

Active Integrity Constraints with Existential Quantification

Marco Calautti¹, Luciano Caroprese², Sergio Greco³, Cristian Molinaro³,
Irina Trubitsyna³ and Ester Zumpano³

¹DISI, University of Trento, Italy

²ICAR-CNR, Italy

³DIMES, University of Calabria, Italy

Abstract

We present the framework of *Existential Active Integrity Constraints* (EAICs) introduced in [1]. EAICs are a powerful extension of Active Integrity Constraints (AICs), which allow us to express a wide range of constraints used in databases and ontological systems. Specifically, the paper discusses a new definition of founded updates for AICs, presents syntax and semantics for EAICs, and a “representative” set of founded updates for EAICs, called *universal*, which suffices for query answering.

1. Introduction

The *consistent query answering* (CQA) framework is a principled approach to answer queries over inconsistent databases. It was first proposed in [2], giving rise to flourishing research activity (e.g., see [3, 4, 5, 6]). It relies on the notions of *repair* and *consistent query answer*. Intuitively, a repair for a possibly inconsistent database is a consistent database that “minimally” differs from the original one. In general, there may be multiple repairs for an inconsistent database. The *consistent* (or *certain*) *answers* to a query over an inconsistent database are the query answers that can be obtained from every repair.

Example 1. Consider the database schema consisting of two relations $\text{emp}(\text{Name}, \text{Dept})$ and $\text{dept}(\text{Name})$, where the former stores information on employees and departments they work for, and the latter stores all departments. Consider a referential integrity constraint stating that every department occurring in the emp relation must appear in the dept relation too. This constraint can be expressed through the following logical formula: $\forall E \forall D [\text{emp}(E, D) \wedge \neg \text{dept}(D) \Rightarrow \perp]$.


Consider now the database D consisting of the facts $\text{emp}(\text{john}, \text{cs})$, $\text{emp}(\text{john}, \text{math})$, and $\text{dept}(\text{math})$. Clearly, D is inconsistent, as the cs department appearing in the emp relation does not appear in the dept relation. A repair can be obtained by applying a minimal (under set-inclusion) set of update operations to the original database. We consider only fact insertions

SEBD 2022: The 30th Italian Symposium on Advanced Database Systems, June 19-22, 2022, Tirrenia (PI), Italy

✉ marco.calautti@unitn.it (M. Calautti); luciano.caroprese@icar.cnr.it (L. Caroprese); greco@dimes.unical.it (S. Greco); cmolinaro@dimes.unical.it (C. Molinaro); trubitsyna@dimes.unical.it (I. Trubitsyna); zumpano@dimes.unical.it (E. Zumpano)



© 2022 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

 CEUR Workshop Proceedings (CEUR-WS.org)

and deletions as admissible update operations.¹ Therefore, there are two repairs: D_1 , obtained by inserting the fact $\text{dept}(\text{cs})$ into D , and D_2 , obtained by deleting the fact $\text{emp}(\text{john}, \text{cs})$ from D . The only consistent answer to the query asking for all departments' names is math .

Although inconsistent databases can be repaired in different ways, in many applications it is natural and desirable to express that only a restricted set of update operations can be performed to restore consistency, which cannot be done with classical integrity constraints. *Active integrity constraints* (AICs) [13] have been introduced to overcome such a limitation.

Example 2. Consider again the scenario of Example 1 and suppose that, when the integrity constraint is violated, we want to restore consistency only by adding missing departments (and thus avoid deleting facts of the emp relation). This behavior can be expressed by means of the following active integrity constraint: $\forall E \forall D [\text{emp}(E, D) \wedge \neg \text{dept}(D) \Rightarrow +\text{dept}(D)]$.

The same constraint of Example 1 is defined on the left-hand side of \Rightarrow , while on the right-hand side the only admissible update operation is specified. Thus, only the insertion of $\text{dept}(\text{cs})$ can be performed to restore consistency of D , and D_1 is the only acceptable repair. As defined in the following, inserting $\text{dept}(\text{cs})$ is a “founded” update, because the AIC above allows it, while deleting $\text{emp}(\text{john}, \text{cs})$ is not a founded update, because the AIC above does not allow it.

Active integrity constraints allow users to express integrity constraints along with admissible update operations. One limitation of AICs is that they do not allow existential quantification, and thus do not allow users to formulate classical constraints such as foreign keys and more general inclusion dependencies, which require existentially quantified variables to be expressed [14].

We can lift the idea of AICs (that is, to specify which update operations should be applied when a constraint is violated) to *Existential Active Integrity Constraints* (EAICs), which generalize AICs enabling users to express a wider class of integrity constraints commonly arising in practice.

Example 3. The EAIC: $\forall E \forall D [\text{emp}(E, D) \wedge \nexists C \text{dept}(D, C) \Rightarrow \exists Z +\text{dept}(D, Z)]$ defines a constraint stating that inconsistency must be resolved by adding missing departments to relation dept . Importantly, EAICs lead to value invention because of existential variables, which is not the case for AICs, and this poses different new issues—for instance, for a database containing only the fact $\text{emp}(\text{john}, \text{cs})$, a city for the cs department needs to be invented.

In this paper we present syntax and semantics of EAICs, and show how to define a “representative” set of founded updates for EAICs, called *universal*, which suffices for query answering.

2. Preliminaries

We assume the existence of the following (pairwise disjoint) sets: *predicates* \mathcal{P} , *variables* \mathcal{V} , and *constants* \mathcal{C} . Each predicate is associated with an *arity*, which is a non-negative integer. A *term* is either a constant or a variable.

¹Other minimality criteria and update operations, such as value updates [7, 8, 9, 10, 11], have been considered in the literature. In this paper, we consider minimality under set-inclusion and insert/delete updates, which indeed are the most common minimality criteria and repair primitives considered in the literature. When only deletions are allowed, the set of operations is a minimal transversal of all minimal inconsistent subsets of the database [12].

An *atom* is of the form $p(t_1, \dots, t_n)$, where p is a predicate of arity n and the t_i 's are terms. We write an atom containing only constants also as $p(\bar{c})$, where \bar{c} is understood to be a sequence of constants, and write $p(\bar{X})$ to refer to an atom whose terms are the variables \bar{X} . A *literal* is either an atom A (*positive literal*) or its negation $\neg A$ (*negative literal*). An *update atom* is of the form $+a(\bar{X})$ or $-a(\bar{X})$, where $a(\bar{X})$ is an atom. Intuitively, a ground update atom $+a(\bar{c})$ (resp. $-a(\bar{c})$) states that $a(\bar{c})$ will be inserted into (resp. deleted from) the database. The *complementary literal* of an update atom $+a(\bar{X})$ (resp. $-a(\bar{X})$) is $CompLit(+a(\bar{X})) = \neg a(\bar{X})$ (resp. $CompLit(-a(\bar{X})) = a(\bar{X})$). For any set \mathcal{U} of update atoms, $CompLit(\mathcal{U}) = \{CompLit(\pm a(\bar{X})) \mid \pm a(\bar{X}) \in \mathcal{U}\}$. Logical formulae are built using literals and logical connectives—the precise syntax will be defined later.

A term/atom/literal/formula is *ground* if it is variable-free. A formula φ' is a *ground instance* of a formula φ if φ' can be obtained from φ by substituting every variable in φ with a constant. We use $ground(\varphi)$ to denote the set of all ground instances of φ , and for a set of formulae Φ , we define $ground(\Phi) = \cup_{\varphi \in \Phi} ground(\varphi)$.

Active Integrity Constraints. An *active integrity constraint* (AIC) σ is of the form:

$$\forall \bar{X} \left[\bigwedge_{i=1}^m b_i(\bar{X}_i) \wedge \bigwedge_{i=m+1}^n \neg b_i(\bar{X}_i) \Rightarrow \bigvee_{i=1}^q -a_i(\bar{X}_i) \vee \bigvee_{i=q+1}^p +a_i(\bar{X}_i) \right] \quad (1)$$

where (i) $n, p > 0$, (ii) the $b_i(\bar{X}_i)$'s are atoms, (iii) the $-a_i(\bar{X}_i)$'s and $+a_i(\bar{X}_i)$'s are update atoms, (iv) variables occurring in negative literals also occur in positive literals, and (v) $CompLit(head(\sigma)) \subseteq body(\sigma)$, where $head(\sigma)$ (resp., $body(\sigma)$) denotes the right (resp., left) hand side of \Rightarrow .

For an AIC σ , $body^+(\sigma)$ and $body^-(\sigma)$ denote the set of positive and negative atoms in $body(\sigma)$, respectively. An AIC specifies both an integrity constraint (in the body) and the actions to be performed (in the head) if the integrity constraint is violated. We use $St(\sigma)$ to denote the integrity constraint derived from σ by removing all the head update atoms. For a set of active integrity constraints Σ , $St(\Sigma)$ denotes the corresponding set of integrity constraints, that is $St(\Sigma) = \{St(\sigma) \mid \sigma \in \Sigma\}$. Furthermore, for any set of AICs Σ and set of ground update atoms \mathcal{U} , $\Sigma[\mathcal{U}]$ denotes the set of AICs derived from $ground(\Sigma)$ by deleting head update atoms not occurring in \mathcal{U} and AICs such that all head update atoms have been deleted.

We now present the semantics of AICs used in [1]. Given a database D and a set of AICs Σ :

- A set \mathcal{R} of ground update atoms is an *update* for $\langle D, \Sigma \rangle$ if it is an update for $\langle D, St(\Sigma) \rangle$.
- An update \mathcal{R} for $\langle D, \Sigma \rangle$ is *founded* iff it is an update for $\langle D, \Sigma[\mathcal{R}] \rangle$.
- A repair $\mathcal{R}(D)$ is *founded* iff \mathcal{R} is a founded update.

The idea underlying the definition above is that the actions of an update must be determined only by the AICs allowing those actions. Observe that the founded semantics guarantees that, given a founded repair \mathcal{R} , for each update atom $\pm A \in \mathcal{R}$ there must be an AIC $\sigma \in ground(\Sigma)$ such that $\pm A \in head(\sigma)$ (otherwise \mathcal{R} is not minimal).

Although the definition introduced in [1] is different from that used in [13], we use the same name since the former is a refinement of the latter, and its purpose is to overcome the problem

of cyclic support, see [15]. Theorem 8 in [1] shows that every founded update according to the current definition is also founded according to the definition given in [13].

The sets of all updates and founded updates for a database D and a set of AICs Σ are denoted as $\mathbf{R}(D, \Sigma)$ and $\mathbf{FR}(D, \Sigma)$ respectively. Clearly, $\mathbf{FR}(D, \Sigma) \subseteq \mathbf{R}(D, \Sigma)$.

More details regarding AIC and their extensions can be found in [1, 13, 16]. The *certain* answers to a query Q on a database D w.r.t. a set of AICs Σ are $\text{CERTAIN}(Q, D, \Sigma) = \bigcap_{\mathcal{R} \in \mathbf{FR}(D, \Sigma)} Q(\mathcal{R}(D))$.

3. Existential Active Integrity Constraints

Syntax. We are now going to present an extension of AICs, that let us define AICs with existentially quantified variables, allowing for more expressive integrity constraints, like inclusion dependencies. The set of *complementary literals* of an update atom is redefined as follows:

- $\text{CompLit}(-a(\bar{X})) = \{a(\bar{X})\}$;
- $\text{CompLit}(\exists \bar{Z} + a(\bar{X}, \bar{Z})) = \{\# \bar{Y} a(\bar{X}', \bar{Y}) \mid \exists \text{subst. } \vartheta \text{ s.t. } a(\bar{X}', \vartheta(\bar{Y})) = a(\bar{X}, \bar{Z})\}$.

For any set \mathcal{U} of update atoms, $\text{CompLit}(\mathcal{U}) = \cup_{\pm A \in \mathcal{U}} \text{CompLit}(\pm A)$.

Definition 1. An Existential Active Integrity Constraint (EAIC) is of the form:

$$\forall \bar{X} \left[\bigwedge_{i=1}^m b_i(\bar{X}_i) \wedge \bigwedge_{i=m+1}^n \# \bar{Z}_i b_i(\bar{X}_i, \bar{Z}_i) \Rightarrow \bigvee_{i=1}^q -a_i(\bar{X}_i) \vee \bigvee_{i=q+1}^p \exists \bar{Z}_i + a_i(\bar{X}_i, \bar{Z}_i) \right] \quad (2)$$

where (i) $n, p > 0$, (ii) universal variables occurring in negative body literals also occur in positive body literals, (iii) every existential variable occurs only in one update atom or negative body literal, and (iv) for each $\pm A \in \text{head}(\sigma)$, the condition $\text{CompLit}(\pm A) \cap \text{body}(\sigma) \neq \emptyset$ holds. \square

Example 4. Consider two relations $\text{node}(\text{Id})$ and $\text{edge}(\text{Source}, \text{Dest}, \text{Weight})$ used to store nodes and weighted edges of a graph, respectively.

For the EAIC $\sigma : \text{node}(\bar{X}_1) \wedge \text{node}(\bar{X}_2) \wedge \# \bar{Y}_1, \bar{Y}_2 \text{ edge}(\bar{X}_1, \bar{Y}_1, \bar{Y}_2) \Rightarrow \exists \bar{Z}_1 + \text{edge}(\bar{X}_1, \bar{X}_2, \bar{Z}_1)$, we have $\text{CompLit}(\exists \bar{Z}_1 + \text{edge}(\bar{X}_1, \bar{X}_2, \bar{Z}_1)) \cap \text{body}(\sigma) = \{\# \bar{Y}_1, \bar{Y}_2 \text{ edge}(\bar{X}_1, \bar{Y}_1, \bar{Y}_2)\} \neq \emptyset$.

A negative body literal $\# \bar{Z}_i b_i(\bar{X}_i, \bar{Z}_i)$ s.t. \bar{Z}_i is empty will be simply written as $\neg b_i(\bar{X}_i)$. For an EAIC σ , $\text{St}(\sigma)$ denotes the constraint, called *existential integrity constraint* (EIC), obtained by deleting all head atoms from σ . For any set of EAICs Σ , we define $\text{St}(\Sigma) = \{\text{St}(\sigma) \mid \sigma \in \Sigma\}$. For ease of presentation (and w.l.o.g.), we assume that constants do not appear in EAICs.

Semantics. We use $\text{pground}(\varphi)$ to denote the set of all *partially ground instances* of a formula φ obtained by replacing universally quantified variables with constants in all possible ways. For a set of formulae Φ , $\text{pground}(\Phi) = \cup_{\varphi \in \Phi} \text{pground}(\varphi)$.

A database D satisfies a partially ground conjunction of literals φ (denoted $D \models \varphi$), if $\varphi^+ \subseteq D$ and there is no substitution ϑ replacing existentially quantified variables in φ with constants s.t. $\vartheta(\varphi^-) \cap D \neq \emptyset$, where φ^+ and φ^- denote the sets of positive and negated atoms in φ , respectively. Thus, for any partially ground EIC σ of the form $\varphi \Rightarrow$, as it expresses a

denial constraint, $D \models \sigma$ iff $D \not\models \varphi$, that is, the following condition holds: if $\text{body}^+(\sigma) \subseteq D$, then there is a substitution ϑ replacing existentially quantified variables with constants s.t. $\vartheta(\text{body}^-(\sigma)) \cap D \neq \emptyset$. Furthermore, D satisfies an EIC σ if D satisfies every partially ground instance in $\text{pground}(\sigma)$; D satisfies an EAIC (or partially ground instance thereof) σ if it satisfies $St(\sigma)$. Finally, D satisfies a set of EAICs (or EICs) Σ if D satisfies every $\sigma \in \Sigma$ —we also say that D is *consistent* w.r.t. Σ . Updates and repairs for databases with EICs and EAICs can be defined analogously to the cases of ICs and AICs, respectively.

Example 5. Consider the database schema consisting of two relations $\text{edge}(\text{Source}, \text{Dest})$ and $\text{node}(\text{Id})$ storing edges and nodes of a graph. The EAIC $\sigma_5: \text{node}(\mathbf{X}) \wedge \nexists \mathbf{Y} \text{edge}(\mathbf{X}, \mathbf{Y}) \Rightarrow \neg \text{node}(\mathbf{X}) \vee \exists \mathbf{Z} +\text{edge}(\mathbf{X}, \mathbf{Z})$ says that every node must have an outgoing edge. When this is not the case either the node is deleted or an outgoing edge is added. The database $D = \{\text{node}(\mathbf{a})\}$ is clearly inconsistent. Since the domain \mathcal{C} is infinite, σ_5 suggests an infinite number of ways to repair the database, namely, by means of update atoms of the form $\{+\text{edge}(\mathbf{a}, \mathbf{c})\}$ with $\mathbf{c} \in \mathcal{C}$. Notice that $\{-\text{node}(\mathbf{a})\}$ is another possible way of restoring consistency.

For any set of EAICs Σ and set of ground update atoms \mathcal{U} , $\Sigma[\mathcal{U}]$ denotes the set of partially ground EAICs derived from $\text{pground}(\Sigma)$ by first deleting every head update atom $\pm A$ for which there does not exist a substitution ϑ such that $\vartheta(\pm A) \in \mathcal{U}$, and then deleting every EAICs where all head update atoms have been deleted.

The definitions of founded update and founded repair are the same as those defined for AICs, that is, for any database D and set of EAICs Σ : (i) an update \mathcal{R} is *founded* iff it is an update for $\langle D, \Sigma[\mathcal{R}] \rangle$, and (ii) a repair $\mathcal{R}(D)$ is *founded* iff \mathcal{R} is a founded.

The introduction of existentially quantified variables increases the expressivity of active integrity constraints. The price to pay is that, differently from the AIC setting, decidability of query answering over knowledge bases with EAICs is no more guaranteed, in general. Example 5 showed that EAICs can admit an infinite number of updates, whereas other EAICs can admit updates of infinite size (i.e., containing an infinite number of update atoms).

To restrict the number of repairs to be considered for query evaluation, we next introduce the concepts of labeled null and universal repairs. A labeled null can be used as a placeholder for any constant from \mathcal{C} . Thus, in addition to the set of constants \mathcal{C} , we assume the existence of an infinite enumerable set of labeled nulls \mathcal{N} of the form \perp_i , where $i \in \mathbb{N}$ is a natural number. For any set of atoms D with values in $\mathcal{C} \cup \mathcal{N} \cup \mathcal{V}$, we use $\mathcal{C}(D)$ (resp. $\mathcal{N}(D)$, $\mathcal{V}(D)$) to denote the set of constants (resp. nulls, variables) occurring in D . For every two sets of atoms D_1 and D_2 over S , a homomorphism h from D_1 to D_2 , denoted $h : D_1 \rightarrow D_2$, is a mapping from $\mathcal{C}(D_1) \cup \mathcal{N}(D_1) \cup \mathcal{V}(D_1)$ to $\mathcal{C}(D_2) \cup \mathcal{N}(D_2) \cup \mathcal{V}(D_2)$ such that: (i) $h(c) = c$, for every $c \in \mathcal{C}(D_1)$; (ii) $h(\perp_i) \in \mathcal{C}(D_2) \cup \mathcal{N}(D_2)$, for every $\perp_i \in \mathcal{N}(D_1)$; (iii) for every fact $R_i(\bar{t})$ of D_1 , we have that $R_i(h(\bar{t}))$ is a fact of D_2 (where, if $\bar{t} = (a_1, \dots, a_n)$, then $h(\bar{t}) = (h(a_1), \dots, h(a_n))$).

A homomorphism that is the identity on $\mathcal{C} \cup \mathcal{N}$ (i.e., it maps variables only) is also called a *substitution*, whereas a substitution whose image is $\mathcal{C} \cup \mathcal{N}$ (resp. \mathcal{C}) is called a *matcher* (resp. *constant matcher*). The concepts of homomorphism can be extended to (sets of) update atoms.

The new definitions of ground (update) atom, update and repair are given in the following. A *ground atom* A is of the form $p(t_1, \dots, t_n)$, where p is an n -ary predicate and $t_1, \dots, t_n \in \mathcal{C} \cup \mathcal{N}$; we write it also as $p(\bar{t})$, where \bar{t} is understood to be a sequence of constants and labeled nulls. Intuitively, A represents all atoms $B = p(c_1, \dots, c_n)$, with $c_1, \dots, c_n \in \mathcal{C}$, such that there

exists a homomorphism from A to B . A *ground update atom* is of the form $+p(\bar{t})$ or $-p(\bar{t})$, where $p(\bar{t})$ is a ground atom. We use $\pm p(\bar{t})$ to refer to a generic ground update atom.

The semantics of a database D with labeled nulls is usually given in terms of the set $\text{POSS}(D)$ of its *possible worlds*, that is, all databases that can be obtained from D by replacing all nulls with constants. The *certain answers* to a query Q over D are $\text{CERTAIN}(Q, D) = \bigcap_{W \in \text{POSS}(D)} Q(W)$.

The definitions of partially ground constraints remains the same. A database D with labeled nulls *satisfies* a partially ground EIC σ if the following condition holds: for every homomorphism h from $\text{body}^+(\sigma)$ to D that maps nulls to constants, there is a constant matcher ϑ s.t. $\vartheta(\text{body}^-(\sigma)) \cap h(D) \neq \emptyset$. The definitions of satisfaction of (sets of) EICs and EAICs remain the same, whereas the definitions of coherent update atoms and update needs to be revised.

A set of ground update atoms \mathcal{U} is *coherent* if there are no two update atoms $+a(t_1), -a(t_2) \in \mathcal{U}$ and a homomorphism h s.t. $a(h(t_1)) = a(h(t_2))$. Let \mathcal{U}_i and \mathcal{U}_j be coherent sets of ground update atoms. \mathcal{U}_i is *more general than* \mathcal{U}_j , denoted $\mathcal{U}_i \sqsupseteq \mathcal{U}_j$, if there exists a homomorphism h from \mathcal{U}_i to \mathcal{U}_j . \mathcal{U}_i and \mathcal{U}_j are (*homomorphically*) *equivalent*, denoted $\mathcal{U}_i \equiv \mathcal{U}_j$, if $\mathcal{U}_i \sqsupseteq \mathcal{U}_j$ and $\mathcal{U}_j \sqsupseteq \mathcal{U}_i$. For instance, $\{+\text{edge}(\perp_1, \perp_2)\} \sqsupseteq \{+\text{edge}(a, \perp_2)\} \sqsupseteq \{+\text{edge}(a, a)\}$, and the sets $\{+\text{edge}(a, \perp_1)\}, \{+\text{edge}(\perp_2, a)\}, \{-\text{node}(a)\}$ are pairwise incomparable.

Definition 2 (Update). *Given a database D and a set of EICs Σ , an update for $\langle D, \Sigma \rangle$ is a coherent set of ground update atoms \mathcal{R} such that (i) $\mathcal{R}(D) \models \Sigma$, and (ii) for every coherent set of ground update atoms \mathcal{R}' such that $\mathcal{R}'(D) \models \Sigma$ if $\mathcal{R}' \sqsupseteq \mathcal{R}$, then also $\mathcal{R} \sqsupseteq \mathcal{R}'$ holds. \square*

Observe that the previous definition coincides with the one provided in Section 2 when applied to AICs, as update atoms contain only constants and $\mathcal{R} \sqsupseteq \mathcal{R}'$ is equivalent to $\mathcal{R} \subseteq \mathcal{R}'$.

Once we have revised the definition of coherent set of ground update atoms, update, founded update atom, founded update and founded repair are defined analogously to the case of AICs. The definition of certain answers has to consider all possible founded repairs for $\langle D, \Sigma \rangle$ and, for each founded repair, all its possible worlds. $\text{CERTAIN}(Q, D, \Sigma) = \bigcap_{\mathcal{R} \in \text{FR}(D, \Sigma) \wedge M \in \text{POSS}(\mathcal{R}(D))} Q(M)$.

Proposition 1 from [1] shows that certain query answering is undecidable in the presence of EAICs. This result does not preclude the existence of interesting classes of EAICs for which the problem of computing certain query answers is decidable. As for AICs, it might be the case that there are no founded updates for a database and set of EAICs. Although the introduction of nulls enlarges the number of (founded) updates, only a subset of these need to be considered in computing certain answers. This idea is captured by a “universal set of founded updates”.

Definition 3 (Universal Set of Updates). *Let D be a database and Σ a set of EAICs. A universal set of updates S is a minimal (w.r.t. \subseteq) set of S -updates for $\langle D, \Sigma \rangle$ s.t. for every update \mathcal{R}_j for $\langle D, \Sigma \rangle$ there is an S -update $\mathcal{R}_i \in S$ s.t. $\mathcal{R}_i \sqsupseteq \mathcal{R}_j$. \square*

Roughly speaking, a *universal set of founded updates* is a set of founded updates that is representative of all founded updates.

Example 6. *Consider the database and the EAIC of Example 5. The founded updates are $\mathcal{R}_0 = \{-\text{node}(a)\}$, every $\mathcal{R}_i = \{+\text{edge}(a, \perp_i)\}$, for some $i \in \mathbb{N}$, and every $\mathcal{R}_c = \{+\text{edge}(a, c)\}$, for some $c \in \mathcal{C}$. The sets of the form $\{\mathcal{R}_0, \mathcal{R}_i\}$ are universal sets of founded updates, whereas the sets of the form $\{\mathcal{R}_0, \mathcal{R}_c\}$, for some $c \in \mathcal{C}$, are not.*

Different interesting results have been shown in [1]. To compute certain query answers, it suffices to consider any universal set of founded updates. As a consequence, certain answers can be computed by only considering the repairs obtained by taking any universal set of founded updates. Also, for databases with EAICs and positive queries, certain answers can be computed by resorting to the naive evaluation of [17]. Finally, since the universal set of founded updates can be infinite, restrictions guaranteeing finiteness and the existence of at most one representative founded update have been presented in [1].

4. Conclusions

The framework presented in this paper finds application in several domains from different fields, including knowledge representation and reasoning and databases. As EAICs generalize classical production rules, they can be used as a basic representation mechanism useful in different contexts, such as automated planning, expert systems and action selection. Regarding data integration, EAICs could be used to implement, using a unified framework, both dataset merging and cleaning mechanisms and to enrich smart data exchange setting [18]. It would be interesting to extend the EAIC language so as to express different kind of preferences, e.g., by incorporating formalisms like CP-nets [19, 20, 21]. Since explaining query (non)entailment has recently received increasing attention (e.g., see [22, 23, 24, 25, 26, 27, 28]), another interesting direction for future work is to investigate notions of explanations in the EAIC framework. Finally, because existential quantification generally leads to undecidability of reasoning tasks, it would be interesting to see how techniques guaranteeing decidability developed for logic programs (e.g., see [29, 30]) might be adapted to the EAIC framework.

References

- [1] M. Calautti, L. Caroprese, S. Greco, C. Molinaro, I. Trubitsyna, E. Zumpano, Existential active integrity constraints, *Expert Syst. Appl.* 168 (2021) 114297.
- [2] M. Arenas, L. E. Bertossi, J. Chomicki, Consistent query answers in inconsistent databases, in: *PODS*, 1999, pp. 68–79.
- [3] M. Calautti, M. Console, A. Pieris, Counting database repairs under primary keys revisited, in: *PODS*, 2019, pp. 104–118.
- [4] T. Lukasiewicz, E. Malizia, C. Molinaro, Complexity of approximate query answering under inconsistency in datalog+/-, in: *IJCAI*, 2018, pp. 1921–1927.
- [5] T. Lukasiewicz, E. Malizia, A. Vaicnavicius, Complexity of inconsistency-tolerant query answering in Datalog+/- under cardinality-based repairs, in: *AAAI*, 2019, pp. 2962–2969.
- [6] T. Lukasiewicz, E. Malizia, M. V. Martinez, C. Molinaro, A. Pieris, G. I. Simari, Inconsistency-tolerant query answering for existential rules, *Artif. Intell.* 307 (2022) 103685.
- [7] S. Greco, C. Molinaro, I. Trubitsyna, Computing approximate query answers over inconsistent knowledge bases, in: *IJCAI*, 2018, pp. 1838–1846.
- [8] J. Wijsen, Database repairing using updates, *ACM TODS* 30 (2005) 722–768.
- [9] S. Greco, C. Molinaro, Approximate probabilistic query answering over inconsistent databases, in: *ER*, 2008, pp. 311–325.

- [10] S. Greco, C. Molinaro, Probabilistic query answering over inconsistent databases, *Annals of Mathematics and Artificial Intelligence* 64 (2012) 185–207.
- [11] S. Flesca, F. Furfaro, F. Parisi, Range-consistent answers of aggregate queries under aggregate constraints, in: *SUM*, 2010, pp. 163–176.
- [12] G. Gottlob, E. Malizia, Achieving new upper bounds for the hypergraph duality problem through logic, *SIAM J. Comput.* 47 (2018) 456–492.
- [13] L. Caroprese, S. Greco, E. Zumpano, Active integrity constraints for database consistency maintenance, *IEEE TKDE* 21 (2009) 1042–1058.
- [14] A. Cali, G. Gottlob, T. Lukasiewicz, A general datalog-based framework for tractable query answering over ontologies, *Journal of Web Semantics* 14 (2012) 57–83.
- [15] L. Caroprese, M. Truszczynski, Active integrity constraints and revision programming, *Theory and Practice of Logic Programming* 11 (2011) 905–952.
- [16] M. Calautti, L. Caroprese, S. Greco, C. Molinaro, I. Trubitsyna, E. Zumpano, Consistent query answering with prioritized active integrity constraints, in: *IDEAS*, 2020, pp. 3:1–3:10.
- [17] T. Imielinski, W. Lipski Jr., Incomplete information in relational databases, *J. ACM* 31 (1984) 761–791.
- [18] S. Greco, E. Masciari, D. Saccà, I. Trubitsyna, HIKE: A step beyond data exchange, in: *ER*, 2019, pp. 423–438.
- [19] T. Lukasiewicz, E. Malizia, On the complexity of mcp-nets, in: D. Schuurmans, M. P. Wellman (Eds.), *AAAI*, 2016, pp. 558–564.
- [20] T. Lukasiewicz, E. Malizia, Complexity results for preference aggregation over (m)cp-nets: Pareto and majority voting, *Artif. Intell.* 272 (2019) 101–142.
- [21] T. Lukasiewicz, E. Malizia, Complexity results for preference aggregation over (m)cp-nets: Max and rank voting, *Artif. Intell.* 303 (2022) 103636.
- [22] I. I. Ceylan, T. Lukasiewicz, E. Malizia, A. Vaicnavicius, Explanations for query answers under existential rules, in: *IJCAI*, 2019, pp. 1639–1646.
- [23] I. I. Ceylan, T. Lukasiewicz, E. Malizia, C. Molinaro, A. Vaicnavicius, Preferred explanations for ontology-mediated queries under existential rules, in: *AAAI*, 2021, pp. 6262–6270.
- [24] T. Lukasiewicz, E. Malizia, C. Molinaro, Explanations for negative query answers under inconsistency-tolerant semantics, in: *IJCAI*, 2022.
- [25] I. I. Ceylan, T. Lukasiewicz, E. Malizia, A. Vaicnavicius, Explanations for ontology-mediated query answering in description logics, in: *ECAI*, 2020, pp. 672–679.
- [26] I. I. Ceylan, T. Lukasiewicz, E. Malizia, C. Molinaro, A. Vaicnavicius, Explanations for negative query answers under existential rules, in: *KR*, 2020, pp. 223–232.
- [27] T. Lukasiewicz, E. Malizia, C. Molinaro, Explanations for inconsistency-tolerant query answering under existential rules, in: *AAAI*, 2020, pp. 2909–2916.
- [28] L. Caroprese, I. Trubitsyna, M. Truszczynski, E. Zumpano, A measure of arbitrariness in abductive explanations, *Theory Pract. Log. Program.* 14 (2014) 665–679.
- [29] M. Calautti, S. Greco, C. Molinaro, I. Trubitsyna, Logic program termination analysis using atom sizes, in: *IJCAI*, 2015, pp. 2833–2839.
- [30] M. Calautti, S. Greco, C. Molinaro, I. Trubitsyna, Checking termination of logic programs with function symbols through linear constraints, in: *Proc. RuleML*, 2014, pp. 97–111.