Ann Math Artif Intell (2018) 83:213–246
https://doi.org/10.1007/s10472-018-9577-y

Active integrity constraints for general-purpose knowledge bases

Luís Cruz-Filipe[1] · Graça Gaspar[2] · Isabel Nunes[2] ·
Peter Schneider-Kamp[1]

Published online: 14 March 2018

Abstract In the database world, *integrity constraints* are essential to guarantee database integrity. The related problem of database repair deals with finding the best way to change a database so that it satisfies its integrity constraints. These two topics have been studied intensively since the 1980s. The formalism of *active integrity constraints*, proposed in 2004, aims at addressing them jointly, by providing a syntax whereby a particular subclass of integrity constraints can be specified together with preferred ways to repair inconsistency. In the last decade, several authors have proposed adaptations of the notion of integrity constraints to other reasoning frameworks than relational databases. In this article, we extend this line of work in two ways. First, we target multi-context systems, a general-purpose framework for combining heterogeneous reasoning systems, able to model most other reasoning frameworks, as we demonstrate. Second, we extend the notions of active integrity constraints and grounded repairs to this generalized setting. This way of including repair suggestions inside integrity constraints, subject to a validity check, enables us to

Supported by the Danish Council for Independent Research, Natural Sciences, grants DFF-1323-00247 and DFF-7014-00041, and by FCT/MCTES/PIDDAC under center grant to BioISI (Center Reference: UID/MULTI/04046/2013).

✉ Luís Cruz-Filipe
lcf@imada.sdu.dk

Graça Gaspar
mdgaspar@fc.ul.pt

Isabel Nunes
minunes@fc.ul.pt

Peter Schneider-Kamp
petersk@imada.sdu.dk

[1] Department of Mathematics and Computer Science, University of Southern Denmark, Campusvej 55, DK-5230 Odense, Denmark

[2] BioISI—Biosystems, Integrative Sciences Institute, Faculty of Sciences, University of Lisbon, Campo Grande, P-1749-016 Lisbon, Portugal

define simple iterative algorithms to find all possible grounded repairs for an inconsistent multi-context system, avoiding the need to solve complex or undecidable problems.

Keywords Integrity constraints · Multi-context systems · Repairs · Ontologies

Mathematics Subject Classification (2010) 03B80 · 68P15

1 Introduction

Integrity constraints in databases have been around for decades, and are universally acknowledged as one of the essential tools to ensure database consistency [2]. The associated problem of finding out how to repair an inconsistent database – i.e., change it so that it again satisfies the integrity constraints – was soon recognized as an important and difficult one [1], which is unlikely to be solvable in a completely automatic way [27]. Typically, inconsistent databases can be repaired in several different ways, and it is important to have the means to establish preferences between different possible repairs. Active integrity constraints [31] were originally introduced as a way to address this issue in a restricted setting – first-order integrity constraints that can be written in denial clausal form.

In parallel with relational databases, many research efforts have been spend in considering more powerful reasoning systems, where information is not entirely described explicitly, but may be inferred by logical means. This work started in the early 1980s with the proposal of deductive databases, and became more diversified since the turn of the century.

In this setting, an important topic of study is how to combine the reasoning capabilities of different systems, preferably preserving the properties that make them useful in practice – e.g. consistency, decidability of reasoning, efficient computation. One of the most general frameworks to combine reasoning systems abstractly is that of heterogeneous non-monotonic multi-context systems [7]. Besides being studied from a theoretical perspective, these have been implemented, and many specialized versions have been introduced to deal with particular aspects deemed relevant in practice [22, 30, 35, 49]. In this article, we work with managed multi-context systems [9], a first-order generalization of the original systems, which were based on a propositional syntax.

As a very simple kind of reasoning system, databases can naturally be viewed as particular cases of MCSs. In this paper we propose to define active integrity constraints in MCSs in a way that naturally generalizes the corresponding theory for relational databases. In particular, we explore how to write clausal integrity constraints in MCSs, and what we should consider as possible repair actions. Our approach differs from previous suggestions on how to model integrity constraints in MCSs [8, 9, 26, 45]: in those works, the authors routinely embed integrity constraints *into* the system, thereby making them part of the reasoning mechanism – unlike the situation in databases, where they form an independent layer whose purpose is only to signal whether the database is in a consistent state. We argue that integrity constraints for MCSs should also follow this principle, and show how our approach is also in line with investigations on how to add integrity constraints to other reasoning frameworks, namely description logic knowledge bases [29, 40]. Due to the richer structure of MCSs, we can define two distinct notions of consistency with respect to integrity constraints, which coincide in the case of databases. Furthermore, our formalism gives us enough information from which we can compute repairs for an inconsistent MCSs, as long as its entailment relation is decidable.

Of particular interest is the application to the ontology domain. Integrity constraints for ontologies have been discussed in recent years, with several approaches on how to define them and how to check their satisfaction [28, 40, 42]. Given its challenges, the more complex problem of repairing ontologies that do not satisfy their integrity constraints has not received as much attention. We dedicate a section of this presentation to the study of AICs over ontologies, and show that we can use our algorithms to find repairs for ontologies that are inconsistent with respect to a set of AICs.

Part of the material included in this article has been previously presented in [20, 21].

1.1 Contributions

We summarize the main contributions of this work.

Active integrity constraints for multi-context systems Our main contribution is the notion of active integrity constraints over an MCS, which in particular yields a uniform notion of active integrity constraint over several formalisms, together with notions of weak and strong satisfaction of these. We show that the problem of deciding whether an MCS satisfies a set of AICs is polynomial-time reducible to the problem of deciding whether an MCS has a model. We show how our definition captures the traditional notion of denial clausal integrity constraints over relational databases, and how it naturally generalizes this concept to distributed databases and deductive databases. We also compare our definition with existing proposals for integrity constraints over ontology languages.

Grounded repairs Our second contribution is a notion of grounded repair for an MCS that is inconsistent with respect to a set of active integrity constraints, together with algorithms for computing grounded repairs. In the general setting of MCSs, the benefits of working with AICs are much more substantial than in the database case: while in the database case every clausal IC can be transformed into an AIC automatically, in the general case such a transformation would require solving complex abduction problems [37]. Using AICs, we can automatically compute repairs for inconsistent MCSs, bypassing the need to solve such reasoning problems, as long as we prove that an AIC is valid (Definition 13). This should in general not pose a problem: integrity constraints are written with a very clear semantic idea in mind, typically by an engineer with a deep knowledge of the underlying system, and the validity of the repair actions proposed should thus follow easily. We argue in more detail for this point in the remainder of the article. Thus, in practice, the complexity involved in computing each repair is moved to a one-time verification of validity of AICs.

Application to ontologies As an example of the expressivity of our framework, we show that we can capture all types of integrity constraints over ontologies identified as relevant in [28]. Furthermore, we show how they can be systematically written as *valid* active integrity constraints, from which we can automatically generate grounded repairs for ontologies that do not respect them.

1.2 Outline

This article is structured as follows.

Section 2 presents an overview of related work that is most directly relevant to us. We summarize the theory of active integrity constraints for databases and the relevant notions from the framework of managed multi-context systems in Section 3.

In Section 4, we propose our definition of active integrity constraints over MCSs, together with notions of weak and strong satisfaction, repair and grounded repair. We show how satisfaction of integrity constraints can be reduced to the problem of logical consistency, obtaining decidability and complexity results. We also discuss how the repair actions allowed in an AIC should be restricted, and provide simple criteria for establishing validity of AICs. As a sanity check, we show that our active integrity constraints generalize the corresponding concept in relational databases. We conclude by presenting algorithms to compute grounded repairs, inspired by their database counterparts.

In Section 5 we look at how our notions particularize to other frameworks, namely distributed databases and deductive databases, compare to previous proposals for defining integrity constraints in those systems, and provide some complexity results.

Section 6 focuses on the case of ontologies. We discuss how an ontology can be viewed as a multi-context system, and show that our notion of active integrity constraint captures previous proposals for modeling integrity constraints in this formalism, while helping with the computation of repairs. We evaluate our formalism by showing that we can capture the classes of integrity constraints identified in [28].

We discuss some design options and possible extensions in Section 7, before concluding with an overview of our results in Section 8.

2 Related work

2.1 Integrity constraints for databases

The topic of integrity constraints has been extensively studied in the literature. In this section, we discuss the work that we feel to be more directly relevant to the tasks we carry out in this paper.

Integrity constraints and updates – ways of repairing inconsistent databases – were identified as a seminal problem in database theory almost thirty years ago [1]. The case for viewing integrity constraints as a layer on top of the database, rather than as a component of it, has been made since the 1980s. The idea is that data inconsistencies captured by integrity constraints need to be resolved, but they should not necessarily interfere with the ability to continue using the database. In this line, much work has been done e.g. in query answering from inconsistent databases [3, 47], by ensuring that the only answers generated are those that hold in minimally repaired versions of the database.

Integrity constraints are typically grouped in different syntactic categories [51]. Many important classes can be expressed as first-order formulas, and can also be written in denial (clausal) form – the fragment expressable in our formalism.

Whenever an integrity constraint is violated, the database must be *repaired* to regain consistency. The problem of database repair is to determine whether such a transformation is possible, and many authors have investigated algorithms for computing database repairs efficiently. Typically, there are several possible ways of repairing an inconsistent database, and several criteria have been proposed to evaluate them. Minimality of change [27, 54] demands that the database be changed as little as possible, while the common-sense law of inertia [44] states that every change should have an underlying reason. While these criteria narrow down the possible database repairs, it is commonly accepted that human interaction is ultimately required to choose the “best” possible repair [50]. (A similar argument has also been made in other contexts related to resolving inconsistencies, see e.g. [24].)

2.2 Active integrity constraints for databases

The formalism of active integrity constraints, introduced in [31], addresses the issue of choosing among several possible repairs. An active integrity constraint specifies not only an integrity constraint, but it also gives indications on how inconsistent databases can be repaired through the inclusion of *update actions*, which can be the addition and the removal of tuples from the database – a minimal set that can implement the three main operations of database updates [1].

The original, declarative, semantics of AICs defined *founded* repairs [10], in which every action is *supported*: it occurs in the head of a constraint that is violated if that action were not included in the repair. Despite this characterization, there are unnatural founded repairs where two actions mutually support each other, but do not have support from other actions – a phenomenon that the authors called *circularity of support*.

The same authors then proposed *justified* repairs [13]. These have several other problems: its verification is challenging, due to a quantification over all subsets of atoms in the database; the authors' claim that they avoid circularity of support has never been formally substantiated; and there are examples where justified repairs are over-restrictive [17]. Moreover, the formal definition of justified repairs is technically complicated, and lacks a clear intuition. Finally, the definition of justified repairs is intrinsically linked to the syntactic structure of databases, and cannot be adapted to other knowledge representation formalisms.

Grounded repairs [16] form a middle ground between both semantics, requiring support not only for individual actions, but also for arbitrary subsets of the repair. They are grounded fixed points of the intuitive operation of "applying one action from the head of each AIC that is not satisfied", which is in line with the intuitive motivation for studying AICs. They are an instance of the general notion of grounded fixpoints [6], and as such have the properties of being buildable "from the ground up" (thereby avoiding circularity of support) and minimal under set inclusion. Grounded repairs are a particular case of semantics for AICs that can be defined using approximation fixpoint theory [5].

Founded and justified repairs can be computed via revision programming [13]. Alternatively, an operational semantics for AICs [17] was implemented for SQL databases [18]. There, repairs are leaves of particular trees, yielding a semantics equivalent to the declarative one when existence of a repair is an NP-complete problem. For grounded and justified repairs, where this existence problem is Σ_2^p-complete, the trees still contain all repairs, but may also include spurious leaves – requiring a post test that brings the overall complexity to the theoretical limit.

2.3 Multi-context systems

Multi-context systems were originally inspired by the work of McCarthy [39] and developed initially by the Trento School [33, 34]. We follow the modern formulation of heterogeneous non-monotonic multi-context systems [7], which gave rise to an extremely powerful formalism that is aimed at the Semantic Web. Multi-context systems can be informally described as collections of logic knowledge bases – the *contexts* – connected by Datalog-style *bridge rules*.

Since their introduction, several variants of multi-context systems have been proposed that add to their potential fields of application. Relational multi-context systems [30] were proposed as a way to allow a formal first-order syntax, introducing variables and aggregate expressions in bridge rules, and extending their semantics accordingly. Managed multi-context systems [9] proceed this line of generalization by abstracting from the possible actions that change individual knowledge bases. Dynamic multi-context systems [23] were intended to cope with situations where knowledge sources and their contents may change over time and are not known *a priori*. Evolving multi-context systems [36] extend this idea by incorporating knowledge resulting from dynamic observations through different belief change operations with different levels of persistence. This framework has been further extended with the notion of evolving bridge rules [36].

2.4 Integrity constraints in other frameworks

The first authors to consider deductive databases [4, 32] also discussed this issue. They identify three ways to look at deductive databases: by viewing the whole system as a first-order theory; by viewing it as an extensional database together with integrity constraints; and a mixed view, where some rules are considered part of the logic theory represented by the database, and others as integrity constraints identifying preferred models. In [4], it is argued that this third approach is the correct one, as it cleanly separates rules that are meant to be used in logic inferencing from those that only specify consistency requirements.

More recently, authors have considered adding integrity constraints to other types of reasoning systems such as ontologies. This integration poses several challenges, mainly due to the open-world assumption and the absence of the unique name assumption [28, 41, 43, 48]. In this context, integrity constraints are conventionally modeled as T-Box axioms [40], but variants based on hybrid knowledge bases, auto-epistemic logic, modal logic, and grounded circumscription have recently been proposed. (An overview of these proposals can be found in Section 2 in [42].)

The authors of [40] also discuss why integrity constraints should be kept separate from the logical theory, even if they can be syntactically written in the language of the ontology. Therefore, they separate the axioms in the T-Box (the deductive part of an ontology) into two groups: reasoning rules, which are used to infer new information, and integrity constraints, which only verify the consistency of the knowledge state without changing it.

The setting of multiple ontologies was also considered in [29]. This work considers the problem of combining information from different knowledge sources while guaranteeing the overall consistency, and preserving this consistency when one of the individual ontologies is changed. This is achieved by external integrity constraints, written in a Datalog-like syntax, which can refer to knowledge in different ontologies in order to express relationships between them. Again, the purpose of these rules is uniquely to identify incompatibilities in the data, and not to infer new information.

By contrast, the authors who have discussed integrity constraints in multi-context systems have not felt the need to take a similar approach. Integrity constraints appear routinely in examples in e.g. [8, 9, 26, 45], but always encoded within the system, so that their violation leads to logical inconsistency of the global knowledge base. These works focus instead on the aspect of identifying the sources of inconsistencies – integrity constraints being only one example, not given any special analysis. Following this line of research, the authors of [25] propose a declarative policy language, together with methodologies to apply it, to provide a means to create policies to avoid or repair inconsistencies in multi-context systems in a controlled way (e.g. specifying which inconsistencies can be repaired automatically, and which ones need external input by a human operator).

3 Background

In this section we summarize the theory and results that are directly relevant for our development.

3.1 Active integrity constraints

Active integrity constraints were originally introduced in [31], and their denotational semantics subsequently developed in [10, 13]. Our presentation follows that of [13], albeit with a different notation.

Let Σ be a first-order signature without function symbols. A database is a set of ground atoms over Σ, and an update action is an expression of the form $+a$ or $-a$, where a is an atom (possibly containing variables) over Σ. An *active integrity constraint* (AIC) over a database DB is a rule r of the form

$$p_1, \ldots, p_m, \mathsf{not}\ (p_{m+1}), \ldots, \mathsf{not}\ (p_\ell) \Longrightarrow \alpha_1 \mid \cdots \mid \alpha_k \tag{1}$$

where each p_i is an atom over the database's signature, for $1 \leq i \leq \ell$, every variable free in $p_{m+1}, \ldots, p_\ell$ occurs in $p_1, \ldots, p_m$, and each update action α_i is either $-p_j$ for some $1 \leq j \leq m$ or $+p_j$ for $m < j \leq \ell$. The *body* of r is $\mathsf{body}(r) = p_1, \ldots, p_m, \mathsf{not}\ (p_{m+1}), \ldots, \mathsf{not}\ (p_\ell)$, and the *head* of r is $\mathsf{head}(r) = \{\alpha_1, \ldots, \alpha_k\}$. The body of an active integrity constraint expresses an integrity constraint in denial clausal form, i.e. in the form $\forall \left(\bigvee l_i\right)$ (universally quantified negation of a disjunction of literals).

The original proposal of AICs [31] also allowed for existentially quantified variables to occur in negative literals. This possibility was not discussed in later work, and we do not consider it here.

If r is ground, then DB *satisfies* r, denoted $\mathsf{DB} \models r$, if $\mathsf{DB} \not\models p_i$ for some $1 \leq i \leq m$ or $\mathsf{DB} \models p_i$ with $m < i \leq \ell$. If r contains variables, then $\mathsf{DB} \models r$ if DB satisfies all ground instances of r. Otherwise, r is *applicable* in DB [31]. If η is a set of AICs, then $\mathsf{DB} \models \eta$ if $\mathsf{DB} \models r$ for every $r \in \eta$.

A set of ground update actions $\mathcal{U}$ is *consistent* if it does not contain both $+a$ and $-a$ for any ground atom a. Given a consistent $\mathcal{U}$, we write $\mathcal{U}(\mathsf{DB})$ for the result of applying all actions in $\mathcal{U}$ to DB:

$$\mathcal{U}(\mathsf{DB}) = (\mathsf{DB} \cup \{a \mid +a \in \mathcal{U}\}) \setminus \{a \mid -a \in \mathcal{U}\}\,.$$

Definition 1 Let DB be a database over a signature Σ and η be a set of active integrity constraints over Σ. A consistent set of ground update actions $\mathcal{U}$ over Σ is a *weak repair* for $\langle \mathsf{DB}, \eta \rangle$ if:

- every action in $\mathcal{U}$ changes DB, i.e. if $+a \in \mathcal{U}$, then $a \notin \mathsf{DB}$, and if $-a \in \mathcal{U}$, then $a \in \mathsf{DB}$;
- $\mathcal{U}(\mathsf{DB}) \models \eta$.

Furthermore, $\mathcal{U}$ is a *repair* if $\mathcal{V}(\mathsf{DB}) \not\models \eta$ for every $\mathcal{V} \subsetneq \mathcal{U}$.

In order to take into account the preferences expressed by the heads of AICs, several different types of repairs have been considered. In this work we focus on *grounded* repairs [5, 16]: they prevent circularities in support for repair actions (unlike founded repairs [13]), and their definition is not strictly tied to the syntax of databases (unlike justified repairs [13]).

Definition 2 Let DB be a database over a signature Σ and η be a set of active integrity constraints over Σ. A repair $\mathcal{U}$ for $\langle \mathsf{DB}, \eta \rangle$ is *grounded* if, for every $\mathcal{V} \subsetneq \mathcal{U}$, there exists a ground instance r of a rule in η such that:

- $\mathcal{V}(\mathsf{DB}) \not\models r$;
- $\mathsf{head}(r) \cap (\mathcal{U} \setminus \mathcal{V}) \neq \emptyset$.

Although the original semantics for AICs was declarative, later work showed that the different types of repairs for an inconsistent database with respect to a set of AICs could be computed by building a tree where each node has a descendant for each repair action that solves an inconsistency [17, 18]. In particular, grounded repairs can be computed as follows: starting with the original database, iteratively choose an AIC whose body is satisfied and apply one of the actions in its head; continue with this updated database until all AICs are satisfied. This procedure is guaranteed to produce all grounded repairs; however, it also produces some repairs that are not grounded; therefore, every time a repair is found, it is still necessary to check that the second condition in Definition 2 holds. This can not be overcome, since this algorithm runs in polynomial time and deciding whether a grounded repair exists is a Σ_2^p-complete problem.[1]

3.2 Multi-context systems

We now briefly introduce multi-context systems (MCSs). Intuitively, these are a collection of logic knowledge bases – the *contexts* – connected by Datalog-style *bridge rules*.

We work with *managed multi-context systems* [9], which are a specialization of *relational multi-context systems* [30]. We present our definitions in several steps, starting with relational logic, moving to relational multi-context systems and their semantics, and finally introducing managed multi-context systems. In this way, the complex framework we actually work in is easier to understand, and we can illustrate the different notions by intermediate examples.

Brewka and Eiter motivate their abstract definition of relational logic as follows [7]. The syntax of a logic L is defined by a set KB_L of its well-formed knowledge bases, which we assume to be sets. The language of L is the set of elements occurring in any of its knowledge bases, i.e., the set of its well-formed formulas. We assume that a subset of this language is inductively generated from a signature Σ_L. The semantics of L is given by the set of its possible belief sets BS_L, which are syntactic representations of an agent's possible beliefs. (This will often also be sets of formulas.) Again, the signature Σ_L generates a subset of all possible beliefs. Finally, we use a function ACC_L to characterize the sets of beliefs consistent with a particular knowledge base kb. For classical, monotonic logics, $\mathsf{ACC}_L(\mathsf{kb})$ typically contains only the set of consequences of kb (see Example 1 below); however, in non-monotonic logics there may be multiple acceptable belief sets for a given knowledge base.

Definition 3 A *relational logic* L is a tuple $\langle \mathsf{KB}_L, \mathsf{BS}_L, \mathsf{ACC}_L, \Sigma_L \rangle$, where:

- KB_L is a set of sets; the elements of KB_L are called *well-formed knowledge bases* of L, and the elements of each $\mathsf{kb} \in \mathsf{KB}_L$ are called *well-formed formulas*;

[1]See page 15 for a definition of the polynomial hierarchy, including the complexity class Σ_2^p.

- BS_L is a set of sets; the elements of BS_L are called *possible belief sets*;
- $\mathsf{ACC}_L : \mathsf{KB}_L \to 2^{\mathsf{BS}_L}$ is a function assigning to each knowledge base a set of belief sets, called its *acceptable sets of beliefs*;
- Σ_L is a signature consisting of sets P_L^{KB} and P_L^{BS} of predicate names (with associated arity) and a universe U_L of object constants, such that $U_L \cap (P_L^{\mathsf{KB}} \cup P_L^{\mathsf{BS}}) = \emptyset$.

As informally described above, the purpose of the signature Σ_L is to identify a first-order sublanguage of L. Given $c_1, \ldots, c_k \in U_L$, and p with arity k, we require that:

- if $p \in P_L^{\mathsf{KB}}$, then $p(c_1, \ldots, c_k)$ must be an element of some knowledge base (i.e., it is a well-formed formula);
- if $p \in P_L^{\mathsf{BS}}$, then $p(c_1, \ldots, c_k)$ must be an element of some belief set (i.e., it is a well-formed possible belief).

The elements in the sublanguage generated by Σ_L are called *relational ground elements*, while the remaining elements of knowledge bases or belief sets are called *ordinary*. The signature Σ_L is a characteristic of relational logics, by contrast with the notion of logic introduced in the original works on heterogeneous non-monotonic multi-context systems [7]. Having a first-order sublanguage of L allows us to have bridge rules (defined below) that use variables as first-class citizens in relational multi-context systems, unlike previous models, where rules with variables were seen only as short-hand for the set of all their ground instances.

Example 1 We can see first-order logic over a first-order signature Σ_{FOL} as a logic $\mathsf{FOL} = \langle \mathsf{KB}_{\mathsf{FOL}}, \mathsf{BS}_{\mathsf{FOL}}, \mathsf{ACC}_{\mathsf{FOL}}, \Sigma_{\mathsf{FOL}} \rangle$, where $\mathsf{KB}_{\mathsf{FOL}}$ is the set of sets of well-formed formulas over Σ_{FOL}, $\mathsf{BS}_{\mathsf{FOL}}$ is the set of first-order interpretations over Σ_{FOL}, and $\mathsf{ACC}_{\mathsf{FOL}}$ maps each set of formulas to the set of its models. This logic only contains relational elements.

Definition 4 Let $\mathfrak{I}$ be a finite set of indices, $\{L_i\}_{i \in \mathfrak{I}}$ be a set of relational logics, and V be a set of (first-order) variables distinct from predicate and constant names in any L_i. A *relational element* of L_i over V has the form $p(t_1, \ldots, t_k)$, where $p \in P_{L_i}^{\mathsf{KB}} \cup P_{L_i}^{\mathsf{BS}}$ has arity k and each t_j is a term from $V \cup U_{L_i}$, for $1 \leq j \leq k$. A *relational k-bridge rule* over $\{L_i\}_{i \in \mathfrak{I}}$ and V is a rule of the form

$$(k : s) \leftarrow (c_1 : p_1), \ldots, (c_q : p_q), \mathsf{not}\,(c_{q+1} : p_{q+1}), \ldots, \mathsf{not}\,(c_m : p_m) \tag{2}$$

such that $k, c_i \in \mathfrak{I}$, s is an ordinary or a relational knowledge base element of L_k and each p_i is either an ordinary or a relational belief of L_{c_i}, for $1 \leq i \leq m$.

The notation $(c : p)$ indicates that p is evaluated in context c. In other words, bridge rules allow one particular context (k) to be changed based on queries that are posed to other contexts. These rules intuitively generalize logic programming rules, and as usual in that context we impose a *safety condition*: all variables from V occurring in $p_{q+1}, \ldots, p_m$ must also occur at least once in $p_1, \ldots, p_q$.

Definition 5 A *relational multi-context system* is a collection $M = \{C_i\}_{i \in \mathfrak{I}}$ of contexts $C_i = \langle L_i, \mathsf{kb}_i, \mathsf{br}_i, D_i \rangle$, where:

- L_i is a relational logic;
- kb_i is a knowledge base;
- br_i is a set of relational i-bridge rules;

– D_i is a set of import domains $D_{i,j}$, with $j \in \mathfrak{I}$, i.e., $D_i = \{D_{i,j} \mid j \in \mathfrak{I}\}$, such that $D_{i,j} \subseteq U_{L_j}$.

Import domains define which constants are exported from one context to another: since the underlying logic languages can be different, these sets are essential to allow one context to reason about individuals introduced in another. Import domains are used in the semantics of relational MCSs (given below) to restrict the possible instantiations of bridge rules: the inclusion $D_{i,j} \subseteq U_{L_j}$ requires that bridge rules yielding conclusions in context C_j can only be instantiated with constants from the universe U_{L_j}. One might wonder why we do not also require the inclusion $D_{i,j} \subseteq U_{L_i}$: this is actually undesirable, since we can use bridge rules to propagate information about an individual that is represented by different names in different contexts. Following [9], we assume that $D_{i,j}$ is the finite domain consisting of the object constants appearing in kb_j or in the head of a relational bridge rule in br_j, unless otherwise stated.

For simplicity of notation, when working with multi-context systems we write only the index i instead of L_i in the components of the logic L_i, i.e., we assume $L_i = \langle \mathsf{KB}_i, \mathsf{BS}_i, \mathsf{ACC}_i, \Sigma_i \rangle$. Using this convention the condition in the last item of Definition 5 above becomes $D_{i,j} \subseteq U_j$.

Example 2 Let C_1 and C_2 be contexts over the first-order logic FOL with R and Rt binary predicates in Σ_{FOL}, and let $\mathsf{kb}_1 = \mathsf{kb}_2 = \emptyset$. We can use the following bridge rules in br_2 to define Rt in C_2 as the transitive closure of R in C_1.

$$(2 : \mathsf{Rt}(x, y)) \leftarrow (1 : \mathsf{R}(x, y)) \qquad (2 : \mathsf{Rt}(x, y)) \leftarrow (1 : \mathsf{R}(x, z)), (2 : \mathsf{Rt}(z, y))$$

We use the MCS $M = \langle C_1, C_2 \rangle$ to exemplify the concepts we introduce below.

The semantics of relational MCSs is defined in terms of ground instances of bridge rules: the instances obtained from each rule $r \in \mathsf{br}_i$ by uniform substitution of each variable X in r by a constant in $\bigcap D_{i,j}$, with j ranging over the indices of the contexts to which queries containing X are made in r.

Definition 6 A *belief state* for M is a collection $S = \{S_i\}_{i \in \mathfrak{I}}$ where $S_i \in \mathsf{BS}_i$ for each $i \in \mathfrak{I}$ – i.e., a tuple of belief sets, one for each context. The ground bridge rule (2) is *applicable* in a belief state S if $p_i \in S_{c_i}$ for $1 \leq i \leq q$ and $p_i \notin S_{c_i}$ for $q < i \leq m$. The set of the heads of all applicable ground instances of bridge rules of context C_i w.r.t. S is denoted by $\mathsf{app}_i(S)$. An *equilibrium* is a belief state S such that $S_i \in \mathsf{ACC}_i(\mathsf{kb}_i \cup \mathsf{app}_i(S))$ for each $i \in \mathfrak{I}$.

Intuitively, an equilibrium is a belief state S that is stable under application of all the bridge rules, i.e., if we consider all ground instances of all bridge rules whose bodies are satisfied by S and we add their heads to the original knowledge base, then S is an acceptable set of beliefs. An MCS is *(logically) consistent* if it admits at least one equilibrium.

Particular types of equilibria (minimal, grounded, well-founded) [7] can be defined for relational MCSs, but they are not used in this work and we do not discuss them here.

Example 3 In the setting of the previous example, all equilibria of M must include the transitive closure of R in S_1 in the interpretation of Rt in S_2. For example, if we take $S = \langle S_1, S_2 \rangle$ with $S_1 = \{\mathsf{R(a, b)}, \mathsf{R(b, c)}\}$ and $S_2 = \{\mathsf{Rt(a, b)}, \mathsf{Rt(b, c)}, \mathsf{Rt(a, c)}\}$, then S is an equilibrium. However, $S' = \langle S_1, S_2' \rangle$ with $S_2' = \{\mathsf{Rt(a, b)}, \mathsf{Rt(b, c)}\}$ is not an equilibrium, as it does not satisfy the second bridge rule.

Checking whether an MCS has an equilibrium is known as the *consistency problem* in the literature. We refer to this property as *logical consistency* (to distinguish from consistency with respect to integrity constraints, defined in the next section) throughout this paper. This problem has been studied extensively [8, 24, 26, 53]; its decidability depends on decidability of reasoning in the underlying contexts. The complexity of checking logical consistency of an MCS M depends on the context complexity of M – the highest complexity of deciding consistency in one of the contexts in M (cf. [26] for a formal definition and known results).

Managed multi-context systems further generalize relational MCSs by abstracting over the possible actions on the heads of bridge rules.

Definition 7 A *managed multi-context system* is a collection of *managed contexts* $\{C_i\}_{i\in\mathfrak{I}}$, with each $C_i = \langle L_i, \mathsf{kb}_i, \mathsf{br}_i, D_i, \mathsf{OP}_i, \mathsf{mng}_i\rangle$ as follows.

- $L_i = \langle \mathsf{KB}_i, \mathsf{BS}_i, \mathsf{ACC}_i, \Sigma_i\rangle$ is a relational logic, $\mathsf{kb}_i \in \mathsf{KB}_i$ is a knowledge base, and D_i is a set of import domains, as in relational MCSs.
- OP_i is a set, whose elements are called *operation names*.
- $\mathsf{mng}_i : \wp(\mathsf{OP}_i \times \bigcup \mathsf{KB}_i) \times \mathsf{KB}_i \to \mathsf{KB}_i$ is a *management function*.
- br_i is a set of *managed bridge rules*: syntactic entities with the form described by (2), but where s is of the form $o(p)$ with $o \in \mathsf{OP}_i$ and $p \in \bigcup \mathsf{KB}_i$.

The intuition is as follows: heads of bridge rules can now contain arbitrary actions (identified by the labels in OP_i), and the management function specifies the semantics of these labels.[2] The use of $\bigcup \mathsf{KB}_i$ (the set of all well-formed formulas over the logic L_i) in mng_i and br_i reflects the fact that any formula can be given as argument to the management function, regardless of the actual state of the knowledge base. Equilibria for managed MCSs are defined exactly as in Definition 6, but with the obvious adaptations to range over managed bridge rules.

Example 4 Every relational MCS can be seen as a managed MCS where $\mathsf{OP}_i = \{\mathsf{add}\}$ for every context i, with the semantics $\mathsf{mng}_i(\{\langle \mathsf{add}, s\rangle\}, \mathsf{kb}) = \mathsf{kb} \cup \{s\}$.

We typically write operation names applied to knowledge base elements, rather than as pairs – e.g., in the previous example, we write $\mathsf{add}(s)$ rather than $\langle \mathsf{add}, s\rangle$.

4 Active integrity constraints

In this section we introduce our main notion: active integrity constraints over managed multi-context systems. We consider the problem of satisfying a set of AICs, and show decidability and complexity results for this problem. We also discuss with repair actions should be allowed on the heads of AICs, and show criteria for automatically establishing validity of AICs. After defining grounded repairs for a multi-context system in an inconsistent state with respect to a set of AICs and discussing how to compute them, we show that our notions and results generalize previous work on AICs over databases by viewing databases as MCSs.

[2]Our definition is slightly simplified from that in [9], where the management function can return several possible effects for each action.

4.1 Key definitions

The definition of managed multi-context system include a set of operation names OP_c for each context c. These operations are used to change the knowledge base in the process of computing equilibria. However, when we consider the process of attempting to repair inconsistencies, not all of these operations make sense: we should only allow those operations that also model updates or revisions of the context's knowledge base. Consider as an example the case of deductive databases. There are two types of tables in these databases: extensional tables and intensional tables (views). The latter can only be manipulated by means of deductive rules, while the former can be changed directly. Inconsistencies in deductive databases should thus only be repaired by changing the extensional tables.

In order to model this behavior, we assume a distinguished subset $\mathsf{OP}^*_c \subseteq \mathsf{OP}_c$ of *update operation names* of each set of operation names in each context c. As we will see, these are the only operations that we allow when attempting to repair inconsistencies. In examples, we assume $\mathsf{OP}^*_c = \mathsf{OP}_c$ unless otherwise stated explicitly; in Sections 5.2 and 6 we present examples where $\mathsf{OP}^*_c \subsetneq \mathsf{OP}_c$.

Definition 8 Let $M = \{C_i\}_{i\in\Im}$ be a managed multi-context system and V be a set of (first-order) variables distinct from predicate and constant names in any L_i. An *active integrity constraint* over M[3] is a rule r of the form

$$(c_1 : p_1), \ldots, (c_m : p_m), \text{not } (c_{m+1} : p_{m+1}), \ldots, \text{not } (c_\ell : p_\ell) \Longrightarrow (c'_1 : \alpha_1) \mid \cdots \mid (c'_k : \alpha_k) \tag{3}$$

where, for $1 \leq i \leq \ell$ and $1 \leq j \leq k$:

- c_i, c'_j are context identifiers (elements of $\Im$);
- each p_i is an ordinary or a relational belief in C_{c_i};
- each α_j, which we call an *update action*, is a pair consisting of an operation name from $\mathsf{OP}^*_{c_j}$ and a set of ordinary or relational knowledge base elements from L_{c_j};[4]
- all variables from V in $p_{m+1}, \ldots, p_\ell$ occur in $p_1, \ldots, p_m$ (safety condition).

This definition follows the syntax of AICs for databases (1), and we define *body* and *head* of r similarly. The body of an AIC can also be read as an integrity constraint in denial clausal form.

We again impose the same safety condition as previously. It can be slightly relaxed to capture general tuple-generating dependencies, by allowing $p_{m+1}, \ldots, p_\ell$ to introduce new variables with the restriction that they can be used only once in the whole rule. Although this generalization poses no significant changes to the theory, it makes the presentation heavier, so we do not consider it hereafter.

Syntactically, active integrity constraints are similar to disjunctive bridge rules. However, we give them a different semantics: bridge rules are directly used in defining equilibria, and thus every equilibrium satisfies every bridge rule; integrity constraints are meant to be checked against equilibria, so equilibria may satisfy them or not (as we formalize in Definition 10). This separation between the internal logic of reasoning systems and formulas that express its consistency can be found in previous work in several settings [4, 14, 29, 40,

[3]Technically, over M and V, but we leave V implicit hereafter.

[4]Thus, heads of grounded bridge rules are elements of $\mathsf{OP}_{c_j} \times \bigcup \mathsf{KB}_{c_j}$.

42], although (interestingly) not in the setting of MCSs, where previous work [8, 9, 26, 45] routinely presents examples of integrity constraints integrated into the knowledge bases.

However, by keeping integrity constraints separate from the data, we do not restrict the models of MCSs, in particular avoiding issues of logical inconsistency. This allows to treat violation of integrity constraints as indicative of an error in the model or in the data, preventing it from deriving additional inferences; furthermore, when we take into account the heads of active integrity constraints, we can find ways to repair the inconsistency that are clearly distinguished from inferences inside the system.

These considerations are similar to those made in Section 2.7 of [40] and in [29], in the (more restricted) context of integrity constraints over description logic knowledge bases. Likewise, several authors defend an approach for integrity constraints in databases where inconsistencies should be brought to the users' attention, but not affect the semantics of the database [1, 27]. In particular, it may be meaningful to work with reasoning systems not satisfying integrity constraints (see [47] for databases and [45] for description logic knowledge bases). Our approach is also in line with [9], where it is argued that in MCSs it is important to "distinguish data from additional operations on it".

Example 5 Continuing the example from the previous section, we can write an active integrity constraint over M stating that the relation R (in context C_1) must be transitive, and if this is not the case, then the missing fact about it must be added.

$$(2 : \mathsf{Rt}(x, y)), \mathsf{not}\ (1 : \mathsf{R}(x, y)) \Longrightarrow (1 : \mathsf{add}(\mathsf{R}(x, y))) \tag{4}$$

Note that we use the embedding of relational MCSs into managed MCSs given in Example 4.

In the following sections, we work with ground instances of AICs. As with bridge rules, we need to consider only instantiations that map each variable X occurring in an AIC r to a constant in $\bigcap D_{i,j}$, with j ranging over the indices of the contexts to which queries containing X are made in $\mathsf{body}(r)$ and i ranging over the indices of the contexts appearing in $\mathsf{head}(r)$ whose actions contain X. We implicitly assume all instantiations to satisfy this property hereafter.

4.2 Satisfaction

Determining whether active integrity constraints are satisfied is done at the level of equilibria. Since this is a property of the body of the AICs, which, as mentioned above, are intuitively integrity constraints in denial clausal form, we omit the qualifier "active" in this subsection.

Definition 9 Let $M = \{C_i\}_{i\in\mathfrak{I}}$ be an MCS and $S = \{S_i\}_{i\in\mathfrak{I}}$ be a belief state for M. Then S *satisfies* the integrity constraint (3) if, for every instantiation θ of the variables in $p_1, \dots, p_m$, either $p_k\theta \notin S_{c_k}$ for some $1 \le k \le m$ or $p_k\theta \in S_{c_k}$ for some $m < k \le \ell$.

In other words: equilibria must satisfy all bridge rules (if their body holds, then so must their heads), but they may or may not satisfy all integrity constraints. In this sense, integrity constraints express preferences among equilibria.

Definition 10 Let M be an MCS and η be a set of integrity constraints.

1. *M strongly satisfies η*, $M \models_s \eta$, if:

   (a) M is logically consistent;
   (b) every equilibrium of M satisfies all integrity constraints in η.

2. *M weakly satisfies η*, $M \models_w \eta$, if there is an equilibrium of M that satisfies all integrity constraints in η.

Example 6 The equilibrium S from Example 3 does not satisfy the integrity constraint (4), thus M does not strongly satisfy this formula. However, M weakly satisfies this constraint, as seen by considering the equilibrium $S'' = \langle S_1', S_2' \rangle$ with S_2' as above and where $S_1' = \{\mathsf{R}(\mathsf{a},\mathsf{b}), \mathsf{R}(\mathsf{b},\mathsf{c}), \mathsf{R}(\mathsf{a},\mathsf{c})\}$.

We say that M is (strongly/weakly) *consistent* with respect to a set of integrity constraints η if M (strongly/weakly) satisfies η, and *inconsistent* otherwise.[5] These two notions express different interpretations of integrity constraints. Strong satisfaction views them as necessary requirements, imposing that all models of the MCS must satisfy them. Examples of these are the usual integrity constraints over databases, which express semantic connections between relations that must always hold. Weak satisfaction views integrity constraints as expressing preferences: the MCS may have several equilibria, and those that satisfy the integrity constraints are considered to be "better".

The distinction is also related to the use of brave (credulous) or cautious (skeptical) reasoning. If M strongly satisfies a set of integrity constraints η, then any inferences drawn from M using brave reasoning are guaranteed to hold in some equilibrium that also satisfies η. If, however, M only weakly satisfies η, then this no longer holds, and we can only use cautious reasoning if we want to be certain that any inferences are still compatible with η.

Since strong and weak satisfaction both require M to be logically consistent, $M \models_s \eta$ implies $M \models_w \eta$. In particular, the problems of deciding whether $M \models_s \eta$ and $M \models_w \eta$ are both at least as hard as deciding whether M has an equilibrium – and thus undecidable in the general case. When logical consistency of M is decidable and its set of equilibria is enumerable, weak satisfaction is semi-decidable (if there is an equilibrium that satisfies η, we eventually encounter it), while strong satisfaction is co-semi-decidable (if there is an equilibrium that does not satisfy η, we eventually encounter it). Furthermore, we have the following results.

Theorem 1 *Weak satisfaction of integrity constraints is reducible to logical consistency in linear time (in the size of the MCS).*

Proof To decide whether $M \models_w \eta$, construct M' by extending M with a context C_0 where $\mathsf{KB}_0 = \wp(\{*\})$, $\mathsf{kb}_0 = \emptyset$, $\mathsf{ACC}_0(\emptyset) = \{\emptyset\}$, $\mathsf{ACC}_0(\{*\}) = \emptyset$, and the bridge rules are obtained by replacing the head of every rule in η with $(0 : *)$. Then M' has an equilibrium iff $M \models_w \eta$: any equilibrium of M not satisfying η corresponds to a belief state of M' where $\mathsf{app}_0(S) = \{*\}$, which is never an equilibrium of M'; but equilibria of M satisfying η give rise to equilibria of M' taking $S_0 = \emptyset$. □

Theorem 2 *Strong satisfaction of integrity constraints is reducible to logical inconsistency in linear time (in the size of the MCS).*

[5] We deal with two different notions of (in)consistency in this work: the one just defined, and the property of having an equilibrium, which is the one usually considered in the literature. To distinguish them, we reserve the term *logical (in)consistency* for the latter.

Proof Construct M' as before, but now defining $\mathsf{ACC}_0(\emptyset) = \emptyset$, $\mathsf{ACC}_0(\{*\}) = \{\{*\}\}$. If M is inconsistent, then $M \not\models_s \eta$. If M is consistent, then any equilibrium of M satisfying η corresponds to a belief state of M' where $\mathsf{app}_0(S) = \emptyset$, which can never be an equilibrium of M'; in turn, equilibria of M not satisfying η give rise to equilibria of M' taking $S_0 = \{*\}$. So if M is consistent, then $M \models_s \eta$ iff M' is inconsistent. □

The reductions given in the proofs of these two theorems are constructible in time linear in the size of the set η, and independent of M. Combining them with the well-known complexity results for consistency checking (Table 1 in [26]), we directly obtain complexity results for these decision problems. We use the standard denotations of complexity classes in the polynomial hierarchy: $\Sigma_0^P = \Delta_0^P = P$ contains the decision problems solvable in polynomial time; Δ_{n+1}^P contains the decision problems solvable in polynomial time given an oracle for decision problems in Σ_n^P; and Σ_{n+1}^P contains the decision problems solvable in non-deterministic polynomial time given an oracle for decision problems in Σ_n^P. In particular, $NP = \Sigma_1^P$.

These complexity results rely on the notion of context complexity. In the following definition, the set Out_i contains all beliefs in C_i that occur in the body of some bridge rule in M.

Definition 11 Let $M = \{C_i\}_{i\in\mathfrak{I}}$ be a relational multi-context system. The *context complexity* of context C_i, with $i \in \mathfrak{I}$, is the computational complexity of deciding, given $H \subseteq \bigcup \mathsf{KB}_i$ and $T \subseteq \mathsf{Out}_i = \{p \mid (i, p) \in \mathsf{body}(r), r \in \mathsf{br}_j, j \in \mathfrak{I}\}$, whether there exists a belief state $S_i \in \mathsf{ACC}_i(\mathsf{kb}_i \cup H)$ such that $S_i \cap \mathsf{Out}_i = T$.

The context complexity of M is $\mathcal{C}$ if: every context in M has context complexity at most $\mathcal{C}$, and there is at least one context in M with context complexity $\mathcal{C}$.

For instance, if the context complexity of M is Σ_2^P, then there is at least one context C_i in M with context complexity Σ_2^P. This is the complexity of deciding, given a set H of well-formed formulas in the logic of C_i and a subset T of well-formed formulas from C_i that occur in the bodies of ground instances of bridge rules in M, whether there is a belief state S_i for C_i such that: (i) S_i is an acceptable set of beliefs with respect to the information in the knowledge base kb_i and in H, and (ii) the set of elements of S_i that occur in the bodies of rules in M is precisely T. The complexity of logical consistency depends on the context complexity of M, since testing whether a belief state is an equilibrium requires solving decision problems similar to the one in the definition of context complexity.

Corollary 1 *The complexity of deciding whether $M \models_w \eta$ or $M \models_s \eta$, depending on the context complexity of M, $\mathcal{CC}(M)$, is given in* Table 1.

These results also imply that integrity constraints can be modeled in MCSs simply by adding them as bridge rules whose head is a special atom interpreted as inconsistency (the approach taken in e.g. [24]), but as argued at the end of the previous section we gain from keeping them separate.

Table 1 Complexity of integrity checking of an MCS in terms of its context complexity

$\mathcal{CC}(M)$	P	NP	Σ_i^P	PSPACE	EXPTIME
$M \models_w \eta$	NP	NP	Σ_i^P	PSPACE	EXPTIME
$M \models_s \eta$	Δ_2^P	Δ_2^P	Δ_{i+1}^P	PSPACE	EXPTIME

4.3 Validity

The actions in the head of active integrity constraints are meant to resolve the inconsistency detected by its body. In the database world, this property is enforced by means of a simple syntactic check. However, the reasoning capabilities of MCSs imply that such a simple restriction is not possible, and we replace it by semantic criteria: we require that each action must be capable of solving the inconsistency in some "reasonable" state of the system. It is also not reasonable to require (as in databases [10]) that every action in $\mathsf{head}(r)$ be able to solve every inconsistency detected by $\mathsf{body}(r)$: since inconsistencies may be triggered by derived information, they may have different origins, and the different actions may be solutions for those different causes.

Example 7 We illustrate this point by means of a concrete toy example: a deductive database with two unary base relations p and q, a view consisting of a relation r such that $\mathsf{r}(x) \leftrightarrow \mathsf{p}(x) \vee \mathsf{q}(x)$, and the integrity constraint $\neg\mathsf{r}(\mathsf{a})$.

We model this database as an MCS $M = \langle C_E, C_I \rangle$ where C_E is an extensional database including predicates p and q (but not r), C_I is the view context including predicate r (but not p or q), and they are connected by the bridge rules

$$(I : \mathsf{add}(\mathsf{r}(X))) \leftarrow (E : \mathsf{p}(X)) \qquad (I : \mathsf{add}(\mathsf{r}(X))) \leftarrow (E : q(X))\,.$$

Furthermore, mng_E and mng_I allow addition and removal of any tuples to C_E and C_I, respectively, using operations add and del. We also assume $\mathsf{OP}_E^* = \mathsf{OP}_E$ and $\mathsf{OP}_I^* = \emptyset$.[6]

Due to the structure of M, $\mathsf{r}(\mathsf{a})$ can only be obtained as a deduction from $\mathsf{p}(\mathsf{a})$ or $\mathsf{q}(\mathsf{a})$ (or both); furthermore, it is reasonable to assume that the definition of the views does not change (in contrast to the actual information in the extensional database), so it makes sense to write an AIC

$$(I : \mathsf{r}(\mathsf{a})) \Longrightarrow (E : \mathsf{del}(\mathsf{p}(\mathsf{a}))) \mid (E : \mathsf{del}(\mathsf{q}(\mathsf{a})))\,.$$

The actions on the head of this AIC solve the problem in all future states of M. However, restoring consistency may require performing both actions (if the database contains both $\mathsf{p}(\mathsf{a})$ and $\mathsf{q}(\mathsf{a})$).

This example illustrates an important point that is implicit also in the database case: active integrity constraints (both their body and the repair actions they suggest) are written under the assumption that there is a "structural" part of the MCS that does not change. In the database case, this is the database's underlying signature – if tables are added or removed, for example, then clearly the integrity constraints associated to the database need to be revised (or at least examined to see whether they still make sense). In a managed MCS, we use the management function to capture the system's allowed variability.

Definition 12 The set of *variants* to an MCS M, denoted $\mathsf{vrt}(M)$, is

$$\mathsf{vrt}(M) = \{\mathcal{U}(M) \mid \mathcal{U} \text{ is a finite set of update actions over } M\}\,.$$

The restrictions on the actions in the head of AICs range over $\mathsf{vrt}(M)$, which contains all possible future evolutions of M.

[6]In Section 5.2 we describe the systematic construction of an MCS from a deductive database, of which this is a particular instance.

Definition 13 An AIC r of the form (3) is (weakly/strongly) *valid* with respect to an MCS M if:

(i) for every logically consistent $M' \in \mathsf{vrt}(M)$ such that $M' \not\models r$, there is $\mathcal{U} \subseteq \mathsf{head}(r)$ with $\mathcal{U}(M') \models r$;
(ii) for every $\alpha \in \mathsf{head}(r)$, there is $M' \in \mathsf{vrt}(M)$ such that $M' \not\models r$ and $\alpha(M') \models r$.

These conditions require that the set of suggested actions be complete (it can solve all inconsistencies) and that it not contain useless actions.

Example 8 The AIC in Example 7 is valid: the only possible changes to M are in kb_E, which only contains information about p and q, thus, in any element of $\mathsf{vrt}(M)$ the only way to derive $\mathsf{r(a)}$ is still from either $\mathsf{p(a)}$ or $\mathsf{q(a)}$. The second condition follows by considering the two variants of M obtained by taking $\mathsf{kb}_E = \{\mathsf{p(a)}\}$ or $\mathsf{kb}_E = \{\mathsf{q(a)}\}$.

Theorem 3 *Deciding whether an AIC is valid is in general undecidable.*

Proof Let L be any logic with an undecidable entailment problem, C be a context over L with $\mathsf{add} \in \mathsf{OP}_C$ such that $\mathsf{mng}_C(\mathsf{add}(\varphi), \Gamma) = \Gamma \cup \{\varphi\}$, and $M = \{C\}$. Assume also that $\mathsf{vrt}(M)$ includes all knowledge bases over L. Then $(C : \neg B) \Longrightarrow (C : \mathsf{add}(A))$ is valid iff $A \models_L B$. □

In practice, though, proving validity of AICs should not pose a problem: AICs are written by humans with a very precise semantic motivation in mind, and this means that the conditions in Definition 13 should be simple for a human to prove. In general, the following simple criteria are useful for establishing validity of AICs over an MCS M.

– If the body of an AIC includes $(c : p)$ such that p does not occur in the head of any bridge rule in M, then including an action "remove p from context c" in the head of the AIC guarantees condition (i), and that action satisfies condition (ii).
– If the body of an AIC includes $(c : p)$ such that p is defined exclusively by means of bridge rules, then including in its head actions that negate the bodies of those bridge rules guarantees condition (i), and those actions all satisfy condition (ii).

These criteria are used repeatedly in the examples in Sections 5 and 6.

4.4 Repairs and their computation

We now consider the problem of repairing an MCS that is inconsistent with respect to a set of active integrity constraints η. Again, the situation is more complex than in the database case. In the database case, we defined repairs to be sets of update actions that could be applied simultaneously, and required consistency in order to make this application well defined. However, in an MCS, we are again faced with the problem that this requirement cannot be expressed syntactically, since two apparently unrelated update actions can interfere with each other depending on the management function.

The consistency requirement we impose on sets of update actions for MCSs is therefore more restrictive: we require that the order in which actions are executed be irrelevant for the end result. This is a strong condition – the management function for each context is defined over sets of actions, so it must in particular be able to process sets including contradictory actions –, but the algorithms we later introduce for computing repairs are iterative, building repairs one action at a time, and therefore need this invariance property.

Definition 14 Let $M = \{C_i\}_{i \in \mathfrak{I}}$ be an MCS, $\mathcal{U}$ be a finite set of update actions, and $\mathcal{U}_i$ be the set of actions in $\mathcal{U}$ affecting C_i. $\mathcal{U}_i$ is *consistent* with respect to kb_i if, for every permutation $\alpha_1, \ldots, \alpha_k$ of the elements of $\mathcal{U}_i$, $\mathsf{mng}_i(\mathcal{U}_i, \mathsf{kb}_i) = \mathsf{mng}_i(\alpha_1, \mathsf{mng}_i(\ldots, \mathsf{mng}_i(\alpha_k, \mathsf{kb}_i)\ldots))$. $\mathcal{U}$ is consistent with respect to M if each $\mathcal{U}_i$ is consistent with respect to kb_i, and in this case we write $\mathcal{U}(M)$ for the result of applying each $\mathcal{U}_i$ to each kb_i.

Definition 15 Let $M = \{C_i\}_{i \in \mathfrak{I}}$ be an MCS, η be a set of AICs over M and $\mathcal{U}$ be a finite set of update actions. $\mathcal{U}$ is a *weak repair* for $\langle M, \eta \rangle$ if $\mathcal{U}$ is consistent with respect to M and $\mathcal{U}(M) \models \eta$. Furthermore, $\mathcal{U}$ is a *grounded repair* if: for every $\mathcal{V} \subsetneq \mathcal{U}$, there is an AIC $r \in \eta$ such that $\mathcal{V}(M) \not\models r$ and $\mathsf{head}(r) \cap (\mathcal{U} \setminus \mathcal{V}) \neq \emptyset$.

In [17], we showed how to use active integrity constraints to compute repairs for inconsistent databases, by using the actions in the head of unsatisfied AICs to build a *repair tree* whose leaves were the repairs. We showed how the construction of the tree could be adapted to the different types of repairs considered originally in [13]; in particular, for the case of grounded repairs (which is the one we are interested in this work), it is enough to expand each node with the actions in the heads of the AICs that are not satisfied in that node. This constructibility "from the ground up" is a general property of grounded fixpoints [6]), which embodies the common-sense law of inertia [44]. The property of being a grounded repair also implies minimality under inclusion [16], which guarantees the principle of minimality of change [54].

The adaptation of the notion of repair tree to the framework of AICs over MCSs requires some changes to the previous algorithms. We show that it still allows us to construct all grounded repairs for a given (inconsistent) MCS automatically, as long as entailment in all contexts is decidable.

In the context of computing repairs, we say that an AIC r is *applicable* to an MCS M if $M \not\models r$.

Definition 16 Let M be an MCS and η be a set of active integrity constraints over M. The *repair tree* for $\langle M, \eta \rangle$, $\mathcal{T}_{\langle M, \eta \rangle}$, is defined as follows.

- Each node is a set of update actions.
- A node n is *consistent* if:
  - $n(M)$ is logically consistent;
  - if n' is the parent of n, then n is a consistent set of update actions with respect to $n'(M)$.
- Each edge is labeled with a closed instance of a rule.
- The root of the tree is the empty set $\emptyset$.
- For each consistent node n and rule r, if $n(M) \not\models r$ then $n' = n \cup \mathcal{U}$ is a child of n if:
  - $\mathcal{U} \subseteq \mathsf{head}(r)$;
  - $n'(M) \models r$;
  - if $\mathcal{U}' \subseteq \mathcal{U}$ then $(n \cup \mathcal{U}')(M) \not\models r$.

In the database case [17], repair trees are trivially finite, since the syntactic restrictions on AICs over databases guarantee that each rule can only be applied at most once in every branch. In the general MCS case, this is not true, as the following example shows.

Example 9 Consider an ontology with four concepts B_1, B_2, B_3 and D, where the T-Box contains axioms $B_1 \sqsubseteq D$ and $B_2 \sqcap B_3 \sqsubseteq D$, and the A-Box is $\{B_1(a), B_3(a)\}$.

We represent this ontology as an MCS with two contexts T and A, corresponding to the T-Box and A-Box, respectively. Both contexts are based on the description logic underlying the ontology, with ACC being the usual first-order consequence operator; the information flow from the A-Box and T-Box is obtained by the bridge rules

$$(T : \mathsf{add}(B_i(X))) \leftarrow (A : B_i(X))$$

and the management function on A allows addition and deletion of instances, while the management function on T only allows addition. Furthermore, we take $\mathsf{OP}^*_A = \mathsf{OP}_A$ and $\mathsf{OP}^*_T = \emptyset$. (See Section 6 for the general construction of an MCS from an ontology, of which this is an instance, and the discussion of the design options.)

Over this ontology we write the following active integrity constraints.

$$(T : D(a)) \Longrightarrow (A : \mathsf{del}(B_1(a))) \mid (A : \mathsf{del}(B_3(a))) \qquad (r_1)$$

$$\mathsf{not}\ (T : B_1(a)), \mathsf{not}\ (T : B_2(a)) \Longrightarrow (A : \mathsf{add}(B_2(a))) \qquad (r_2)$$

Constructing the repair tree as described above, we obtain

$$\begin{tikzcd}
\emptyset \arrow[d, "r_1"] \\
\{\mathsf{del}(B_1(a))\} \arrow[d, "r_2"] \\
\{\mathsf{del}(B_1(a)), \mathsf{add}(B_2(a))\} \arrow[d, "r_1"] \\
\{\mathsf{del}(B_1(a)), \mathsf{add}(B_2(a)), \mathsf{del}(B_3(a))\}
\end{tikzcd}$$

and the leaf of this tree is a grounded repair for the system. Note however that the same ground instance of rule r_1 was applied twice.

Nevertheless, we can establish termination and completeness of this construction.

Lemma 1 *Let M be a managed multi-context system and η be a finite set of active integrity constraints over M. Then the tree $\mathcal{T}_{\langle M,\eta\rangle}$ is finite.*

Proof By definition, every node of $\mathcal{T}_{\langle M,\eta\rangle}$ has a finite number of descendants, since there are only finitely many ground instances of AICs with a finite number of actions in each one's head. By construction, in every branch the labels of the nodes form an increasing sequence (with respect to set inclusion), and each node is again a subset of the (finite) set of all actions in the heads of all rules. Therefore, $\mathcal{T}_{\langle M,\eta\rangle}$ has finite depth and finite degree, hence it is finite. □

Lemma 2 *Let M and η be as above. Every grounded repair for $\langle M, \eta\rangle$ is a leaf of $\mathcal{T}_{\langle M,\eta\rangle}$.*

Proof Let $\mathcal{U}$ be a grounded repair for M and η. By definition of grounded repair, if $\mathcal{U}' \subsetneq \mathcal{U}$ then there is a ground instance r of an AIC such that: there exists $\mathcal{V} \subseteq \mathsf{head}(r) \cap \mathcal{U}$ such that $(\mathcal{U}' \cup \mathcal{V})(M) \models r$. This directly yields a branch of the repair tree ending at $\mathcal{U}$. □

(This is essentially the same argument for showing that, in the database case, grounded repairs are well-founded, see [16].)

The construction of $\mathcal{T}_{\langle M,\eta\rangle}$ is similar to that of the well-founded repair tree in the database case [17].[7] In both cases, this tree may, in general, contain leaves that are not grounded repairs [16]. Under the assumption that $\mathrm{P} \neq \mathrm{NP}$, this cannot be avoided, since existence of grounded repairs for databases is already a Σ_2^p-complete problem [16].

Theorem 4 *Let M be a managed multi-context system that is inconsistent with respect to a set of active integrity constraints η. Given an oracle that decides whether an MCS satisfies η, the problem of deciding whether there exists a grounded repair for $\langle M, \eta\rangle$ is Σ_2^p-complete.*

Proof The proof of Lemma 1 shows that the depth of $\mathcal{T}_{\langle M,\eta\rangle}$ is polynomial in the size of the ground instances of η, which is also polynomial on η (since M is fixed). Given an oracle that decides whether an MCS satisfies a set of AICs, we can find a grounded repair for M, when one exists, as follows: find the right leaf of $\mathcal{T}_{\langle M,\eta\rangle}$ (this can be done in non-deterministic polynomial time, guessing which rule to apply at each node and using the oracle to decide whether the descendant is a leaf), and validate that it is a grounded repair (again in co-NP time: if the set of actions is not consistent, we guess a permutation that yields a different result when applied to M and check this in polynomial time; if it is not a grounded repair, then we guess the subset that violates the definition and use the oracle to confirm this). $\square$

4.5 Relation with databases

The development of AICs for multi-context systems throughout this section was always motivated by the existing theory for the database case. In this subsection, we show how to model databases as managed multi-context systems, and show that our formalism faithfully captures the previous work on AICs for databases.

Definition 17 Let DB be a database. The context generated by DB, $\mathsf{Ctx}(\mathsf{DB})$, is defined as follows.

- The underlying logic is first-order logic over the database's signature Σ (Example 1).
- Belief sets are sets of ground literals over Σ.
- The knowledge base is DB.
- For all kb, there is only one belief set compatible with kb: $\mathsf{ACC}(\mathsf{kb}) = \{\mathsf{kb}^{\vdash}\} = \{\mathsf{kb} \cup \{\neg a \mid a \notin \mathsf{kb}\}\}$.
- The set of bridge rules is empty.

We can see any database DB as a single-context MCS consisting of exactly the context $\mathsf{Ctx}(\mathsf{DB})$; we also denote this MCS by $\mathsf{Ctx}(\mathsf{DB})$, as this poses no ambiguity. The only equilibrium for $\mathsf{Ctx}(\mathsf{DB})$ is $\mathsf{DB}^{\vdash}$ itself, corresponding to the usual closed-world semantics of relational databases. Previous work (cf. [9, 26]) implicitly treats databases in this way, although Ctx is not formally defined.

[7] There is also a notion of repair tree for databases in [17], but it relies on the ability of inferring heads of AICs automatically, which does not exist in the MCS setting.

Let DB be a database and r be an active integrity constraint over DB (in the sense of e.g. [13]).[8] We write r as an active integrity constraint over $\mathsf{Ctx}(\mathsf{DB})$ as follows: if r is

$$p_1, \ldots, p_m, \mathsf{not}\ (p_{m+1}), \ldots, \mathsf{not}\ (p_\ell) \Longrightarrow \alpha_1 \mid \cdots \mid \alpha_k$$

then $\mathsf{br}(r)$ is

$$(1 : p_1), \ldots, (1 : p_m), \mathsf{not}\ (1 : p_{m+1}), \ldots, \mathsf{not}\ (1 : p_\ell) \Longrightarrow (1 : \alpha_1) \mid \cdots \mid (1 : \alpha_k)\,.$$

The following result follows straightforwardly from the definition of $\mathsf{Ctx}(\mathsf{DB})$.

Theorem 5 *Let* DB *be a database and* η *be a set of AICs over* DB. *Then* DB *satisfies all AICs in* η *iff* $\mathsf{Ctx}(\mathsf{DB}) \models_s \mathsf{br}(\eta)$ *iff* $\mathsf{Ctx}(\mathsf{DB}) \models_w \mathsf{br}(\eta)$, *where* br *is extended to sets in the standard way.*

Since every database (viewed as an MCS) has exactly one equilibrium, weak and strong satisfaction of active integrity constraints coincide.

Lemma 3 *Every AIC over a database* DB *yields a valid AIC over* $\mathsf{Ctx}(\mathsf{DB})$.

Proof Recall that $\mathsf{Ctx}(\mathsf{DB})$ has exactly one equilibrium. If DB does not satisfy the body of (1), then it can always be repaired by performing exactly one of the actions in its head [12], establishing both conditions for validity. □

Corollary 2 *Let* DB *be a database and* η *a finite set of AICs over* DB. *A set of update actions* $\mathcal{U}$ *is a grounded repair for* $\langle \mathsf{DB}, \eta \rangle$ *iff* $\mathcal{U}$ *is a grounded repair for* $\langle \mathsf{Ctx}(\mathsf{DB}), \mathsf{br}(\eta) \rangle$.

Proof Straightforward by applying Theorem 5. □

5 Specializations

In this section we show how our notion of AICs for MCSs can be specialized to two particular types of reasoning systems: distributed databases and deductive databases. When applicable, we also compare to previous proposals for integrity constraints for those systems.

5.1 Distributed databases

Distributed databases are databases that store their information at different sites in a network, typically including information that is duplicated at different nodes [52] in order to promote resilience of the whole system.

A distributed database consisting of individual databases $\mathsf{DB}_1, \ldots, \mathsf{DB}_n$ can be modeled as an MCS with n contexts $\mathsf{Ctx}(\mathsf{DB}_1), \ldots, \mathsf{Ctx}(\mathsf{DB}_n)$. The internal consistency of the database, in the sense that tables that occur in different DB_is must have the same rows, can

[8] As discussed earlier, the original presentation in [31] allowed more general tuple-generating dependencies; modeling these requires allowing singleton variables in the B_is, which as discussed earlier would not pose significant changes to our theory. If we take first-order logic with equality as the underlying logic for $\mathsf{Ctx}(\mathsf{DB})$, we can also write AICs whose body is an equality-generating constraint.

be specified as integrity constraints over this MCS as follows. For each relation p, let $\gamma(p)$ be the number of columns of p and $\delta(p)$ be the set of indices of the databases containing p. Then the set of all AICs of the form

$$(i : p(x_1, \ldots, x_{\gamma(p)})), \mathsf{not}\ (j : p(x_1, \ldots, x_{\gamma(p)})) \Longrightarrow \alpha_1, \ldots, \alpha_k$$

where p is a relation and $i, j \in \delta(p)$ logically specify the integrity of the system (each body corresponds to an inconsistency). Different strategies for fixing inconsistencies in distributed databases can be implemented by adequately choosing the repair actions.

- To define a repair strategy based on "always add missing information" (meaningful, for example, if we know updates to the database to consist only of additions), we have only the action $(j : \mathsf{add}(p(x_1, \ldots, x_{\gamma(p)})))$ in the head of each AIC.
- Dually, a repair strategy based on "always remove extra information" would be achieved by having only the action $(i : \mathsf{del}(p(x_1, \ldots, x_{\gamma(p)})))$ in the head of each AIC.
- Majority vote can be implemented by writing $(i : \mathsf{del}(p(x_1, \ldots, x_{\gamma(p)}))) \mid (j : \mathsf{add}(p(x_1, \ldots, x_{\gamma(p)})))$ as the head of each AIC, and choosing repairs that are minimal with respect to cardinality.

A more sophisticated repair strategy is described below.

Example 10 Another interesting possibility is choosing a repair strategy that always picks the information on the most recently updated node. To implement this, we need to add a context C_0 to the MCS that tracks this information, with $\mathsf{OP}^*_{C_0} = \emptyset$. This context has a knowledge base containing a single table $\mathsf{most\_recent}(p, k_p)$ indicating, for each relation p, the index of the most recently updated context among those in $\delta(p)$, and bridge rules

$$(0 : \mathsf{add}(p(x_1, \ldots, x_{\gamma(p)}))) \leftarrow (0 : \mathsf{most\_recent}(p, i)), (i : p(x_1, \ldots, x_{\gamma(p)}))$$

for every context i. Note that this is implemented not as a higher-order bridge rule, but as a set of all rules for the possible values of i and p.

The active integrity constraints now become

$$\begin{aligned}
&(i : p(x_1, \ldots, x_{\gamma(p)})), \mathsf{not}\ (j : p(x_1, \ldots, x_{\gamma(p)})),\\
&(0 : \mathsf{most\_recent}(p, k)), (k : p(x_1, \ldots, x_{\gamma(p)})) \Longrightarrow (j : \mathsf{add}(p(x_1, \ldots, x_{\gamma(p)})))\\
&(i : p(x_1, \ldots, x_{\gamma(p)})), \mathsf{not}\ (j : p(x_1, \ldots, x_{\gamma(p)})),\\
&(0 : \mathsf{most\_recent}(p, k)), \mathsf{not}\ (k : p(x_1, \ldots, x_{\gamma(p)})) \Longrightarrow (i : \mathsf{del}(p(x_1, \ldots, x_{\gamma(p)})))
\end{aligned}$$

for all relations p and $i, j, k \in \delta(p)$.

A similar approach could be used to implement majority vote, bypassing the quantification over all repairs suggested earlier.

In this case, we can internalize the integrity constraints as bridge rules of the form

$$(j : \mathsf{add}(p(x_1, \ldots, x_{\gamma(p)}))) \leftarrow (i : p(x_1, \ldots, x_{\gamma(p)}))\,.$$

but these significantly change the semantics of the database: instead of describing preferred equilibria, they impose a flow of information between nodes.

Example 11 Consider a country with a central person register (CPR), mapping a unique identifying number to the name and current address of each citizen using a relation person, e.g. person(1111111118, *old_lady*, *gjern*). Furthermore, each electoral district keeps a local voter register using a relation voter, e.g. voter(1111111118), and a list of addresses local to the given electoral district using a relation $\mathsf{address}$, e.g. $\mathsf{address}(gjern)$. Then the active integrity constraints[9]

$$(\mathsf{Skbg} : \mathsf{voter}(Id)), \mathsf{not}\ (\mathsf{CPR} : \mathsf{person}(Id)) \Longrightarrow (\mathsf{Skbg} : \mathsf{del}(\mathsf{voter}(Id)))$$
$$(\mathsf{Skbg} : \mathsf{voter}(Id)), (\mathsf{CPR} : \mathsf{person}(Id, Add)),$$
$$\mathsf{not}\ (\mathsf{Skbg} : \mathsf{address}(Add)) \Longrightarrow (\mathsf{Skbg} : \mathsf{del}(\mathsf{voter}(Id)))$$

ensure that all voters registered in the Silkeborg electoral district are registered in the central person register, and that they are registered with an address that is local to the Silkeborg electoral district. If any of these conditions fails, then the only acceptable solution is to remove that person from the voter register.

In addition, the following set of active integrity constraints (where C_i ranges over $ED \setminus \{\mathsf{Skbg}\}$) models the fact that each person registered in the Silkeborg electoral district cannot be registered in any other electoral districts from the set ED; if that is the case, then the person must be removed from one of the registers.

$$(\mathsf{Skbg} : \mathsf{voter}(Id)), (C_i : \mathsf{voter}(Id)) \Longrightarrow (\mathsf{Skbg} : \mathsf{del}(\mathsf{voter}(Id))) \mid (C_i : \mathsf{del}(\mathsf{voter}(Id)))$$

If there is a single voter v that appears in the register for both Silkeborg and one or more elements of ED, the grounded repairs for the database are either $\{(\mathsf{Skbg} : \mathsf{del}(\mathsf{voter}(\mathsf{v})))\}$ or $\{(C_i : \mathsf{del}(\mathsf{voter}(\mathsf{v}))) \mid C_i \in ED, \mathsf{voter}(\mathsf{v}) \in \mathsf{kb}_i\}$. In general, the grounded repairs for databases that are inconsistent with respect to this active integrity constraint are unions of sets of the above forms, one for each voter.

This section's treatment of distributed databases is equivalent to considering their disjoint union as a database. Consequently, there is no need to use MCSs for distributed databases, but this mapping shows that our theory abstracts the practice in this field. Furthermore, results in previous work [15] indicate that the processing of active integrity constraints can be efficiently parallelized in this disjoint scenario, given suitable assumptions.

5.2 Deductive databases

We now address the case of deductive databases. These consist of two different components: the (extensional) fact database, containing only concrete instances of relations, and the (intensional) rule database, containing Datalog-style rules defining new relations. Every relation must be either intensional or extensional, unlike in e.g. full-fledged logic programming.

One standard way to see the intensional component(s) of deductive databases is as *views* of the original database. The instances of the new relations defined by rules are generated automatically from the data in the database, and these relations can thus be seen as content-free, having a purely presentational nature. For simplicity of presentation, we consider the case where there is one single view.

Definition 18 Let Σ_E and Σ_I be two disjoint first-order signatures. A *deductive database* over Σ_E and Σ_I is a pair $\langle \mathsf{DB}, R\rangle$, where DB is a relational database over Σ_E and R is a set of rules of the form $p \leftarrow q_1, \ldots, q_n$, where p is an atom of Σ_I and $q_1, \ldots, q_n$ are atoms over $\Sigma_E \cup \Sigma_I$.

[9] We implicitly assume that the database is closed under projection, and overload the person relation for the sake of simplicity. This assumption is meaningful from a practical point of view, and has been implemented for databases e.g. in [18].

More precisely, this definition corresponds to the definite deductive databases in [32]; we do not consider the case of indefinite databases in this work.

Definition 19 Let $\langle \mathsf{DB}, R\rangle$ be a deductive database over Σ_E and Σ_I. The MCS induced by $\langle \mathsf{DB}, R\rangle$ is $M = \langle C_E, C_I\rangle$, where $C_E = \mathsf{Ctx}(\mathsf{DB})$ as in Definition 17 and $C_I = \mathsf{Ctx}(R)$ is a similar context where:

- the knowledge base is $\emptyset$;
- for each rule $p \leftarrow q_1, \ldots, q_n$ in R there is a bridge rule $(I : p) \leftarrow (i_1 : q_1), \ldots, (i_n : q_n)$ in $\mathsf{Ctx}(R)$, where $i_k = E$ if q_k is an atom over Σ_E and $i_k = I$ otherwise;
- there are no update operation names, i.e., $\mathsf{OP}^*_I = \emptyset$.

We already presented a deductive database in this manner (Example 7).

The bodies of active integrity constraints over MCSs generated in this way correspond precisely to integrity constraints over deductive databases as defined in [4]. The choice of update operation names captures the typical constraints of deductive databases – that consistency can only be regained by changing extensional predicates – in line with the traditional view-update problem. More modern works [11] restrict the syntax of integrity constraints, allowing them to use only extensional relations; in the induced MCS, this translates to the additional requirement that only relational elements from C_E appear in the body of active integrity constraints.

Example 12 Consider a deductive database for class diagrams, where information about direct subclasses is stored in the extensional database using a relation isa, e.g. $\mathsf{isa}(\mathit{list}, \mathit{collection})$ and $\mathsf{isa}(\mathit{array}, \mathit{list})$. Intensionally, we model the transitive closure of the subclass relation using a view created by the two rules $\mathsf{sub}(A, B) \leftarrow \mathsf{isa}(A, B)$ and $\mathsf{sub}(A, C) \leftarrow \mathsf{isa}(A, B), \mathsf{sub}(B, C)$, thus allowing us to find out that in our example $\mathsf{sub}(\mathit{array}, \mathit{collection})$.

The integrity constraint $\neg\mathsf{sub}(A, A)$ states the acyclicity of the subclass relation. To write it as an AIC, we need to specify which repair actions we can take to resolve the inconsistency. Since sub is a defined predicate, the only possible way to break this inference is by removing facts concerning isa, namely by writing e.g.

$$(C_I : \mathsf{sub}(A, A)) \Longrightarrow (C_E : \mathsf{del}(\mathsf{isa}(A)))$$

with the intended meaning that the management function deletes *all* instances of $\mathsf{isa}(A, X)$ for some X.[10] Observe that this AIC is valid, since a cycle through A must necessarily include an instance $\mathsf{isa}(A, X)$ for some X, and removing all such X is guaranteed to eliminate all such cycles. In this case, all repairs are grounded.

This integrity constraint cannot be expressed in the language of the extensional database – there is no way to define a fixpoint in this language, and the only (incomplete) solution would be to add n integrity constraints disallowing cycles of length up to n. This example thus illustrates our gain of expressive power compared to the approach in [11].

We can also consider databases with several, different views, each view generating a different context. Active integrity constraints over the resulting MCS can then specify relationships between relations in different views.

[10]This extended semantics of the behavior of the management function is again very reasonable in practice; see the discussion in the previous subsection.

The complexity of verifying whether an MCS induced by a deductive database satisfies its active integrity constraints is lower than the general case. In particular, consistency checking is reducible to query answering (all active integrity constraints are satisfied iff there are no answers to the queries expressed in their bodies). If we do not allow negation in the definition of the intensional relations, then there is only one model of the database as before, and consistency checking w.r.t. a fixed set of active integrity constraints is PTIME-complete [46]. In the general case, weak and strong consistency correspond, respectively, to brave and cautious reasoning for Datalog programs under answer set semantics, which are known to be co-NP-complete and NP-complete, respectively. At the algorithmic level, the advantage of working with active integrity constraints is that we avoid dealing with the view update problem to fix inconsistencies, as we simply need to consider the actions in the head of applicable AICs.

5.3 Other models

In [21] we also considered the model of peer-to-peer systems [14] and showed how they could be interpreted as MCSs with integrity constraints (in denial clausal form). However, since these generalize deductive databases, we cannot systematically translate them into the richer formalism we present herein – that would require solving the view-update problem systematically in order to determine which repair actions to include in the heads of AICs, which is beyond the scope of this work.

6 The case of ontologies

This section is devoted to examples illustrating how our framework can be applied to the particular case of integrity constraints over ontologies.

6.1 Ontologies as multi-context systems

Previous work [19, 21] shows how to view an ontology as a context of an MCS. This encoding is significantly different from ones considered previously for different types of databases: since description logics reject the closed-world assumption, thereby contradicting the semantics of negation-by-failure, encoding ontologies as a context in an MCS requires a different approach.

In the present work, we refine the previous construction by representing an ontology as *two* contexts: one for the A-Box, one for the T-Box, connected by bridge rules that port every instance from the former into the latter. (This is reminiscent of how deductive databases are encoded in MCSs, see Section 5.2.) This finer encoding allows us, in particular, to reason about asserted instances (which are given in the A-Box) and those that are derived using the axioms (see Example 15). Previous work [43] has already discussed the importance of being able to write integrity constraints that distinguish asserted and derived information.

We further assume that the A-Box only contains instances of atomic concepts or roles ($C(t)$ or $R(t, t')$). This option does not restrict the expressive power of the ontology, but it helps structure AICs: to include instance axioms about e.g. $C \sqcup D$, one instead defines a new concept $E = C \sqcup D$ in the T-Box and includes instance axioms about E in the A-Box (see also Example 19).

Definition 20 A description logic $\mathcal{L}$ is represented as the relational logic $L_{\mathcal{L}} = \langle \mathsf{KB}_{\mathcal{L}}, \mathsf{BS}_{\mathcal{L}}, \mathsf{ACC}_{\mathcal{L}}, \Sigma_L \rangle$, where:

– $\mathsf{KB}_{\mathcal{L}}$ contains all well-formed knowledge bases of $\mathcal{L}$;
– $\mathsf{BS}_{\mathcal{L}}$ contains all sets of queries in the language of $\mathcal{L}$;
– $\mathsf{ACC}_{\mathcal{L}}(\mathsf{kb})$ is the singleton set containing the set of queries to which kb answers "Yes".
– $\Sigma_{\mathcal{L}}$ is the first-order signature underlying $\mathcal{L}$.

An ontology $\mathcal{O} = \langle T, A \rangle$ based on $\mathcal{L}$ induces the multi-context system $M(\mathcal{O}) = \langle \mathsf{Ctx}(T), \mathsf{Ctx}(A) \rangle$ where $\mathsf{Ctx}(T) = \langle L_{\mathcal{L}}, T, \mathsf{br}_T, \Sigma_0, \mathsf{OP}_T, \mathsf{mng}_T \rangle$ with

– br_T contains all rules of the form $(T : \mathsf{add}(C))(X) \leftarrow (A : C)(X)$ where C is a concept, and $(T : \mathsf{add}(R))(X, Y) \leftarrow (A : R)(X, Y)$ where R is a role;
– Σ_0 is the set of constants in $\Sigma_{\mathcal{L}}$;
– $\mathsf{OP}_T = \{\mathsf{add}\}$, $\mathsf{mng}_T(\mathsf{add}(p), \mathsf{kb})$ adds p to kb, and $\mathsf{OP}^*_T = \emptyset$;

and $\mathsf{Ctx}(A) = \langle L_{\mathcal{L}}, A, \emptyset, \Sigma_0, \mathsf{OP}_A, \mathsf{mng}_A \rangle$ where OP_A and mng_A are the set of allowed (update) operation names and their definition.

The management function only allows adding instances to the T-Box, and this is not an update operation. The particular operations in the A-Box depend on the concrete ontology. This is in line with our motivation that writing AICs requires knowledge of the system's deductive abilities (expressed by the T-Box), which should not change.

As in the database scenario, ontologies viewed as MCSs always have one equilibrium, as long as they are logically consistent. Therefore, the notions of weak and strong satisfaction of active integrity constraints again coincide, and we get the same notion of consistency with respect to a set of integrity constraints as that defined in [40]; however, our syntax is more restricted, as we do not allow general formulas as integrity constraints, but only queries. Our active integrity constraints only apply to named individuals (explicitly mentioned in the ontology's A-Box), which is a desirable consequence that yet again can only be gained from keeping integrity constraints separate from the knowledge base. This is standard practice in ontologies, taken in the works cited throughout this section.

Our scenario is also expressive enough to model the distributed ontology scenario of [29], which defines integrity constraints as logic programming-style rules with empty head whose body can include atoms from different ontologies: we can simply consider the MCS obtained from viewing each ontology as a separate context, and the active integrity constraints (with appropriate actions added to their heads) as ranging over the joint system.

6.2 Evaluation of expressivity

We now evaluate the expressivity of our development by showing how to formalize several types of ICs over ontologies. We follow the classification in Section 4.5 of [28], which describes families of ICs determined by OWL engineers and ontologists as the most interesting, as well as other types of ICs considered in the scientific literature. Several classes of ICs are syntactically similar, so we do not include examples for all categories in [28], but explain in the text how the missing ones can be treated. In addition, we discuss how to deal with missing properties and unnamed individuals in our framework.

Most of our examples are adapted from [28], which frames them in a variant of the Lehigh University Benchmark [38], an ontology designed with the goal of providing a realistic scenario for testing. We follow the description in Section 4.4 of [28]. This ontology considers concepts $\mathsf{student}$, $\mathsf{gradStudent}$, class and email, and roles $\mathsf{hasEmail}$, $\mathsf{enrolled}$ and webEnrolled. Our semantics is: class is a concept including all classes of a common course; enrolled(c, s) holds if student s is enrolled in course s; and webEnrolled holds if the student is furthermore to be contacted only electronically.[11]

The actual contents of the A-Box are immaterial for our presentation, and we restrict ourselves to the fragment of the T-Box containing the following axioms.

$$\begin{array}{llll} \mathsf{gradStudent} \sqsubseteq \mathsf{student} & \exists\mathsf{enrolled.student} & \sqsubseteq \mathsf{class} \\ \mathsf{webEnrolled} \sqsubseteq \mathsf{enrolled} & \exists\mathsf{hasEmail.email} & \sqsubseteq \mathsf{student} \\ & \exists\mathsf{webEnrolled}^R\mathsf{.class} & \sqsubseteq \exists\mathsf{hasEmail} \end{array}$$

6.2.1 *Functional dependencies*

Functional dependencies are one of the most frequently occurring families of ICs: they are requirements that certain relations be functional on one argument. In our example, this applies to hasEmail: two distinct students cannot have the same e-mail.

Since ontologies do not have the Unique Name Assumption, we cannot distinguish individuals by checking name equality (as in databases), but must query the ontology instead. Furthermore, while in the database world such violations can only be repaired by removing one of the offending instances, in ontologies, we can also add the information that two individuals are the same.

Example 13 Suppose that the management function includes operations add and del to add or remove a particular instance from the A-Box, as well as assertEqual, establishing equality of two individuals. Under these assumptions, we can express functionality of e-mail as the following AIC.

$$\begin{aligned} &(A : \mathsf{hasEmail}(X, Z)), (A : \mathsf{hasEmail}(Y, Z)), \mathsf{not}\ (T : (X = Y)) \\ \Longrightarrow\ &(A : \mathsf{del}(\mathsf{hasEmail}(X, Z))) \mid (A : \mathsf{assert}(X = Y)) \end{aligned} \tag{5}$$

Observe that, if T explicitly proves that $X \neq Y$, then only the first action can be used, as asserting equality between X and Y would lead to an inconsistency. However, if this is not the case then the second action is also a repair possibility, and hence this AIC is valid. There are several possibilities for the implementation of assert: it can add the equality $X = Y$ to the A-Box, but it can also syntactically replace every occurrence of one of them for the other.

6.2.2 *Key constraints, uniqueness constraints, functionality constraints*

These types of dependencies from Section 4.5 of [28] are expressed by similar formulas.

6.2.3 *Property domain constraints*

This family of ICs specifies that the domain of a role should be a subset of a particular concept. In case such a constraint is violated, the offending element has to be added as an instance of that concept. The treatment of these ICs is thus very similar to the database case.

[11] This semantics is slightly changed from that of [28], in order to make some aspects of our example more realistic.

Example 14 To model that only students can be enrolled in courses, we write the following AIC.

$$(T : \mathsf{enrolled}(X, Y), \mathsf{not}\ (T : \mathsf{student}(Y)) \Longrightarrow (A : \mathsf{add}(\mathsf{student}(Y))) \tag{6}$$

We could also add the action $(A : \mathsf{del}(\mathsf{enrolled}(X, Y)))$ to the head of this AIC; note that it would only restore consistency in the case where this fact is explicitly stated in the A-Box and not otherwise derivable.

6.2.4 Property range constraints

These constraints, which restrict the range of a role, can be similarly treated.

6.2.5 Specific type constraints

In many applications, it is interesting to minimize redundancy in the A-Box. In particular, in the presence of inclusion axioms, it is often desirable only to include instances pertaining to the most specific type class of each individual. In order to address this issue, we take advantage of the separation between the A-Box and the T-Box in a multi-context system.

Example 15 Since $\mathsf{gradStudent} \sqsubseteq \mathsf{student}$, we guarantee that the A-Box only contains instances of the most specific class a student belongs to by writing:

$$(A : \mathsf{gradStudent}(X)), (A : \mathsf{student}(X)) \Longrightarrow (A : \mathsf{del}(\mathsf{student}(X))) \tag{7}$$

Thus, if the A-Box contains e.g. student(john) and gradStudent(john), then the axiom student(john) must be removed. Observe that the system can still derive student(john), but only in context C_T (using the information in the T-Box). The separation of the A-Box and T-Box in different contexts is essential to express this integrity constraint in our formalism. Constraints that distinguish between assertions explicitly stated in the A-Box and derived ones have been considered e.g. in [43].

6.2.6 Min-cardinality constraints

We now consider a more interesting type of ICs: min-cardinality constraints. Inconsistencies arising from the violation of such constraints are hard to repair automatically, as such a repair requires "guessing" which instances to add. Using AICs and adequate management functions, we can even specify the construction of "default" values that may depend on the actual ontology.

Example 16 We want to express that each class must have a minimum of 10 students. Classes with less enrolled students should be closed, and those students moved to the smallest remaining class using an operation redistribute.

$$(T : (\leq 10.\mathsf{enrolled})(X)) \Longrightarrow (A : \mathsf{redistribute}(\neg\mathsf{class}(X))) \tag{8}$$

This is an example of an AIC whose repair is performed in a context different from the one where the inconsistency is detected: even though all instances of enrolled are in C_A (they come from the A-Box), the inference that there are less than 10 such instances can only be done in C_T.

For this AIC to be valid, redistribute must check whether students are enrolled or webEnrolled and change the appropriate instance in the A-Box. This also uses the knowledge that instances of enrolled cannot be derived in other ways.

6.2.7 Max-cardinality constraints

Dually to the previous ones, max-cardinality constraints express that there may be a maximum number of instances of some concept or role. We can solve inconsistencies detected by these by similar techniques as demonstrated both for functional dependencies and for min-cardinality constraints: add repair actions, unifying some individuals, or calling more sophisticated management functions.

Example 17 Dually to the previous example, we now want to express that each class can have at most 25 students. If a class has more enrolled students, then either some student must be removed from it, or (alternatively) students may be redistributed between two classes.

$$\begin{aligned} &(T : (\geq 25.\mathsf{enrolled})(X)), (T : \mathsf{enrolled}(S, X)) \\ \Longrightarrow\ &(A : \mathsf{del}(\mathsf{enrolled}(S, X))) \mid (A : \mathsf{redistribute}(\neg\mathsf{class}(X))) \end{aligned} \tag{9}$$

Observe that S may be instantiated by any student in the offending class, so that removing *any* excess student is a viable repair option. This is a valid option, because this particular instance of the AIC will no longer be applicable – although, if there still are more than 25 students left in the class, other instances of the same AIC are still able to fire.

6.2.8 Totality constraints

A similar kind of constraints are totality constraints, which require that a role be total on one of its arguments.

Example 18 Suppose we want to ensure that every student is enrolled in at least one course. We can write this by means of the following active integrity constraint

$$\mathsf{not}\ (T : (\mathsf{enrolled}^R)(X)) \Longrightarrow (A : \mathsf{enroll}(\mathsf{student}(X))) \tag{10}$$

This situation is similar to Example 16. The management function enroll has the task of enrolling X in a particular course (using criteria that may depend on the state of the knowledge base, for example, on which courses X has completed previously). Again, the repair is performed in a different context from the one where the body of the AIC is tested.

Validity of this AIC depends again on the semantics of enroll.

6.2.9 Missing property value constraints

Another kind of ICs, also very common in ontologies, is disallowing unnamed individuals for particular properties [43].

Example 19 Our ontology specifies that all students that are webEnrolled in a class must have an e-mail address. However, for the purpose of contacting these individuals, this e-mail address must be explicitly provided. We address this issue with the following AIC.

$$(T : (\exists \mathsf{hasEmail})(X)), \mathsf{not}\ (T : \mathsf{hasEmail}(X, Y)) \\ \Longrightarrow (A : \mathsf{unregister}(\neg\exists \mathsf{webEnrolled}^R(X))) \tag{11}$$

Here, unregister replaces the axiom $\mathsf{webEnrolled}(X)$ with $\mathsf{enrolled}(X)$, as it makes sense to keep the student enrolled in the course. (Recall that webEnrolled is included in enrolled.) Validity of this AIC follows from observing that the only possible ways to derive $\exists\mathsf{hasEmail}(X)$ are either from an explicit assertion $\mathsf{hasEmail}(X, Y)$ or indirectly from $\mathsf{webEnrolled}(Z, X)$.

This example also justifies our requirement that the A-Box can only contain instances of atomic concepts or roles. If the A-Box were allowed to contain e.g. ∃hasEmail(john), then AIC (11) would no longer be valid. By restricting to atomic concepts, the only way to perform a similar change would be by defining a new concept as equivalent to ∃hasEmail – and this information would be present in the T-Box, making it clear that AICs should consider it.

6.2.10 Managing unnamed individuals

Finally, we illustrate how we can write AICs in different ways to control whether they range over all individuals of a certain class, or only over named ones.

Example 20 For ecological reasons, we want all students with an e-mail address to be enrolled in the web version of courses. We can write this as follows.

$$(T : (\mathsf{hasEmail})(Y, Z)), (T : \mathsf{enrolled}(X, Y)), \mathsf{not}\ (T : \mathsf{webEnrolled}(X, Y)) \\ \Longrightarrow (A : \mathsf{webEnroll}(\mathsf{webEnrolled}(X, Y))) \tag{12}$$

Operation webEnroll will replace $\mathsf{enrolled}(X, Y)$ with $\mathsf{webEnrolled}(X, Y)$, dually to unregister in the previous example.

Alternatively, we could consider writing

$$(T : (\exists\mathsf{hasEmail})(Y)), (T : \mathsf{enrolled}(X, Y)), \mathsf{not}\ (T : \mathsf{webEnrolled}(X, Y)) \\ \Longrightarrow (A : \mathsf{webEnroll}(\mathsf{webEnrolled}(X, Y))) \tag{13}$$

In this particular context, this formulation is undesirable, as it will also affect individuals who do not have a known e-mail address. By writing an explicit variable in the first query of the body, as in (12), we guarantee that we only affect those individuals whose e-mail address is known.

Similar considerations about the two possible ways to formulate this type of ICs can be found in [43].

6.3 Discussion

The examples in this section illustrate how our framework is powerful enough to capture the types of integrity constraints over ontologies discriminated in [28], which presents a comprehensive overview of integrity constraints that are desired in practice. We furthermore discussed two examples from [43], of a slightly different nature.

Although this section consists mostly of examples, we believe that they substantiate our claim regarding expressiveness of our framework of active integrity constraints over multi-context systems, when applied to the case of ontologies. Indeed, our underlying ontology is generic and simple enough for these examples to be readily adaptable to other instances of the same types of integrity constraints. Furthermore, by presenting them in a concrete ontology rather than as abstract formulations, we were also able to illustrate the criteria for choosing the repair actions to include in the heads of the active integrity constraints, as well as how to show validity. This is in line with our earlier claim that, even though validity of an AIC is in general an undecidable problem (Theorem 3), in practice the semantics behind the predicates used in concrete AICs makes this task feasible for a human.

7 Discussion

We briefly consider some questions and generalizations that were not addressed in this paper.

7.1 Normalization

The original study of AICs dedicated special attention to *normal* AICs: those with only one action in their head. This is possible because any AIC of the form in (1) can be split into k separate AICs with the same body and only one action in the head; this trivially preserves satisfaction, and it also preserves the set of repairs (except for justified repairs [13], which we do not consider in this work). The drawback is that normalization increases the size of the set of AICs.

In the setting of AICs over MCSs, this transformation is not straightforward. As Example 7 shows, we may need different actions due to there being several possible different origins for the inconsistency; restricting to normalized AICs would thus require strengthening their bodies to guarantee that they only apply in the cases where the particular action in their head suffices. Furthermore, we would need additional AICs for cases where several actions are needed simultaneously, and we cannot express this in our formalism – consider e.g. a variation of Example 7 where p and q originate from different contexts, so that no single invocation of a management function (which is local to a given context) can remove $\mathsf{p}(\mathsf{a})$ and $\mathsf{q}(\mathsf{a})$ simultaneously. Therefore, normalized AICs are not as interesting in the general scenario as in the database case.

7.2 Validity

At the end of Example 7, we pointed out that restoring consistency with respect to an AIC r may require applying several actions in $\mathsf{head}(r)$. This suggests allowing sets of actions (rather than actions) in the heads of AICs. Besides increasing the complexity of our development, it is not clear that this change would bring significant benefits. In terms of computing repairs, we already cover those cases, since we add sets of actions when going from a node to its descendants. Also, it is not clear that there exists a situation when *every* possible inconsistent MCS requires a set of actions to repair.

One could also remove the second condition of validity of an AIC, i.e. allow the actions in the head to be insufficient to restore consistency of some MCSs. This would remove some burden from the programmer who has to specify the AICs, and would not affect the performance of the algorithms in Section 4.4. However, it would contradict the original motivation for AICs [31]: that the actions in the head of a rule should provide the means for restoring consistency. It might then be the case that, in some situations, repairs (that are not grounded) cannot be found because the inconsistency can only be addressed in ways that were not considered by the programmer.

7.3 Variants of AICs

The authors of [31] also considered *conditioned active integrity constraints*, where the actions on the head of AICs are guarded by additional conditions that have to be satisfied. In their setting, conditioned AICs do not add expressive power to the formalism, as they can be split into several unconditioned AICs (with more specific bodies) preserving the notions of consistency and repairs. In our setting, this transformation is not possible, and it would thus be interesting to study conditioned active integrity constraints over multi-context systems. However, we point out that the management function *can* use information about the actual knowledge bases in its implementation, so some conditions can actually be expressed in our setting (see Example 16).

8 Conclusions

In this paper, we proposed a notion of active integrity constraints for multi-context systems, a general framework for combining reasoning systems. We showed that our notion generalizes the well-studied concept of active integrity constraint over databases, and studied its relation to proposed notions of integrity constraints in other formalisms. Satisfaction of active integrity constraints comes in two variants, weak and strong, related to the usual concepts of brave and cautious reasoning.

By showing how to encode the bodies of active integrity constraints within the syntax of MCSs, we obtained decidability and complexity results for the problem of whether a particular MCS weakly or strongly satisfies a set of active integrity constraints, and of deciding whether it can be repaired (in the negative case). We argued however that keeping active integrity constraints as an added layer on top of an MCS allows for a clean separation of intrinsic logical inconsistency from inconsistencies that may arise e.g. from improper changes to an individual context, which we want to detect and fix, rather than propagate to other contexts – in line with several authors who have argued for these concerns. Our examples show that we indeed capture the usual behavior of integrity constraints in several existing formalisms.

We also defined a notion of grounded repair, the type of repair in the database setting that appears most suitable for generalization to other frameworks. We showed that grounded repairs for inconsistent MCSs can be computed automatically. We presented simple criteria for validity of AICs and illustrated by means of examples that we can write valid AICs systematically for several interesting scenarios. This shows that the undecidability of validity in the general case does not limit their practical application in many useful reasoning frameworks. We evaluated our formalism for the ontology scenario, and showed its wide applicability.

References

1. Abiteboul, S.: Updates, a new frontier. In: Gyssens, M., Paredaens, J., van Gucht, D. (eds.) ICDT, volume 326 of LNCS, pp. 1–18. Springer (1988)
2. Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases. Addison Wesley, Boston (1995)

3. Arenas, M., Bertossi, L.E., Chomicki, J.: Consistent query answers in inconsistent databases. In: Vianu, V., Papadimitriou, C.H. (eds.) PODS, pp. 68–79. ACM Press (1999)
4. Asirelli, P., de Santis, M., Martelli, M.: Integrity constraints for logic databases. J. Log Program. **2**(3), 221–232 (1985)
5. Bogaerts, B., Cruz-Filipe, L.: Semantics for active integrity constraints using approximation fixpoint theory. In: Sierra, C. (ed.) IJCAI, pp. 866–872 (2017). ijcai.org
6. Bogaerts, B., Vennekens, J., Denecker, M.: Grounded fixpoints and their applications in knowledge representation. Artif. Intell. **224**, 51–71 (2015)
7. Brewka, G., Eiter, T.: Equilibria in heterogeneous nonmonotonic multi-context systems. In: AAAI, pp. 385–390. AAAI Press (2007)
8. Brewka, G., Eiter, T., Fink, M.: Nonmonotonic multi-context systems: A flexible approach for integrating heterogeneous knowledge sources. In: Balduccini, M., Cao Son, T. (eds.) Logic Programming, Knowledge Representation, and Nonmonotonic Reasoning - Essays Dedicated to Michael Gelfond on the Occasion of His 65th Birthday, volume 6565 of LNCS, pp. 233–258. Springer (2011)
9. Brewka, G., Eiter, T., Fink, M., Weinzierl, A.: Managed multi-context systems. In: Walsh, T. (ed.) IJCAI, pp. 786–791 (2011). IJCAI/AAAI
10. Caroprese, L., Greco, S., Sirangelo, C., Zumpano, E.: Declarative semantics of production rules for integrity maintenance. In: Etalle, S., Truszczynski, M. (eds.) ICLP, volume 4079 of LNCS, pp. 26–40. Springer (2006)
11. Caroprese, L., Trubitsyna, I., Truszczynski, M., Zumpano, E.: The view-update problem for indefinite databases. In: Fariñas del Cerro, L., Herzig, A., Mengin, J. (eds.) JELIA, volume 7519 of LNCS, pp. 134–146. Springer (2012)
12. Caroprese, L., Truszczynski, M.: Declarative semantics for active integrity constraints. In: de la Banda, M.G., Pontelli, E. (eds.) ICLP, volume 5366 of LNCS, pp. 269–283. Springer (2008)
13. Caroprese, L., Truszczyński, M.: Active integrity constraints and revision programming. Theory Pract. Log. Program. **11**(6), 905–952 (2011)
14. Caroprese, L., Zumpano, E.: Consistent data integration in P2P deductive databases. In: Prade, H., Subrahmanian, V.S. (eds.) SUM, volume 4772 of LNCS, pp. 230–243. Springer (2007)
15. Cruz-Filipe, L., Beierle, C., Meghini, C.: Optimizing computation of repairs from active integrity constraints. In: FoIKS, volume 8367 of LNCS, pp. 361–380. Springer (2014)
16. Cruz-Filipe, L.: Grounded fixpoints and active integrity constraints. In: Carro, M., King, A., De Vos, M., Saeedloei, N. (eds.) ICLP'16, volume 52 of OASIcs, pp. 11.1–11.14. Schloss Dagstuhl (2016)
17. Cruz-Filipe, L., Engrácia, P., Gaspar, G., Nunes, I.: Computing repairs from active integrity constraints. In: Wang, H., Banach, R. (eds.) TASE, pp. 183–190. IEEE (2013)
18. Cruz-Filipe, L., Franz, M., Hakhverdyan, A., Ludovico, M., Nunes, I., Schneider-Kamp, P.: repAIrC: A tool for ensuring data consistency by means of active integrity constraints. In: Fred, A.L.N., Dietz, J.L.G., Aveiro, D., Liu, K., Filipe, J. (eds.) KMIS, pp. 17–26. SciTePress (2015)
19. Cruz-Filipe, L., Gaspar, G., Nunes, I.: Information flow within relational multi-context systems. In: Janowicz, K., Schlobach, S., Lambrix, P., Hyvönen, E. (eds.) EKAW, volume 8876 of LNAI, pp. 97–108. Springer (2014)
20. Cruz-Filipe, L., Gaspar, G., Nunes, I., Schneider-Kamp, P.: Active integrity constraints for multi-context systems. In: Blomqvist, E., Vitali, F., Ciancarini, P., Poggi, F. (eds.) EKAW 2016, volume 10024 of LNAI. Springer (2016). accepted for publication
21. Cruz-Filipe, L., Nunes, I., Schneider-Kamp, P.: Integrity constraints for general-purpose knowledge bases. In: Gyssens, M., Simari, G. (eds.) FoIKS, volume 9616 of LNCS, pp. 235–254. Springer (2016)
22. Dao-Tran, M., Eiter, T., Fink, M., Krennwallner, T.: Distributed nonmonotonic multi-context systems. In: Lin, F., Sattler, U., Truszczynski, M. (eds.) KR. AAAI Press (2010)
23. Dao-Tran, M., Eiter, T., Fink, M., Krennwallner, T.: Dynamic distributed nonmonotonic multi-context systems. In: Brewka, G., Marek, V., Truszczynski, M. (eds.) Nonmonotonic Reasoning, Essays Celebrating its 30th Anniversary, volume 31 of Studies in Logic. College Publications (2011)
24. Eiter, T., Fink, M., Ianni, G., Schüller, P.: The IMPL policy language for managing inconsistency in multi-context systems. In: Tompits, H., Abreu, S., Oetsch, J., Pührer, J., Seipel, D., Umeda, M., Wolf, A. (eds.) INAP/WLP, volume 7773 of LNCS, pp. 3–26. Springer (2011)
25. Eiter, T., Fink, M., Ianni, G., Schüller, P.: Towards a policy language for managing inconsistency in multi-context systems. In: Mileo, A., Fink, M. (eds.) Workshop on Logic-based Interpretation of Context: Modelling and Applications, pp. 23–35 (2011)
26. Eiter, T., Fink, M., Schüller, P.: Finding explanations of inconsistency in multi-context systems. Artif Intell. **216**, 233–274 (2014)
27. Eiter, T., Gottlob, G.: On the complexity of propositional knowledge base revision, updates, and counterfactuals. Artif. Intell. **57**(2-3), 227–270 (1992)

28. Fang, M.: Maintaining integrity constraints in semantic web. Ph.D. thesis Georgia State University (2013)
29. Fang, M., Li, W., Sunderraman, R.: Maintaining integrity constraints among distributed ontologies. In: CISIS, pp. 184–191. IEEE (2011)
30. Fink, M., Ghionna, L., Weinzierl, A.: Relational information exchange and aggregation in multi-context systems. In: Delgrande, J.P., Faber, W. (eds.) LPNMR, volume 6645 of *LNCS*, pp. 120–133. Springer (2011)
31. Flesca, S., Greco, S., Zumpano, E.: Active integrity constraints. In: Moggi, E., Scott Warren, D. (eds.) PPDP, pp. 98–107. ACM (2004)
32. Gallaire, H., Minker, J., Nicolas, J.-M.: Logic and databases: A deductive approach. ACM Comput. Surv. **16**(2), 153–185 (1984)
33. Ghidini, C., Giunchiglia, F.: Local models semantics, or contextual reasoning=locality+compatibility. Artif. Intell. **127**(2), 221–259 (2001)
34. Giunchiglia, F., Serafini, L.: Multilanguage hierarchical logics or: How we can do without modal logics. Artif. Intell. **65**(1), 29–70 (1994)
35. Gonçalves, R., Knorr, M., Leite, J.: Evolving multi-context systems. In: Schaub, T., Friedrich, G., O'Sullivan, B. (eds.) ECAI, volume 263 of Frontiers in Artificial Intelligence and Applications, pp. 375–380. IOS Press (2014)
36. Gonçalves, R., Knorr, M., Leite, J.: Evolving multi-context systems. In: Schaub, T., Friedrich, G., O'Sullivan, B. (eds.) PAIS, volume 263 of Frontiers in Artificial Intelligence and Applications, pp. 375–380. IOS Press (2014)
37. Guessoum, A.: Abductive knowledge base updates for contextual reasoning. J. Intell. Inf. Syst. **11**(1), 41–67 (1998)
38. Guo, Y., Pan, Z., Heflin, J.: LUBM: A benchmark for OWL knowledge base systems. J. Web Sem. **3**(2–3), 158–182 (2005)
39. McCarthy, J.: Notes on formalizing context. In: Bajcsy, R. (ed.) IJCAI, pp. 555–562. Morgan Kaufmann (1993)
40. Motik, B., Horrocks, I., Sattler, U.: Bridging the gap between OWL and relational databases, vol. 7 (2011)
41. Motik, B., Rosati, R.: Reconciling description logics and rules. J. ACM, **57**, Article Nr 30 (2010)
42. Ouyang, D., Cui, X., Ye, Y.: Integrity constraints in OWL ontologies based on grounded circumscription. Front Comput Sci **7**(6), 812–821 (2013)
43. Patel-Schneider, P.F., Franconi, E.: Ontology constraints in incomplete and complete data. In: Cudré-Mauroux, P., Heflin, J., Sirin, E., Tudorache, T., Euzenat, J., Hauswirth, M., Parreira, J.X., Hendler, J., Schreiber, G., Bernstein, A., Blomqvist, E. (eds.) ISWC, volume 7649 of LNCS, pp. 444–459. Springer (2012)
44. Przymusinski, T.C., Turner, H.: Update by means of inference rules. J Log. Program. **30**(2), 125–143 (1997)
45. Pührer, J., Heymans, S., Eiter, T.: Dealing with inconsistency when combining ontologies and rules using dl-programs. In: Aroyo, L., Antoniou, G., Hyvönen, E., ten Teije, A., Stuckenschmidt, H., Cabral, L., Tudorache, T. (eds.) ESWC(1), volume 6088 of LNCS, pp. 183–197. Springer (2010)
46. Schlipf, J.S.: Complexity and undecidability results for logic programming. Ann. Math. Artif. Intell. **15**(3–4), 257–288 (1995)
47. Staworko, S., Chomicki, J.: Consistent query answers in the presence of universal constraints. Inf. Syst. **35**(1), 1–22 (2010)
48. Tao, J., Sirin, E., Bao, J., McGuinness, D.L. In: Fox, M., Poole, D. (eds.) AAAI. AAAI Press (2010)
49. Tasharrofi, S., Ternovska, E.: Generalized multi-context systems. In: Baral, C., de Giacomo, G., Eiter, T. (eds.) KR. AAAI Press (2014)
50. Teniente, E., Olivé, A.: Updating knowledge bases while maintaining their consistency. VLDB J. **4**(2), 193–241 (1995)
51. Thalheim, B.: Dependencies in Relational Databases. Teubner-Texte zur Mathematik, B.G Teubner (1991)
52. Ullman, J.D.: Principles of Database and Knowledge-Base Systems, vol. I. Computer Science Press, Rockville (1988)
53. Weinzierl, A.: Advancing multi-context systems by inconsistency management. In: Bragaglia, S., Damásio, C., Montali, M., Preece, A.D., Petrie, C.J., Proctor, M., Straccia, U. (eds.) RuleML2011@BRF Challenge, volume 799 of CEUR Workshop Proceedings. CEUR-WS.org (2011)
54. Winslett, M.: Updating Logical Databases. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, Cambridge (1990)