminus \{\alpha\}) \models \mathbf{E}(\kappa)$.[6] This leads to the following definition: a weak repair A is a *founded weak ECA repair* of $\langle \mathcal{D}, \mathcal{H} \rangle$ via a set of ECAs $\mathcal{K}$ if for every $\alpha \in A$ there is a $\kappa \in \mathcal{K}$ such that

- $\alpha \in \mathbf{A}(\kappa)$,
- $\mathcal{D} \circ (A \setminus \{\alpha\}) \models \mathbf{C}(\kappa)$,
- $\mathcal{H} \circ (A \setminus \{\alpha\}) \models \mathbf{E}(\kappa)$.

[6] A naive adaption would only require $\mathcal{H} \models \mathbf{E}(\kappa)$. However, the support for α would be much weaker; see also the example below.

Once again, a *founded ECA repair* is a founded weak ECA repair that is also a minimal repair.

Moving on to well-founded repairs, an appropriate definition of ECA-based repairs should not only check the condition part of constraints in a sequential way, but should also do so for their triggering event part. Thus we get the following definition: a weak repair A of $\langle \mathcal{D}, \mathcal{H} \rangle$ via the set of ECA constraints $\mathcal{K}$ is a *well-founded weak ECA repair* if there is a sequence of assignments $\langle \alpha_1, \ldots, \alpha_n \rangle$ such that $A = \{\alpha_1, \ldots, \alpha_n\}$ and such that for every α_i, $1 \leq i \leq n$, there is a $\kappa_i \in \mathcal{K}$ such that

- $\alpha_i \in \mathbf{A}(\kappa_i)$,
- $\mathcal{D} \circ \{\alpha_1, \ldots, \alpha_{i-1}\} \models \mathbf{C}(\kappa_i)$,[7]
- $\mathcal{H} \circ \{\alpha_1\} \circ \cdots \circ \{\alpha_{i-1}\} \models \mathbf{E}(\kappa_i)$.

As always, a *well-founded ECA repair* is defined as a well-founded weak ECA repair that is also a minimal repair.

Example 4 (Example 2, ctd.). Consider the ECA constraints $\mathcal{K}_{\mathtt{mg}}$ of Example 2 and the h-database $\langle \mathcal{D}, \mathcal{H} \rangle$ with $\mathcal{D} = \{\mathtt{mg}_{e,p}, \mathtt{pr}_{p,d_1}, \mathtt{em}_{e,d_2}\}$ and $\mathcal{H} = \{+\mathtt{mg}_{e,p}\}$, that is, e just became manager of project p. There is only one intended repair: $A = \{+\mathtt{em}_{e,d_1}, -\mathtt{em}_{e,d_2}\}$. Based on the above definitions it is easy to check that A is a founded ECA repair (both assignments in A are sufficiently supported by $\mathcal{D}$ and $\mathcal{H}$) as well as a well-founded ECA repair of $\langle \mathcal{D}, \mathcal{H} \rangle$ via $\mathcal{K}_{\mathtt{mg}}$.

Remark 1. A well-founded weak ECA repair of $\langle \mathcal{D}, \mathcal{H} \rangle$ via $\mathcal{K}$ is also a weak repair. It follows that $\mathcal{D} \circ A \models \mathsf{OK}(\mathcal{K})$, i.e., the repaired database satisfies the static integrity constraints.

In the rest of the paper we undertake a formal analysis of ECA repairs in a version of dynamic logic. The iteration operator of the latter allows us to get close to the procedural semantics while discarding infinite runs. We follow the arguments in [7] against founded and justified AICs and focus on well-founded ECA repairs.

5 Dynamic Logic of Propositional Assignments with Deterministic Past

In the models of Section 4 we can actually interpret a richer language that is made up of boolean formulas built from $\mathbb{P} \cup \mathbb{A}$. We further add to that hybrid language modal operators indexed by programs. The resulting logic DL-PA$^{\pm}$ extends Dynamic Logic of Propositional Assignments DL-PA [3,2].

[7] This is equivalent to $\mathcal{D} \circ \{\alpha_1\} \circ \cdots \circ \{\alpha_{i-1}\} \models \mathbf{C}(\kappa_i)$ because A is consistent (cf. Proposition 1).

5.1 Language of DL-PA$^{\pm}$

The hybrid language $\mathcal{L}_{\text{DL-PA}^{\pm}}$ of DL-PA with past has two kinds of atomic formulas: propositional variables of the form p and history atoms of the form $+p$ and $-p$. Moreover, it has two kinds of expressions: formulas, noted φ, and programs, noted π. It is defined by the following grammar

$$\varphi ::= p \mid +p \mid -p \mid \neg\varphi \mid \varphi \vee \varphi \mid \langle\pi\rangle\varphi,$$
$$\pi ::= A \mid \pi;\pi \mid \pi \cup \pi \mid \pi^* \mid \varphi?,$$

where p ranges over $\mathbb{P}$ and A over subsets of $\mathbb{A}$. The formula $\langle\pi\rangle\varphi$ combines a formula φ and a program π and reads "π can be executed and the resulting database satisfies φ". Programs are built from sets of assignments by means of the program operators of PDL: $\pi_1;\pi_2$ is sequential composition, $\pi_1 \cup \pi_2$ is nondeterministic composition, π^* is unbounded iteration, and $\varphi?$ is test. Note that the expression $+p$ is a formula while the expression $\{+p\}$ is a program. The program $\{+p\}$ makes p true, while the formula $+p$ expresses that p has just been made true. The languages $\mathcal{L}_{\mathbb{P}}$ and $\mathcal{L}_{\mathbb{A}}$ of Section 3 are both fragments of the language of DL-PA$^{\pm}$.

For a set of propositional variables $P = \{p_1, \ldots, p_n\}$, we abbreviate the atomic program $\{-p_1, \ldots, -p_n\}$ by $-P$. The program π^+ abbreviates $\pi;\pi^*$. For $n \geq 0$, we define n-ary sequential composition $;_{i=1,\ldots,n}\,\pi_i$ by induction on n

$$;_{i=1,\ldots,0}\,\pi_i = \top?,$$
$$;_{i=1,\ldots,n+1}\,\pi_i = (;_{i=1,\ldots,n}\,\pi_i);\pi_{n+1}.$$

A particular case of such arbitrary sequences is n-times iteration of π, expressed as $\pi^n = ;_{i=1,\ldots,n}\,\pi$. We also define $\pi^{\leq n}$ as $(\top?\cup\pi)^n$. (It could as well be defined as $\bigcup_{0\leq i\leq n}\pi^i$.) Finally and as usual in dynamic logic, the formula $[\pi]\varphi$ abbreviates $\neg\langle\pi\rangle\neg\varphi$.

Given a program π, $\mathbb{P}_\pi$ is the set of propositional variables occurring in π. For example, $\mathbb{P}_{+p\wedge\langle -q\rangle\neg -r?} = \{p, q, r\}$.

5.2 Semantics of DL-PA$^{\pm}$

The semantics is the same as that of ECA constraints, namely in terms of databases with history. The interpretation of a formula is a subset of the set of all h-databases $\mathbb{M}$, and the interpretation of a program is a relation on $\mathbb{M}$. More precisely, the interpretation of formulas is

$$\begin{aligned}
\Delta \models p &\quad \text{if} \quad \Delta = \langle\mathcal{D}, \mathcal{H}\rangle \text{ and } p \in \mathcal{D},\\
\Delta \models +p &\quad \text{if} \quad \Delta = \langle\mathcal{D}, \mathcal{H}\rangle \text{ and } +p \in \mathcal{H},\\
\Delta \models -p &\quad \text{if} \quad \Delta = \langle\mathcal{D}, \mathcal{H}\rangle \text{ and } -p \in \mathcal{H},\\
\Delta \models \langle\pi\rangle\varphi &\quad \text{if} \quad \Delta||\pi||\Delta' \text{ and } \Delta' \models \varphi \text{ for some } \Delta',
\end{aligned}$$

and as expected for the boolean connectives; the interpretation of programs is

$$\begin{aligned}
\Delta||A||\Delta' &\text{ if } A \text{ is consistent, } \Delta = \langle \mathcal{D}, \mathcal{H}\rangle, \text{ and } \Delta' = \langle \mathcal{D} \circ A, \mathcal{H} \circ A\rangle,\\
\Delta||\pi_1;\pi_2||\Delta' &\text{ if } \Delta||\pi_1||\Delta'' \text{ and } \Delta''||\pi_2||\Delta' \text{ for some } \Delta'',\\
\Delta||\pi_1 \cup \pi_2||\Delta' &\text{ if } \Delta||\pi_1||\Delta' \text{ or } \Delta||\pi_2||\Delta',\\
\Delta||\pi^*||\Delta' &\text{ if } \Delta||\pi^n||\Delta' \text{ for some } n \geq 0,\\
\Delta||\varphi?||\Delta' &\text{ if } \Delta \models \varphi \text{ and } \Delta' = \Delta.
\end{aligned}$$

Hence the interpretation of a set of assignments A updates both the database $\mathcal{D}$ and the history $\mathcal{H}$ by A.[8]

We say that a program π is *executable* at an h-database Δ if $\Delta \models \langle\pi\rangle\top$, i.e., if there is an h-database Δ' such that $\Delta||\pi||\Delta'$. Clearly, consistency of $A \subseteq \mathbb{A}$ is the same as executability of the program A at every h-database Δ.

The definitions of validity and satisfiability in the class of models $\mathbb{M}$ are standard. For example, $+p \rightarrow p$ and $-p \rightarrow \neg p$ are both valid while the other direction is not, i.e., $p \wedge \neg{+p}$ and $\neg p \wedge \neg{-p}$ are both satisfiable.

Proposition 2. *The decision problems of DL-PA$^{\pm}$ satisfiability, validity and model checking are all PSPACE complete.*

6 Well-Founded ECA Repairs in DL-PA$^{\pm}$

We now show that well-founded weak ECA repairs and well-founded ECA repairs can be captured in DL-PA$^{\pm}$.

6.1 Well-Founded Weak ECA Repairs

Given a set of ECA constraints $\mathcal{K}$, our translation of well-founded weak ECA repairs uses fresh auxiliary propositional variables $\mathsf{D}(\alpha)$ recording that $\alpha \in \mathbb{A}$ has been executed during the (tentative) repair. For the set of assignments $A = \{\alpha_1, \ldots, \alpha_n\} \subseteq \mathbb{A}$, let the associated set of auxiliary propositional variables be $\mathsf{D}(A) = \{\mathsf{D}(\alpha) \ : \ \alpha \in A\}$. Then the program

$$-\mathsf{D}(A) = \{-\mathsf{D}(\alpha_1), \ldots, -\mathsf{D}(\alpha_n), -\mathsf{D}(\overline{\alpha}_1), \ldots, -\mathsf{D}(\overline{\alpha}_n)\}$$

initialises the auxiliary variables $\mathsf{D}(\alpha_i)$ and $\mathsf{D}(\overline{\alpha}_i)$ to false. Note that the propositional variable $\mathsf{D}(\alpha)$ is to be distinguished from the history atom $\alpha \in \mathbb{A}$ expressing that α was one of the last assignments that brought the database into the violation state.

[8] An alternative to the update of $\mathcal{H}$ would be to erase the history and replace it by A. Altogether, this would make us go from $\langle \mathcal{D}, \mathcal{H}\rangle$ to $\langle \mathcal{D} \circ A, A\rangle$. This would be appropriate in order to model external updates such as the update $\{+\mathtt{mg}_{e,p}\}$ that had occurred in Example 2 and brought the database into an inconsistent state, differing from the kind of updates occurring during the repair process that our definition accounts for.

The next step is to associate repair programs $\mathsf{rep}(\alpha)$ to assignments α. These programs check whether α is triggered by some ECA constraint κ and if so performs it. This involves some bookkeeping by means of the auxiliary variables $\mathsf{D}(\alpha)$, for two reasons: first, to make sure that none of the assignments $\alpha \in A$ is executed twice; second, to make sure that A is consistent w.r.t. α, in the sense that it does not contain both α and its opposite $\overline{\alpha}$.

$$\mathsf{rep}(\alpha) = \neg\mathsf{D}(\alpha) \wedge \neg\mathsf{D}(\overline{\alpha}) \wedge \bigvee_{\kappa \in \mathcal{K} \,:\, \alpha \in \mathbf{A}(\kappa)} (\mathbf{E}(\kappa) \wedge \mathbf{C}(\kappa))?; \{\alpha, +\mathsf{D}(\alpha)\}.$$

The program first performs a test: neither α nor its opposite $\overline{\alpha}$ has been done up to now (this is the conjunct $\neg\mathsf{D}(\alpha) \wedge \neg\mathsf{D}(\overline{\alpha})$) and there is an ECA constraint $\kappa \in \mathcal{K}$ with α in the action part such that the event description $\mathbf{E}(\kappa)$ and the condition $\mathbf{C}(\kappa)$ are both true (this is the conjunct $\bigvee_{\kappa \,:\, \alpha \in \mathbf{A}(\kappa)}(\mathbf{E}(\kappa) \wedge \mathbf{C}(\kappa))$). If that big test program succeeds then α is executed and this is stored by making $\mathsf{D}(\alpha)$ true (this is the update $\{\alpha, +\mathsf{D}(\alpha)\}$).

Theorem 1. *Let Δ be an h-database and $\mathcal{K}$ a set of ECA constraints. There exists a well-founded weak ECA repair of Δ via $\mathcal{K}$ if and only if the program*

$$\mathsf{rep}^{\mathsf{wwf}}_{\mathcal{K}} = \text{–}\mathsf{D}(\mathbb{A}); \Big(\bigcup_{\alpha \in \mathbb{A}} \mathsf{rep}(\alpha) \Big)^*; \mathsf{OK}(\mathcal{K})?$$

is executable at Δ.

The unbounded iteration $(\bigcup_{\alpha \in \mathbb{A}} \mathsf{rep}(\alpha))^*$ in our repair program $\mathsf{rep}^{\mathsf{wwf}}_{\mathcal{K}}$ can be replaced by $(\bigcup_{\alpha \in \mathbb{A}} \mathsf{rep}(\alpha))^{\leq \mathsf{card}(\mathbb{P})}$, i.e., the number of iterations of $\bigcup_{\alpha \in \mathbb{A}} \mathsf{rep}(\alpha)$ can be bound by the cardinality of $\mathbb{P}$. This is the case because well-founded ECA repairs are consistent update actions: each propositional variable can occur at most once in a well-founded ECA repair. This also holds for the other repair programs that we are going to define below.

Observe that finiteness of $\mathbb{P}$ is necessary for Theorem 1. This is not the case for the next result.

Theorem 2. *Let Δ be an h-database and $\mathcal{K}$ a set of ECA constraints. The set of assignments $A = \{\alpha_1, \ldots, \alpha_n\}$ is a well-founded weak ECA repair of Δ via $\mathcal{K}$ if and only if the program*

$$\mathsf{rep}^{\mathsf{wwf}}_{\mathcal{K}}(A) = \text{–}\mathsf{D}(A); \Big(\bigcup_{\alpha \in A} \mathsf{rep}(\alpha) \Big)^*; \mathsf{OK}(\mathcal{K})?; \bigwedge_{\alpha \in A} \mathsf{D}(\alpha)?$$

is executable at Δ.

The length of both repair programs is polynomial in the size of the set of ECA constraints and the cardinality of $\mathbb{P}$.

6.2 Well-Founded ECA Repairs

In order to capture well-founded ECA repairs we have to integrate a minimality check into its DL-PA$^{\pm}$ account. We take inspiration from the translation of Winslett's Possible Models Approach for database updates [25,26] into DL-PA of [21]. The translation associates to each propositional variable p a fresh copy p' whose role is to store the truth value of p. This is done before repair programs $\mathsf{rep}(+p)$ or $\mathsf{rep}(-p)$ are executed so that the initial value of p is remembered: once a candidate repair has been computed we check that it is minimal, in the sense that no consistent state can be obtained by modifying less variables.

For α being either $+p$ or $-p$, the programs $\mathtt{copy}(\alpha)$ and $\mathtt{undo}(\alpha)$ respectively copy the truth value of p into p' and, the other way round, copy back the value of p' into p. They are defined as follows

$$\mathtt{copy}(\alpha) = (p?;+p') \cup (\neg p?;-p'),$$
$$\mathtt{undo}(\alpha) = (p'?;+p) \cup (\neg p'?;-p).$$

For example, the formulas $p \to \langle \mathtt{copy}(+p)\rangle(p \wedge p')$ and $\neg p \to \langle \mathtt{copy}(+p)\rangle(\neg p \wedge \neg p')$ are both valid, as well as $p' \to \langle \mathtt{undo}(+p)\rangle(p \wedge p')$ and $\neg p' \to \langle \mathtt{undo}(+p)\rangle(\neg p \wedge \neg p')$. Then the program

$$\mathsf{init}(\{\alpha_1,\ldots,\alpha_n\}) = \{-\mathsf{D}(\alpha_1),\ldots,-\mathsf{D}(\alpha_n),$$
$$-\mathsf{D}(\overline{\alpha}_1),\ldots,-\mathsf{D}(\overline{\alpha}_n)\};\mathtt{copy}(\alpha_1);\cdots;\mathtt{copy}(\alpha_n)$$

initialises the values of both kinds of auxiliary variables: it resets all 'done' variables $\mathsf{D}(\alpha_i)$ and $\mathsf{D}(\overline{\alpha}_i)$ to false as before and moreover makes copies of all variables that are going to be assigned by α_i (where the order of the α_i does not matter).

Theorem 3. *Let Δ be an h-database and $\mathcal{K}$ a set of ECA constraints. There exists a well-founded ECA repair of Δ via $\mathcal{K}$ if and only if the program*

$$\mathsf{rep}^{\mathsf{wf}}_{\mathcal{K}} = \mathsf{init}(\mathbb{A});\Big(\bigcup_{\alpha\in\mathbb{A}}\mathsf{rep}(\alpha)\Big)^*;\mathsf{OK}(\mathcal{K})?;$$
$$\neg\Big\langle\Big(\bigcup_{\alpha\in\mathbb{A}}\mathsf{D}(\alpha)?;\mathtt{undo}(\alpha)\Big)^+\Big\rangle\mathsf{OK}(\mathcal{K})?$$

is executable at Δ.

Theorem 4. *Let Δ be an h-database and $\mathcal{K}$ a set of ECA constraints. The set of assignments $A = \{\alpha_1,\ldots,\alpha_n\}$ is a well-founded ECA repair of Δ via $\mathcal{K}$ if and only if the program*

$$\mathsf{rep}^{\mathsf{wf}}_{\mathcal{K}}(A) = \mathsf{init}(A);\Big(\bigcup_{\alpha\in A}\mathsf{rep}(\alpha)\Big)^*;\mathsf{OK}(\mathcal{K})?;\Big(\bigwedge_{\alpha\in A}\mathsf{D}(\alpha)\Big)?;$$
$$\neg\Big\langle\Big(\bigcup_{\alpha\in A}\mathtt{undo}(\alpha)\Big)^+\Big\rangle\mathsf{OK}(\mathcal{K})?$$

is executable at Δ.

The length of both repair programs is polynomial in the size of the set of ECA constraints and the cardinality of $\mathbb{P}$.

7 Other Decision Problems

In the present section we discuss how several other interesting decision problems related to well-founded weak ECA repairs and well-founded ECA repairs can be expressed. Let $\mathsf{rep}_{\mathcal{K}}$ stand for either $\mathsf{rep}^{\mathsf{wwf}}_{\mathcal{K}}$ or $\mathsf{rep}^{\mathsf{wf}}_{\mathcal{K}}$, i.e., the program performing well-founded (weak) ECA repairs according to the set of ECA constraints $\mathcal{K}$.

7.1 Properties of a Set of ECA Constraints

Here are three other decision problems about a given set of ECA constraints:

1. Is there a unique repair of Δ via $\mathcal{K}$?
2. Does every Δ have a unique repair via $\mathcal{K}$?
3. Does every Δ have a repair via $\mathcal{K}$?

Each of them can be expressed in DL-PA$^{\pm}$. First, we can verify whether there is a unique repair of Δ via $\mathcal{K}$ by checking whether the program $\mathsf{rep}_{\mathcal{K}}$ is deterministic. This can be done by model checking, for each of the variables $p \in \mathbb{P}$ occurring in $\mathcal{K}$, whether $\Delta \models \langle \mathsf{rep}_{\mathcal{K}} \rangle p \rightarrow [\mathsf{rep}_{\mathcal{K}}]p$. Second, global unicity of the repairs (independently of a specific database Δ) can be verified by checking for each of the variables $p \in \mathbb{P}$ whether $\langle \mathsf{rep}_{\mathcal{K}} \rangle p \rightarrow [\mathsf{rep}_{\mathcal{K}}]p$ is DL-PA$^{\pm}$ valid. Third, we can verify whether $\mathcal{K}$ can repair every database by checking whether the formula $\langle \mathsf{rep}_{\mathcal{K}} \rangle \top$ is DL-PA$^{\pm}$ valid.

7.2 Comparing Two Sets of ECA Constraints

We start by two definitions that will be useful for our purposes. The program π_1 is *included* in the program π_2 if $||\pi_1|| \subseteq ||\pi_2||$. Two programs π_1 and π_2 are *equivalent* if each is included in the other, that is, if $||\pi_1|| = ||\pi_2||$.

These comparisons can be polynomially reduced into validity checking problems. Our translation makes use of the assignment-recording propositional variables of Section 6.1 and of the copies of propositional variables that we have introduced in Section 6.2. Hence we suppose that for every variable p there is a fresh variable p' that will store the truth value of p, as well as two fresh variables $\mathsf{D}(+p')$ and $\mathsf{D}(-p')$. For a given set of propositional variables $P \subseteq \mathbb{P}$ we make use of the following three sets of auxiliary variables

$$P' = \{p' \ : \ p \in P\},$$
$$\mathsf{D}(+(P')) = \{\mathsf{D}(+p') \ : \ p \in P\},$$
$$\mathsf{D}(-(P')) = \{\mathsf{D}(-p') \ : \ p \in P\}.$$

The auxiliary variables are used in a 'generate and test' schema. For a given set of propositional variables $P \subseteq \mathbb{P}$, the program

$$\mathsf{guess}(P) = -(P' \cup \mathsf{D}(+(P')) \cup \mathsf{D}(-(P'))); (\bigcup_{q \in P'} +q)^*$$

guesses nondeterministically which of the auxiliary variables for P are going to be modified: first all are set to false and then some subset is made true. The formula

$$\mathsf{Guessed}(P) = \bigwedge_{p \in P} \big((p \leftrightarrow p') \wedge (+p \leftrightarrow \mathsf{D}(+p)) \wedge (-p \leftrightarrow \mathsf{D}(-p))\big)$$

checks that the guess was correct.

Now we are ready to express inclusion of programs in DL-PA$^{\pm}$: we predict the outcome of π_1 and then check if π_2 produces the same set of changes as π_1.

Proposition 3. *Let π_1 and π_2 be two DL-PA programs.*

1. *π_1 is included in π_2 if and only if*

$$[\mathsf{guess}(\mathbb{P}_{\pi_1} \cup \mathbb{P}_{\pi_2})]\big(\langle \pi_1 \rangle \mathsf{Guessed}(\mathbb{P}_{\pi_1} \cup \mathbb{P}_{\pi_2}) \rightarrow \langle \pi_2 \rangle \mathsf{Guessed}(\mathbb{P}_{\pi_1} \cup \mathbb{P}_{\pi_2})\big)$$

   is DL-PA$^{\pm}$ valid.
2. *π_1 and π_2 are equivalent if and only if*

$$[\mathsf{guess}(\mathbb{P}_{\pi_1} \cup \mathbb{P}_{\pi_2})]\big(\langle \pi_1 \rangle \mathsf{Guessed}(\mathbb{P}_{\pi_1} \cup \mathbb{P}_{\pi_2}) \leftrightarrow \langle \pi_2 \rangle \mathsf{Guessed}(\mathbb{P}_{\pi_1} \cup \mathbb{P}_{\pi_2})\big)$$

   is DL-PA$^{\pm}$ valid.

The reduction is polynomial. The following decision problems can be reformulated in terms of program inclusion and equivalence:

1. Is the ECA constraint $\kappa \in \mathcal{K}$ redundant?
2. Are two sets of ECA constraints $\mathcal{K}_1$ and $\mathcal{K}_2$ equivalent?
3. Can all databases that are repaired by $\mathcal{K}_1$ also be repaired by $\mathcal{K}_2$?

Each can be expressed in DL-PA$^{\pm}$ by means of program inclusion or equivalence: the first can be decided by checking whether the programs $\mathsf{rep}_{\mathcal{K}}$ and $\mathsf{rep}_{\mathcal{K}\setminus\{\kappa\}}$ are equivalent; the second can be decided by checking whether the programs $\mathsf{rep}_{\mathcal{K}_1}$ and $\mathsf{rep}_{\mathcal{K}_2}$ are equivalent; the third problem can be decided by checking the validity of $\langle \mathsf{rep}_{\mathcal{K}_1} \rangle \top \rightarrow \langle \mathsf{rep}_{\mathcal{K}_2} \rangle \top$.

7.3 Termination

In our DL-PA$^{\pm}$ framework, the taming of termination problems is not only due to the definition of well-founded ECA repairs itself: in dynamic logics, the Kleene star is about unbounded but finite iterations. The modal diamond operator therefore quantifies over terminating executions only, disregarding any infinite executions. We can nevertheless reason about infinite computations by dropping the tests $\neg\mathsf{D}(\alpha)\wedge\neg\mathsf{D}(\overline{\alpha})$ from the definition of $\mathsf{rep}(\alpha)$. Let the resulting 'unbounded' repair program be $\mathsf{urep}(\alpha)$. Let $\mathsf{urep}_{\mathcal{K}}$ stand for either the programs $\mathsf{urep}^{\mathsf{wwf}}_{\mathcal{K}}$ or $\mathsf{urep}^{\mathsf{wf}}_{\mathcal{K}}$ resulting from the replacement of $\mathsf{rep}(\alpha)$ by $\mathsf{urep}(\alpha)$. Then the repair of Δ loops if and only if $\Delta \models [(\mathsf{urep}_{\mathcal{K}})^*]\langle\mathsf{urep}_{\mathcal{K}}\rangle\top$.

8 Conclusion

Our dynamic logic semantics for ECA constraints generalises the well-founded AIC repairs of [7], and several decision problems can be captured in DL-PA$^{\pm}$. Proposition 2 provides a PSPACE upper bound for all these problems. A closer look at the characterisations of Section 6 shows that the complexity is actually lower. For the results of Section 6.1 (Theorem 1 and Theorem 2), as executability of a program π at Δ is the same as truth of $\langle\pi\rangle\top$ at Δ, our characterisation involves a single existential quantification (a modal diamond operator), with a number of nondeterministic choices that is quadratic in $\mathsf{card}(\mathbb{P})$ (precisely, $1+2\mathsf{card}(\mathbb{P})$ nondeterministic choices that are iterated $\mathsf{card}(\mathbb{P})$ times, cf. what we have remarked after the theorem, as well as the definition of $\pi^{\leq n}$ in Section 5). Just as the corresponding QBF fragment, this fragment is in NP. For the results of Section 6.2 (Theorem 3 and Theorem 4), as executability of $\pi;\neg\langle\pi'\rangle\varphi?$ at Δ is the same as truth of $\langle\pi\rangle[\pi']\varphi$ at Δ, our characterisation involves an existential diamond containing the same program as above that is preceded by $\mathsf{init}(\mathbb{A})$ and that is followed by a universal quantification (a modal box operator). Just as the corresponding QBF fragment, this fragment is in Σ^p_2.

Beyond these decision problems we can express repair algorithms as DL-PA$^{\pm}$ programs, given that the standard programming constructions such as if-then-else and while can all be expressed in dynamic logic. Correctness of such a program π can be verified by checking whether π is included in the program $\mathsf{rep}_{\mathcal{K}}$. The other way round, one can check whether π is able to output *any* well-founded ECA repair by checking whether $\mathsf{rep}_{\mathcal{K}}$ is included in π.

It remains to study further our founded ECA repairs of Section 4.2 and their grounded versions. We also plan to check whether the more expressive existential AICs of [9] transfer. Finally, we would like to generalise the history component of h-databases from update actions to event algebra expressions as studied e.g. in [22,1]; dynamic logic should be beneficial here, too.

References

1. Alferes, J.J., Banti, F., Brogi, A.: An event-condition-action logic programming language. In: Fisher, M., van der Hoek, W., Konev, B., Lisitsa, A. (eds.) Logics

in Artificial Intelligence, 10th European Conference, JELIA 2006, Liverpool, UK, September 13-15, 2006, Proceedings. LNCS, vol. 4160, pp. 29–42. Springer (2006). https://doi.org/10.1007/11853886_5, https://doi.org/10.1007/11853886_5
2. Balbiani, P., Herzig, A., Schwarzentruber, F., Troquard, N.: DL-PA and DCL-PC: model checking and satisfiability problem are indeed in PSPACE. CoRR **abs/1411.7825** (2014), http://arxiv.org/abs/1411.7825
3. 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. pp. 143–152. IEEE Computer Society (2013), https://doi.org/10.1109/LICS.2013.20
4. Bertossi, L.E.: Database Repairing and Consistent Query Answering. Synthesis Lectures on Data Management, Morgan & Claypool Publishers (2011), https://doi.org/10.2200/S00379ED1V01Y201108DTM020
5. Bertossi, L.E., Pinto, J.: Specifying active rules for database maintenance. In: Saake, G., Schwarz, K., Türker, C. (eds.) Transactions and Database Dynamics, Proceedings of the Eight International Workshop on Foundations of Models and Languages for Data and Objects, Schloß Dagstuhl, Germany, September 27-30, 1999. vol. Preprint Nr. 19, pp. 65–81. Fakultät für Informatik, Otto-von-Guericke-Universität Magdeburg (1999)
6. Bidoit, N., Maabout, S.: A model theoretic approach to update rule programs. In: Afrati, F.N., Kolaitis, P.G. (eds.) Database Theory - ICDT '97, 6th International Conference, Delphi, Greece, January 8-10, 1997, Proceedings. LNCS, vol. 1186, pp. 173–187. Springer (1997). https://doi.org/10.1007/3-540-62222-5_44, https://doi.org/10.1007/3-540-62222-5_44
7. Bogaerts, B., Cruz-Filipe, L.: Fixpoint semantics for active integrity constraints. Artif. Intell. **255**, 43–70 (2018). https://doi.org/10.1016/j.artint.2017.11.003, https://doi.org/10.1016/j.artint.2017.11.003
8. Calautti, M., Caroprese, L., Greco, S., Molinaro, C., Trubitsyna, I., Zumpano, E.: Consistent query answering with prioritized active integrity constraints. In: Desai, B.C., Cho, W. (eds.) IDEAS 2020: 24th International Database Engineering & Applications Symposium, Seoul, Republic of Korea, August 12-14, 2020. pp. 3:1–3:10. ACM (2020), https://dl.acm.org/doi/10.1145/3410566.3410592
9. Calautti, M., Caroprese, L., Greco, S., Molinaro, C., Trubitsyna, I., Zumpano, E.: Existential active integrity constraints. Expert Syst. Appl. **168**, 114297 (2021). https://doi.org/10.1016/j.eswa.2020.114297, https://doi.org/10.1016/j.eswa.2020.114297
10. Caroprese, L., Greco, S., Sirangelo, C., Zumpano, E.: Declarative semantics of production rules for integrity maintenance. In: Etalle, S., Truszczynski, M. (eds.) Logic Programming, 22nd International Conference, ICLP 2006, Seattle, WA, USA, August 17-20, 2006, Proceedings. LNCS, vol. 4079, pp. 26–40. Springer (2006), https://doi.org/10.1007/11799573_5
11. Caroprese, L., Greco, S., Zumpano, E.: Active integrity constraints for database consistency maintenance. IEEE Trans. Knowl. Data Eng. **21**(7), 1042–1058 (2009), https://doi.org/10.1109/TKDE.2008.226
12. Caroprese, L., Truszczynski, M.: Active integrity constraints and revision programming. TPLP **11**(6), 905–952 (2011). https://doi.org/10.1017/S1471068410000475, https://doi.org/10.1017/S1471068410000475
13. Ceri, S., Fraternali, P., Paraboschi, S., Tanca, L.: Automatic generation of production rules for integrity maintenance. ACM Trans. Database Syst. **19**(3), 367–422 (1994), http://doi.acm.org/10.1145/185827.185828

14. Chomicki, J., Lobo, J., Naqvi, S.A.: Conflict resolution using logic programming. IEEE Trans. Knowl. Data Eng. **15**(1), 244–249 (2003). https://doi.org/10.1109/TKDE.2003.1161596, https://doi.org/10.1109/TKDE.2003.1161596
15. Chomicki, J., Marcinkowski, J.: Minimal-change integrity maintenance using tuple deletions. Inf. Comput. **197**(1-2), 90–121 (2005), https://doi.org/10.1016/j.ic.2004.04.007
16. 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. LNCS, vol. 8367, pp. 361–380. Springer (2014), https://doi.org/10.1007/978-3-319-04939-7_18
17. Cruz-Filipe, L., Gaspar, G., Engrácia, P., Nunes, I.: Computing repairs from active integrity constraints. In: Seventh International Symposium on Theoretical Aspects of Software Engineering, TASE 2013, 1-3 July 2013, Birmingham, UK. pp. 183–190. IEEE Computer Society (2013), https://doi.org/10.1109/TASE.2013.32
18. Cruz-Filipe, L., Gaspar, G., Nunes, I., Schneider-Kamp, P.: Active integrity constraints for general-purpose knowledge bases. Ann. Math. Artif. Intell. **83**(3-4), 213–246 (2018). https://doi.org/10.1007/s10472-018-9577-y, https://doi.org/10.1007/s10472-018-9577-y
19. Flesca, S., Greco, S.: Declarative semantics for active rules. Theory Pract. Log. Program. **1**(1), 43–69 (2001), http://journals.cambridge.org/action/displayAbstract?aid=71136
20. Flesca, S., Greco, S., Zumpano, E.: Active integrity constraints. In: Moggi, E., Warren, D.S. (eds.) Proceedings of the 6th International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming, 24-26 August 2004, Verona, Italy. pp. 98–107. ACM (2004), http://doi.acm.org/10.1145/1013963.1013977
21. Herzig, A.: Belief change operations: A short history of nearly everything, told in dynamic logic of propositional assignments. In: Baral, C., Giacomo, G.D., 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 (2014), http://www.aaai.org/ocs/index.php/KR/KR14/paper/view/7960
22. Lausen, G., Ludäscher, B., May, W.: On logical foundations of active databases. In: Chomicki, J., Saake, G. (eds.) Logics for Databases and Information Systems (the book grow out of the Dagstuhl Seminar 9529: Role of Logics in Information Systems, 1995). pp. 389–422. Kluwer (1998)
23. Ludäscher, B., May, W., Lausen, G.: Nested transactions in a logical language for active rules. In: Pedreschi, D., Zaniolo, C. (eds.) Logic in Databases, International Workshop LID'96, San Miniato, Italy, July 1-2, 1996, Proceedings. LNCS, vol. 1154, pp. 197–222. Springer (1996). https://doi.org/10.1007/BFb0031742, https://doi.org/10.1007/BFb0031742
24. Widom, J., Ceri, S.: Active database systems: Triggers and rules for advanced database processing. Morgan Kaufmann (1996)
25. Winslett, M.: Reasoning about action using a possible models approach. In: Shrobe, H.E., Mitchell, T.M., Smith, R.G. (eds.) Proceedings of the 7th National Conference on Artificial Intelligence, St. Paul, MN, USA, August 21-26, 1988. pp. 89–93. AAAI Press / The MIT Press (1988), http://www.aaai.org/Library/AAAI/1988/aaai88-016.php
26. Winslett, M.A.: Updating Logical Databases. Cambridge Tracts in Theoretical Computer Science, Cambridge University Press (1990)