Verification of Generalized Inconsistency-Aware Knowledge and Action Bases

Diego Calvanese, Marco Montali, Ario Santoso
KRDB Research Centre for Knowledge and Data
Free University of Bozen-Bolzano
lastname@inf.unibz.it

Abstract

Knowledge and Action Bases (KABs) have been put forward as a semantically rich representation of a domain, using a DL KB to account for its static aspects, and actions to evolve its extensional part over time, possibly introducing new objects. Recently, KABs have been extended to manage inconsistency, with ad-hoc verification techniques geared towards specific semantics. This work provides a twofold contribution along this line of research. On the one hand, we enrich KABs with a high-level, compact action language inspired by Golog, obtaining so called Golog-KABs (GKABs). On the other hand, we introduce a parametric execution semantics for GKABs, so as to elegantly accomodate a plethora of inconsistency-aware semantics based on the notion of repair. We then provide several reductions for the verification of sophisticated first-order temporal properties over inconsistency-aware GKABs, and show that it can be addressed using known techniques, developed for standard KABs.

1 Introduction

The combination of static and dynamic aspects in modeling complex organizational domains is a challenging task that has received increased attention, and has led to the study of settings combining formalisms from knowledge representation, database theory, and process management [Hull, 2008; Vianu, 2009; Calvanese *et al.*, 2013a]. Specifically, Knowledge and Action Bases (KABs) [Bagheri Hariri *et al.*, 2013b] have been put forward to provide a semantically rich representation of a domain. In KABs, static aspects are modeled using a knowledge base (KB) expressed in the lightweight Description Logic (DL) [Baader *et al.*, 2003] *DL-Lite*$_{\mathcal{A}}$ [Calvanese *et al.*, 2007b; 2009], while actions are used to evolve its extensional part over time, possibly introducing fresh individuals from the external environment. An important aspect that has received little attention so far in such systems is the management of inconsistency with respect to domain knowledge that may arise when the extensional information is evolved over time. In fact, inconsistency is typically handled naively by just rejecting updates in actions when they would lead to inconsistency. This shortcoming is not only present in KABs, but virtually in all related approaches in the literature, e.g., [Deutsch *et al.*, 2009; Belardinelli *et al.*, 2012; Bagheri Hariri *et al.*, 2013a].

To overcome this limitation, KABs have been extended lately with mechanisms to handle inconsistency [Calvanese *et al.*, 2013b]. However, this has been done by defining adhoc execution semantics and corresponding ad-hoc verification techniques geared towards specific semantics for inconsistency management. Furthermore, it has been left open whether introducing inconsistency management in the rich setting of KABs, effectively leads to systems with a different level of expressive power. In this paper, we attack these issues by: *(i)* Proposing (standard) GKABs, which enrich KABs with a compact action language inspired by Golog [Levesque *et al.*, 1997] that can be conveniently used to specify processes at a high-level of abstraction. As in KABs, standard GKABs still manage inconsistency naively. *(ii)* Defining a parametric execution semantic for GKABs that is able to elegantly accomodate a plethora of inconsistency-aware semantics based on the well-known notion of repair [Eiter and Gottlob, 1992; Bertossi, 2006; Lembo *et al.*, 2010; Calvanese *et al.*, 2010]. *(iii)* Providing several reductions showing that verification of sophisticated first-order temporal properties over inconsistency-aware GKABs can be recast as a corresponding verification problem over standard GKABs. *(iv)* Showing that verification of standard and inconsistency-aware GKABs can be addressed using known techniques, developed for standard KABs. Full proofs can be found in an extended technical report [Calvanese *et al.*, 2015].

2 Preliminaries

We start by introducing the necessary technical preliminaries.

2.1 *DL-Lite*$_{\mathcal{A}}$

We fix a countably infinite set Δ of *individuals*, acting as standard names. To model KBs, we use the lightweight logic *DL-Lite*$_{\mathcal{A}}$ [Calvanese *et al.*, 2007b; 2009], whose *concepts* and *roles* are built according to $B ::= N \mid \exists R$ and $R ::= P \mid P^-$, where N is a *concept name*, B a *basic concept*, P a *role name*, P^- an *inverse role*, and R a *basic role*.

A *DL-Lite*$_{\mathcal{A}}$ *KB* is a pair $\langle T, A\rangle$, where: *(i)* A is an Abox, i.e., a finite set of *ABox assertions* (or *facts*) of the form $N(c_1)$ or $P(c_1, c_2)$, where c_1, c_2 are individuals. *(ii)* $T = T_p \uplus T_n \uplus T_f$ is a *TBox*, i.e., a finite set constituted by a subset T_p of *positive inclusion assertions* of the form $B_1 \sqsubseteq B_2$ and

$R_1 \sqsubseteq R_2$, a subset T_n of *negative inclusion assertions* of the form $B_1 \sqsubseteq \neg B_2$ and $R_1 \sqsubseteq \neg R_2$, and a subset T_f of *functionality assertions* of the form $(\mathsf{funct}\ R)$. We denote by $\textsc{adom}(A)$ the set of individuals explicitly present in A.

We rely on the standard semantics of DLs based on FOL interpretations $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, where $c^{\mathcal{I}} \in \Delta^{\mathcal{I}}$, $N^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$, and $P^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$. The semantics of the *DL-Lite*$_{\mathcal{A}}$ constructs and of TBox and ABox assertions, and the notions of *satisfaction* and of *model* are as usual (see, e.g., [Calvanese *et al.*, 2007b]). We say that A is T*-consistent* if $\langle T, A\rangle$ is satisfiable, i.e., admits at least one model. We also assume that all concepts and roles in T are satisfiable, i.e., for every concept N in T, there exists at least one model $\mathcal{I}$ of T such that $N^{\mathcal{I}}$ is non-empty, and similarly for roles.

Queries. We use queries to access KBs and extract individuals of interest. A *union of conjunctive queries (UCQ)* q over a KB $\langle T, A\rangle$ is a FOL formula of the form $\bigvee_{1\leq i\leq n} \exists \vec{y_i}.conj_i(\vec{x}, \vec{y_i})$, where each $conj_i(\vec{x}, \vec{y_i})$ is a conjunction of atoms, whose predicates are either concept/role names of T, or equality assertions involving variables $\vec{x}$ and $\vec{y_i}$, and/or individuals.

The *(certain) answers* of q over $\langle T, A\rangle$ are defined as the set $ans(q, T, A)$ of substitutions σ of the free variables in q with invidisuals in $\textsc{adom}(A)$, such that $q\sigma$ evaluates to true in every model of $\langle T, A\rangle$. If q has no free variables, then it is called *boolean* and its certain answers are either the empty substitution (corresponding to true), or the empty set (corresponding to false). We also consider the extension of UCQs named *EQL-Lite*(UCQ) [Calvanese *et al.*, 2007a] (briefly, ECQs), that is, the FOL query language whose atoms are UCQs evaluated according to the certain answer semantics above. Formally, an *ECQ* over a TBox T is a (possibly open) formula of the form[1]:

$$Q ::= [q] \mid \neg Q \mid Q_1 \wedge Q_2 \mid \exists x.Q$$

where q is a UCQ, and $[q]$ denotes the fact that q is evaluated under the (minimal) knowledge operator [Calvanese *et al.*, 2007a].[2] Intuitively, the *certain answers* $\textsc{ans}(Q, T, A)$ *of an ECQ Q over $\langle T, A\rangle$* are obtained by computing the certain answers of the UCQs embedded in Q, then composing such answers through the FO constructs in Q (interpreting existential variables as ranging over $\textsc{adom}(A)$).

2.2 Inconsistency Management in DL KBs

Retrieving certain answers from a KB makes sense only if the KB is consistent: if it is not, then each query returns all possible tuples of individuals of the ABox. In a dynamic setting where the ABox evolves over time, consistency is a too strong requirement, and in fact a number of approaches have been proposed to handle the instance-level evolution of KBs, managing inconsistency when it arises. Such approaches typically follow one of the two following two strategies: *(i)* inconsistencies are kept in the KBs, but the semantics of query answering is refined to take this into account (*consistent query answering* [Bertossi, 2006]); *(ii)* the extensional part of an inconsistent KB is (minimally) *repaired* so as to remove inconsistencies, and certain answers are then applied over the curated KB. In this paper, we follow the approach in [Calvanese *et al.*, 2013b], and consequently focus on repair-based approaches. However, our results seamlessly carry over the setting of consistent query answering. We then recall the basic notions related to inconsistency management via repair, distinguishing approaches that repair an ABox and those that repair an update.

ABox repairs. Starting from the seminal work in [Eiter and Gottlob, 1992], in [Lembo *et al.*, 2010] two approaches for repairing KBs are proposed: *ABox repair* (AR) and *intersection ABox repair* (IAR). In [Calvanese *et al.*, 2013b], these approaches are used to handle inconsistency in KABs, and are respectively called *bold-repair* (*b-repair*) and *certain-repair* (*c-repair*). Formally, a *b-repair of an ABox A w.r.t. a TBox T* is a *maximal T-consistent subset A' of A*, i.e.: *(i)* $A' \subseteq A$, *(ii)* A' is T-consistent, and *(iii)* there does not exists A'' such that $A' \subset A'' \subseteq A$ and A'' is T-consistent. We denote by $\textsc{b-rep}(T, A)$ the set of all b-repairs of $\langle T, A\rangle$. The *c-repair of an ABox A w.r.t. a TBox T* is the (unique) set $\textsc{c-rep}(T, A) = \cap_{A_i \in \textsc{b-rep}(T,A)} A_i$ of ABox assertions, obtained by intersecting all b-repairs.

Inconsistency in KB evolution. In a setting where the KB is subject to instance-level evolution, b- and c-repairs are computed agnostically from the updates: each update is committed, and only secondly the obtained ABox is repaired if inconsistent. In [Calvanese *et al.*, 2010], a so-called *bold semantics* is proposed to apply the notion of repair to the update itself. Specifically, the bold semantics is defined over a consistent KB $\langle T, A\rangle$ and an instance-level update that comprises two ABoxes F^- and F^+, respectively containing those assertions that have to be deleted from and then added to A. It is assumed that F^+ is consistent with T, and that new assertions have "priority": if an inconsistency arises, newly introduced facts are preferred to those already present in A. Formally, the *evolution of an ABox A w.r.t. a TBox T by F^+ and F^-*, written $\textsc{evol}(T, A, F^+, F^-)$, is an ABox $A_e = F^+ \cup A'$, where *(i)* $A' \subseteq (A \setminus F^-)$, *(ii)* $F^+ \cup A'$ is T-consistent, and *(iii)* there does not exists A'' such that $A' \subset A'' \subseteq (A \setminus F^-)$ and $F^+ \cup A''$ is T-consistent.

2.3 Knowledge and Action Bases

Knowledge and Action Bases (KABs) [Bagheri Hariri *et al.*, 2013b] have been proposed as a unified framework to simultaneously account for the static and dynamic aspects of an application domain. This is done by combining a semantically-rich representation of the domain (via a DL KB), with a process that evolves the extensional part of such a KB, possibly introducing, through service calls, new individuals from the external world. We briefly recall the main aspects of KABs, by combining the framework in [Bagheri Hariri *et al.*, 2013b] with the action specification formalism in [Montali *et al.*, 2014].

We consider a finite set of distinguished individuals $\Delta_0 \subset \Delta$, and a finite set $\mathcal{F}$ of *functions* representing *service calls*, which abstractly account for the injection of fresh individuals from Δ into the system. A *KAB* is a tuple $\mathcal{K} = \langle T, A_0, \Gamma, \Pi\rangle$ where: *(i)* T is a *DL-Lite*$_{\mathcal{A}}$ TBox that captures the intensional aspects of the domain of interest; *(ii)* A_0 is the initial *DL-Lite*$_{\mathcal{A}}$ ABox, describing the initial configuration of data; *(iii)* Γ is a finite set of parametric actions that evolve the ABox; *(iv)* Π is a finite set of condition-action rules forming a process,

[1] In this work we only consider domain independent ECQs.

[2] We omit the square brackets for single-atom UCQs.

which describes when actions can be executed, and with which parameters. We assume that $\text{ADOM}(A_0) \subseteq \Delta_0$.

An *action* $\alpha \in \Gamma$ has the form $\alpha(\vec{p}) : \{e_1, \dots, e_m\}$, where *(i)* α is the *action name*, *(ii)* $\vec{p}$ are the *input parameters*, and *(iii)* $\{e_1, \dots, e_m\}$ is the set of *effects*. Each effect has the form $Q(\vec{x}) \rightsquigarrow$ **add** F^+, **del** F^-, where: *(i)* $Q(\vec{x})$ is an ECQ, possibly mentioning individuals in Δ_0 and action parameters $\vec{p}$. *(ii)* F^+ is a set of atoms (over the alphabet of T) to be *added* to the ABox, each having as terms: individuals in Δ_0, action parameters $\vec{p}$, free variables $\vec{x}$ of Q, and service calls, represented as Skolem terms formed by applying a function $f \in \mathcal{F}$ to one of the previous kinds of terms. *(iii)* F^- is a set of atoms (over the alphabet of T) to be *deleted* from the ABox, each having as terms: individuals in Δ_0, input parameters $\vec{p}$, and free variables of Q. We denote by $\text{EFF}(\alpha)$ the set of effects in α. Intuitively, action α is executed by grounding its parameters, and then applying its effects in parallel. Each effect instantiates the atoms mentioned in its head with all answers of Q, then issues the corresponding service calls possibly contained in F^+, and substitutes them with the obtained results (which are individuals from Δ). The update induced by α is produced by adding and removing the ground atoms so-obtained to/from the current ABox, giving higher priority to additions.

The *process* Π comprises a finite set of *condition-action rules* of the form $Q(\vec{x}) \mapsto \alpha(\vec{x})$, where: $\alpha \in \Gamma$ is an action, and $Q(\vec{x})$ is an ECQ over T, whose terms are free variables $\vec{x}$, quantified variables, and individuals in Δ_0. Each condition-action rule determines the instantiations of parameters with which to execute the action in its head over the current ABox.

The execution semantics of a KAB is given in terms of a possibly infinite-state *transition system*, whose construction depends on the adopted semantics of inconsistency [Calvanese *et al.*, 2013b]. In general, the transition systems we consider are of the form $\langle \Delta, T, \Sigma, s_0, abox, \Rightarrow \rangle$, where: *(i)* T is a *DL-Lite*$_\mathcal{A}$ TBox; *(ii)* Σ is a (possibly infinite) set of states; *(iii)* $s_0 \in \Sigma$ is the initial state; *(iv)* $abox$ is a function that, given a state $s \in \Sigma$, returns an ABox associated to s; *(v)* $\Rightarrow \subseteq \Sigma \times \Sigma$ is a transition relation between pairs of states.

Following the terminology in [Calvanese *et al.*, 2013b], we call S-KAB a KAB under the standard execution semantics of KABs, where inconsistency is naively managed by simply rejecting those updates that lead to an inconsistent state. The transition system $\Upsilon_\mathcal{K}^S$ accounting for the standard execution semantics of KAB $\mathcal{K}$ is then constructed by starting from the initial ABox, applying the executable actions in all possible ways, and generating the (consistent) successor states by applying the corresponding updates, then iterating through this procedure. As for the semantics of service calls, in line with [Calvanese *et al.*, 2013b] we adopt the *deterministic semantics*, i.e., services return always the same result when called with the same inputs. Nondeterministic services can be seamlessly added without affecting our technical results.

To ensure that services behave deterministically, the states of the transition system are also equipped with a service call map that stores the service calls issued so far, and their corresponding results. Technically, a *service call map* is a partial function $m : \mathbb{SC} \rightarrow \Delta$, where $\mathbb{SC} = \{sc(v_1, \dots, v_n) \mid sc/n \in \mathcal{F} \text{ and } \{v_1, \dots, v_n\} \subseteq \Delta\}$ is the set of (Skolem terms representing) service calls.

2.4 Verification Formalism

To specify sophisticated temporal properties to be verified over KABs, taking into account the system dynamics as well as the evolution of data over time, we rely on the $\mu\mathcal{L}_A^{\text{EQL}}$ logic, the FO variant of the μ-calculus defined in [Bagheri Hariri *et al.*, 2013b]. $\mu\mathcal{L}_A^{\text{EQL}}$ combines the standard temporal operators of the μ-calculus with EQL queries over the states. FO quantification is interpreted with an active domain semantics, i.e., it ranges over those individuals that are explicitly present in the current ABox, and fully interacts with temporal modalities, i.e., it applies *across* states. The $\mu\mathcal{L}_A^{\text{EQL}}$ syntax is:

$$\Phi \ := \ Q \mid \neg\Phi \mid \Phi_1 \wedge \Phi_2 \mid \exists x.\Phi \mid \langle - \rangle\Phi \mid Z \mid \mu Z.\Phi$$

where Q is a possibly open EQL query that can make use of the distinguished individuals in Δ_0, Z is a second-order variable denoting a predicate (of arity 0), $\langle - \rangle\Phi$ indicates the existence of a next state where Φ holds, and μ is the least fixpoint operator, parametrized with the free variables of its bounding formula. We make use of the following standard abbreviations: $\forall x.\Phi = \neg(\exists x.\neg\Phi)$, $\Phi_1 \vee \Phi_2 = \neg(\neg\Phi_1 \wedge \neg\Phi_2)$, $[-]\Phi = \neg\langle - \rangle\neg\Phi$, and $\nu Z.\Phi = \neg\mu Z.\neg\Phi[Z/\neg Z]$.

For the semantics of $\mu\mathcal{L}_A^{\text{EQL}}$, which is given over transition systems of the form specified in Section 2.3, we refer to [Bagheri Hariri *et al.*, 2013b]. Given a transition system Υ and a closed $\mu\mathcal{L}_A^{\text{EQL}}$ formula Φ, we call *model checking* verifying whether Φ holds in the initial state of Υ, written $\Upsilon \models \Phi$.

3 Golog-KABs and Inconsistency

In this section, we leverage on the KAB framework (cf. Section 2.3) and provide a twofold contribution. On the one hand, we enrich KABs with a high-level action language inspired by Golog [Levesque *et al.*, 1997]. This allows modelers to represents processes much more compactly, and will be instrumental for the reductions discussed in Sections 4 and 5. On the other hand, we introduce a parametric execution semantics, which elegantly accomodates a plethora of inconsistency-aware semantics based on the notion of repair.

A *Golog-KAB (GKAB)* is a tuple $\mathcal{G} = \langle T, A_0, \Gamma, \delta \rangle$, where T, A_0, and Γ are as in standard KABs, and δ is the Golog program characterizing the evolution of the GKAB over time, using the atomic actions in Γ. For simplicity, we only consider a core fragment[3] of Golog based on the action language in [Calvanese *et al.*, 2011], and define a *Golog program* as:

$$\begin{array}{c}\delta ::= \varepsilon \mid \textbf{pick } Q(\vec{p}).\alpha(\vec{p}) \mid \delta_1|\delta_2 \mid \delta_1;\delta_2 \mid \\ \textbf{if } \varphi \textbf{ then } \delta_1 \textbf{ else } \delta_2 \mid \textbf{while } \varphi \textbf{ do } \delta\end{array}$$

where: (1) ε is the *empty program*; (2) **pick** $Q(\vec{p}).\alpha(\vec{p})$ is an *atomic action invocation* guarded by an ECQ Q, such that $\alpha \in \Gamma$ is applied by non-deterministically substituting its parameters $\vec{p}$ with an answer of Q; (3) $\delta_1|\delta_2$ is a *non-deterministic choice* between programs; (4) $\delta_1;\delta_2$ is *sequencing*; (5) **if** φ **then** δ_1 **else** δ_2 and **while** φ **do** δ are *conditional* and *loop* constructs, using a boolean ECQ φ as condition.

[3]The other Golog constructs, including non-deterministic iteration and unrestricted pick, can be simulated with the constructs considered here.

Execution Semantics. As for normal KABs, the execution semantics of a GKAB $\mathcal{G}$ is given in terms of a possibly infinite-state transition system $\Upsilon_{\mathcal{G}}$, whose states are labelled with ABoxes. The states we consider, are tuples of the form $\langle A, m, \delta\rangle$, where A is an ABox, m a service call map, and δ a program. Together, A and m constitute the *data-state*, which captures the result of the actions executed so far, together with the answers returned by service calls issued in the past. Instead, δ is the *process-state*, which represents the program that still needs to be executed from the current data-state.

We adopt the functional approach by Levesque [1984] in defining the semantics of action execution over $\mathcal{G}$, i.e., we assume $\mathcal{G}$ provides two operations: *(i)* ASK, to answer queries over the current KB; *(ii)* TELL, to update the KB through an atomic action. Since we adopt repairs to handle inconsistency, the ASK operator corresponds to certain answers computation.

We proceed now to formally define TELL. Given an action invocation **pick** $Q(\vec{p}).\alpha(\vec{p})$ and an ABox A, we say that substitution σ of parameters $\vec{p}$ with individuals in Δ is *legal for α in A* if $\textsc{ans}(Q\sigma, T, A)$ is true. If so, we also say that $\alpha\sigma$ *is executable in* A, and we define the sets of atoms to be added and deleted by **pick** $Q(\vec{p}).\alpha(\vec{p})$ with σ in A as follows:

$$\textsc{add}^{A}_{\alpha\sigma} = \bigcup_{(Q \rightsquigarrow \textbf{add } F^+, \textbf{del } F^-) \text{ in } \textsc{Eff}(\alpha)} \bigcup_{\rho \in \textsc{ans}(Q\sigma,T,A)} F^+\sigma\rho$$
$$\textsc{del}^{A}_{\alpha\sigma} = \bigcup_{(Q \rightsquigarrow \textbf{add } F^+, \textbf{del } F^-) \text{ in } \textsc{Eff}(\alpha)} \bigcup_{\rho \in \textsc{ans}(Q\sigma,T,A)} F^-\sigma\rho$$

In general, $\textsc{add}^{A}_{\alpha\sigma}$ is not a proper set of facts, because it could contain (ground) service calls, to be substituted with corresponding results. We denote by $\textsc{calls}(\textsc{add}^{A}_{\alpha\sigma})$ the set of ground service calls in $\textsc{add}^{A}_{\alpha\sigma}$, and by $\textsc{eval}(\textsc{add}^{A}_{\alpha\sigma})$ the set of call substitutions with individuals in Δ, i.e., the set

$$\{\theta \mid \theta \text{ is a total function, } \theta : \textsc{calls}(\textsc{add}^{A}_{\alpha\sigma}) \to \Delta\}$$

Given two ABoxes A and A' where A is assumed to be T-consistent, and two sets F^+ and F^- of facts, we introduce a so-called *filter relation* to indicate that A' is obtained from A by adding the F^+ facts and removing the F^- ones. To account for inconsistencies, the filter could drop some additional facts when producing A'. Hence, a filter consists of tuples of the form $\langle A, F^+, F^-, A'\rangle$ satisfying $\emptyset \subseteq A' \subseteq ((A \setminus F^-) \cup F^+)$. In this light, filter relations provide an abstract mechanism to accommodate several inconsistency management approaches.

We now concretize TELL as follows. Given a GKAB $\mathcal{G}$ and a filter f, we define $\textsc{tell}_f$ as the following relation over pairs of data-states in $\Upsilon_{\mathcal{G}}$: tuple $\langle\langle A, m\rangle, \alpha\sigma, \langle A', m'\rangle\rangle \in \textsc{tell}_f$ if
- σ is a legal parameter substitution for α in A, and
- there exists $\theta \in \textsc{eval}(\textsc{add}^{A}_{\alpha\sigma})$ such that: *(i)* θ and m agree on the common values in their domains (this enforces the deterministic semantics for services); *(ii)* $m' = m \cup \theta$; *(iii)* $\langle A, \textsc{add}^{A}_{\alpha\sigma}\theta, \textsc{del}^{A}_{\alpha\sigma}, A'\rangle \in f$, where $\textsc{add}^{A}_{\alpha\sigma}\theta$ denotes the set of facts obtained by applying θ over the atoms in $\textsc{add}^{A}_{\alpha\sigma}$; *(iv)* A' is T-consistent.

As a last preliminary notion towards the parametric execution semantics of GKABs, we specify when a state $\langle A, m, \delta\rangle$ is considered to be *final* by its program δ, written $\langle A, m, \delta\rangle \in \mathbb{F}$. This is done by defining the set $\mathbb{F}$ of final states as follows:
1. $\langle A, m, \varepsilon\rangle \in \mathbb{F}$;
2. $\langle A, m, \delta_1|\delta_2\rangle \in \mathbb{F}$ if $\langle A, m, \delta_1\rangle \in \mathbb{F}$ or $\langle A, m, \delta_2\rangle \in \mathbb{F}$;
3. $\langle A, m, \delta_1;\delta_2\rangle \in \mathbb{F}$ if $\langle A, m, \delta_1\rangle \in \mathbb{F}$ and $\langle A, m, \delta_2\rangle \in \mathbb{F}$;
4. $\langle A, m, \textbf{if } \varphi \textbf{ then } \delta_1 \textbf{ else } \delta_2\rangle \in \mathbb{F}$ if $\textsc{ans}(\varphi, T, A) = \mathsf{true}$, and $\langle A, m, \delta_1\rangle \in \mathbb{F}$;
5. $\langle A, m, \textbf{if } \varphi \textbf{ then } \delta_1 \textbf{ else } \delta_2\rangle \in \mathbb{F}$ if $\textsc{ans}(\varphi, T, A) = \mathsf{false}$, and $\langle A, m, \delta_2\rangle \in \mathbb{F}$;
6. $\langle A, m, \textbf{while } \varphi \textbf{ do } \delta\rangle \in \mathbb{F}$ if $\textsc{ans}(\varphi, T, A) = \mathsf{false}$;
7. $\langle A, m, \textbf{while } \varphi \textbf{ do } \delta\rangle \in \mathbb{F}$ if $\textsc{ans}(\varphi, T, A) = \mathsf{true}$, and $\langle A, m, \delta\rangle \in \mathbb{F}$.

Now, given a filter relation f, we define the *program execution relation* $\xrightarrow{\alpha\sigma, f}$, describing how an atomic action with parameters simultaneously evolves the data- and program-state:

1. $\langle A, m, \textbf{pick } Q(\vec{p}).\alpha(\vec{p})\rangle \xrightarrow{\alpha\sigma, f} \langle A', m', \varepsilon\rangle$, if $\langle\langle A, m\rangle, \alpha\sigma, \langle A', m'\rangle\rangle \in \textsc{tell}_f$;
2. $\langle A, m, \delta_1|\delta_2\rangle \xrightarrow{\alpha\sigma, f} \langle A', m', \delta'\rangle$, if $\langle A, m, \delta_1\rangle \xrightarrow{\alpha\sigma, f} \langle A', m', \delta'\rangle$ or $\langle A, m, \delta_2\rangle \xrightarrow{\alpha\sigma, f} \langle A', m', \delta'\rangle$;
3. $\langle A, m, \delta_1;\delta_2\rangle \xrightarrow{\alpha\sigma, f} \langle A', m', \delta_1';\delta_2\rangle$, if $\langle A, m, \delta_1\rangle \xrightarrow{\alpha\sigma, f} \langle A', m', \delta_1'\rangle$;
4. $\langle A, m, \delta_1;\delta_2\rangle \xrightarrow{\alpha\sigma, f} \langle A', m', \delta_2'\rangle$, if $\langle A, m, \delta_1\rangle \in \mathbb{F}$, and $\langle A, m, \delta_2\rangle \xrightarrow{\alpha\sigma, f} \langle A', m', \delta_2'\rangle$;
5. $\langle A, m, \textbf{if } \varphi \textbf{ then } \delta_1 \textbf{ else } \delta_2\rangle \xrightarrow{\alpha\sigma, f} \langle A', m', \delta_1'\rangle$, if $\textsc{ans}(\varphi, T, A) = \mathsf{true}$, and $\langle A, m, \delta_1\rangle \xrightarrow{\alpha\sigma, f} \langle A', m', \delta_1'\rangle$;
6. $\langle A, m, \textbf{if } \varphi \textbf{ then } \delta_1 \textbf{ else } \delta_2\rangle \xrightarrow{\alpha\sigma, f} \langle A', m', \delta_2'\rangle$, if $\textsc{ans}(\varphi, T, A) = \mathsf{false}$, and $\langle A, m, \delta_2\rangle \xrightarrow{\alpha\sigma, f} \langle A', m', \delta_2'\rangle$;
7. $\langle A, m, \textbf{while } \varphi \textbf{ do } \delta\rangle \xrightarrow{\alpha\sigma, f} \langle A', m', \delta'; \textbf{while } \varphi \textbf{ do } \delta\rangle$, if $\textsc{ans}(\varphi, T, A) = \mathsf{true}$, and $\langle A, m, \delta\rangle \xrightarrow{\alpha\sigma, f} \langle A', m', \delta'\rangle$.

Given a GKAB $\mathcal{G} = \langle T, A_0, \Gamma, \delta\rangle$ and a filter relation f, we finally define the *transition system of $\mathcal{G}$ w.r.t. f*, written $\Upsilon^f_{\mathcal{G}}$, as $\langle \Delta, T, \Sigma, s_0, abox, \Rightarrow\rangle$, where $s_0 = \langle A_0, \emptyset, \delta\rangle$, and Σ and $\Rightarrow$ are defined by simultaneous induction as the smallest sets such that $s_0 \in \Sigma$, and if $\langle A, m, \delta\rangle \in \Sigma$ and $\langle A, m, \delta\rangle \xrightarrow{\alpha\sigma, f} \langle A', m', \delta'\rangle$, then $\langle A', m', \delta'\rangle \in \Sigma$ and $\langle A, m, \delta\rangle \Rightarrow \langle A', m', \delta'\rangle$. By suitbably concretizing the filter relation, we can obtain a plethora of execution semantics.

Standard and Inconsistency-Aware Semantics. Given a GKAB $\mathcal{G} = \langle T, A_0, \Gamma, \delta\rangle$, we exploit filter relations to define its standard execution semantics (reconstructing that of [Calvanese *et al.*, 2013b] for normal KABs), and three inconsistency-aware semantics that incorporate the repair-based approaches reviewed in Section 2.2. In particular, we introduce 4 filter relations f_S, f_B, f_C, f_E, as follows. Given an ABox A, an atomic action $\alpha(\vec{p}) \in \Gamma$, a legal parameter substitution σ for α in A, and a service call evaluation $\theta \in \textsc{eval}(\textsc{add}^{A}_{\alpha\sigma})$, let $F^+ = \textsc{add}^{A}_{\alpha\sigma}\theta$ and $F^- = \textsc{del}^{A}_{\alpha\sigma}$. We then have $\langle A, F^+, F^-, A'\rangle \in f$, where

$$\begin{cases} A' = (A \setminus F^-) \cup F^+, & \text{if } f = f_S \\ A' \in \textsc{b-rep}(T, (A \setminus F^-) \cup F^+), & \text{if } f = f_B \\ A' = \textsc{c-rep}(T, (A \setminus F^-) \cup F^+), & \text{if } f = f_C \\ A' = \textsc{evol}(T, A, F^+, F^-), & \text{if } f = f_E \text{ and} \\ & \quad F^+ \text{ is } T\text{-consistent} \end{cases}$$

Filter f_S gives rise to the *standard execution semantics* for $\mathcal{G}$, since it just applies the update induced by the ground atomic action $\alpha\sigma$ (giving priority to additions over deletions). Filter f_B gives rise to the *b-repair execution semantics* for $\mathcal{G}$, where inconsistent ABoxes are repaired by non-deterministically picking a b-repair. Filter f_C gives rise to the *c-repair execution semantics* for $\mathcal{G}$, where inconsistent ABoxes are repaired by computing their unique c-repair. Filter f_E gives rise to the *b-evol execution semantics* for $\mathcal{G}$, where for updates leading to inconsistent ABoxes, their unique bold-evolution is computed. We call the GKABs adopting these semantics *S-GKABs*, *B-GKABs*, *C-GKABs*, and *E-GKABs*, respectively, and we group the last three forms of GKABs under the umbrella of *inconsistency-aware GKABs (I-GKABs)*.

Transforming S-KABs to S-GKABs. We close this section by showing that our S-GKABs are able to capture normal S-KABs in the literature [Bagheri Hariri *et al.*, 2013b; Calvanese *et al.*, 2013b]. In particular, we show the following.

Theorem 1. *Verification of $\mu\mathcal{L}_A^{\text{EQL}}$ properties over S-KABs can be recast as verification over S-GKABs.*

Proof sketch. We provide a translation τ_S that, given an S-KAB $\mathcal{K} = \langle T, A_0, \Gamma, \Pi\rangle$ with transition system $\Upsilon_{\mathcal{K}}^S$, generates an S-GKAB $\tau_S(\mathcal{K}) = \langle T, A_0, \Gamma, \delta\rangle$. Program δ is obtained from Π as $\delta = \textbf{while}\ \mathsf{true}\ \textbf{do}\ (a_1|a_2|\ldots|a_{|\Pi|})$, where, for each condition-action rule $Q_i(\vec{x}) \mapsto \alpha_i(\vec{x}) \in \Pi$, we have $a_i = \textbf{pick}\ Q_i(\vec{x}).\alpha_i(\vec{x})$. The translation produces a program that continues forever to non-deterministically pick an executable action with parameters (as specified by Π), or stops if no action is executable. It can be then proven directly that for every $\mu\mathcal{L}_A^{\text{EQL}}$ property Φ, $\Upsilon_{\mathcal{K}}^S \models \Phi$ iff $\Upsilon_{\tau_S(\mathcal{K})}^{f_S} \models \Phi$. $\square$

4 Compilation of Inconsistency Management

This section provides a general account of inconsistency management in GKABs, proving that all inconsistency-aware variants introduced in Section 3 can be reduced to S-GKABs.

Theorem 2. *Verification of $\mu\mathcal{L}_A^{\text{EQL}}$ properties over I-GKABs can be recast as verification over S-GKABs.*

The remainder of this section is devoted to prove this result, case by case. Our general strategy is to show that S-GKABs are sufficiently expressive to incorporate the repair-based approaches of Section 2.2, so that an action executed under a certain inconsistency semantics can be compiled into a Golog program that applies the action with the standard semantics, and then explicitly handles the inconsistency, if needed.

We start by recalling that checking whether a *DL-Lite*$_{\mathcal{A}}$ KB $\langle T, A\rangle$ is inconsistent is FO rewritable, i.e., can be reduced to evaluating a boolean query Q_{unsat}^T over A (interpreted as a database) [Calvanese *et al.*, 2007b]. To express such queries compactly, we make use of the following abbreviations. For role $R = P^-$, atom $R(x,y)$ denotes $P(y,x)$. For concept $B = \exists P$, atom $B(x)$ denotes $P(x,\_)$, where '_' stands for an anonymous existentially quantified variable. Similarly, for $B = \exists P^-$, atom $B(x)$ denotes $P(\_,x)$.

In particular, the boolean query Q_{unsat}^T is:

$$\begin{aligned} Q_{\text{unsat}}^T = & \textstyle\bigvee_{(\mathsf{funct}\ R)\in T} \exists x,y,z.q_{\text{unsat}}^f((\mathsf{funct}\ R),x,y,z) \lor \\ & \textstyle\bigvee_{T\models B_1\sqsubseteq\neg B_2} \exists x.q_{\text{unsat}}^n(B_1 \sqsubseteq \neg B_2, x) \lor \\ & \textstyle\bigvee_{T\models R_1\sqsubseteq\neg R_2} \exists x,y.q_{\text{unsat}}^n(R_1 \sqsubseteq \neg R_2, x, y) \end{aligned}$$

where:

- $q_{\text{unsat}}^f((\mathsf{funct}\ R),x,y,z) = R(x,y) \land R(x,z) \land \neg[y=z]$;
- $q_{\text{unsat}}^n(B_1 \sqsubseteq \neg B_2, x) = B_1(x) \land B_2(x)$;
- $q_{\text{unsat}}^n(R_1 \sqsubseteq \neg R_2, x, y) = R_1(x,y) \land R_2(x,y)$.

4.1 From B-GKABs to S-GKABs

To encode B-GKABs into S-GKABs, we use a special fact $\mathsf{M}(\mathit{rep})$ to distinguish *stable* states, where an atomic action can be applied, from intermediate states used by the S-GKABs to incrementally remove inconsistent facts from the ABox. Stable/repair states are marked by the absence/presence of $\mathsf{M}(\mathit{rep})$. To set/unset $\mathsf{M}(\mathit{rep})$, we define set $\Gamma_{rep} = \{\alpha_{rep}^+(), \alpha_{rep}^-()\}$ of actions, where $\alpha_{rep}^+() : \{\mathsf{true} \rightsquigarrow \textbf{add}\ \{\mathsf{M}(\mathit{rep})\}\}$, and $\alpha_{rep}^-() : \{\mathsf{true} \rightsquigarrow \textbf{del}\ \{\mathsf{M}(\mathit{rep})\}\}$.

Given a B-GKAB $\mathcal{G} = \langle T, A_0, \Gamma, \delta\rangle$, we define the *set* Γ_b^T *of b-repair actions* and the set Λ_b^T of *b-repair atomic action invocations* as follows. For each functionality assertion $(\mathsf{funct}\ R) \in T$, we include in Γ_b^T and Λ_b^T respectively:

- **pick** $\exists z.q_{\text{unsat}}^f((\mathsf{funct}\ R),x,y,z).\alpha_F(x,y) \in \Lambda_b^T$, and
- $\alpha_F(x,y) : \{R(x,z) \land \neg[z=y] \rightsquigarrow \textbf{del}\ \{R(x,z)\}\} \in \Gamma_b^T$

This invocation repairs an inconsistency related to $(\mathsf{funct}\ R)$ by removing all tuples causing the inconsistency, except one. For each negative concept inclusion $B_1 \sqsubseteq \neg B_2$ s.t. $T \models B_1 \sqsubseteq \neg B_2$, we include in Γ_b^T and Λ_b^T respectively:

- **pick** $q_{\text{unsat}}^n(B_1 \sqsubseteq \neg B_2, x).\alpha_{B_1}(x) \in \Lambda_b^T$, and
- $\alpha_{B_1}(x) : \{\mathsf{true} \rightsquigarrow \textbf{del}\ \{B_1(x)\}\} \in \Gamma_b^T$

This invocation repairs an inconsistency related to $B_1 \sqsubseteq \neg B_2$ by removing an individual that is both in B_1 and B_2 from B_1. Similarly for negative role inclusions. Given $\Lambda_b^T = \{a_1, \ldots, a_n\}$, we then define the *b-repair program*

$$\delta_b^T = \textbf{while}\ Q_{\text{unsat}}^T\ \textbf{do}\ (a_1|a_2|\ldots|a_n),$$

Intuitively, δ_b^T iterates while the ABox is inconsistent, and at each iteration, non-deterministically picks one of the sources of inconsistency, and removes one or more facts causing it. Consequently, the loop is guaranteed to terminate, in a state that corresponds to one of the b-repairs of the initial ABox.

With this machinery at hand, we are ready to define a translation τ_B that, given $\mathcal{G}$, produces S-GKAB $\tau_B(\mathcal{G}) = \langle T_p, A_0, \Gamma \cup \Gamma_b^T \cup \Gamma_{rep}, \delta'\rangle$, where only the positive inclusion assertions T_p of the original TBox T are maintained (guaranteeing that $\tau_B(\mathcal{G})$ never encounters inconsistency). Program δ' is obtained from program δ of $\mathcal{G}$ by replacing each occurrence of an atomic action invocation **pick** $Q(\vec{p}).\alpha(\vec{p})$ with

$$\textbf{pick}\ Q(\vec{p}).\alpha(\vec{p}); \textbf{pick}\ \mathsf{true}.\alpha_{rep}^+(); \delta_b^T; \textbf{pick}\ \mathsf{true}.\alpha_{rep}^-()$$

This program concatenates the original action invocation with a corresponding "repair" phase. Obviously, this means that when an inconsistent ABox is produced, a single transition in $\mathcal{G}$ corresponds to a sequence of transitions in $\tau_B(\mathcal{G})$. Hence, we need to introduce a translation t_B that takes a $\mu\mathcal{L}_A^{\text{EQL}}$ formula Φ over $\mathcal{G}$ and produces a corresponding formula over $\tau_B(\mathcal{G})$. This is done by first obtaining formula $\Phi' = \textsc{nnf}(\Phi)$, where $\textsc{nnf}(\Phi)$ denotes the *negation normal form* of Φ. Then, every subformula of Φ of the form $\langle - \rangle\Psi$ becomes $\langle - \rangle\langle - \rangle\mu Z.((\mathsf{M}(\mathit{rep}) \land \langle - \rangle Z) \lor (\neg\mathsf{M}(\mathit{rep}) \land t_B(\Psi)))$, so as to translate a next-state condition over $\mathcal{G}$ into reachability of the next stable state over $\tau_B(\mathcal{G})$. Similarly for $[-]\Psi$.

With these two translations at hand, we can show that $\Upsilon_{\mathcal{G}}^{f_B} \models \Phi$ iff $\Upsilon_{\tau_B(\mathcal{G})}^{f_S} \models t_B(\Phi)$.

4.2 From C-GKABs to S-GKABs

Making inconsistency management for C-GKABs explicit requires just a single action, which removes all individuals that are involved in some form of inconsistency. Hence, given a TBox T, we define a 0-ary *c-repair action* α_c^T, where $\textsc{Eff}(\alpha_c^T)$ is the smallest set containing the following effects:

- for each assertion $(\mathsf{funct}\ R) \in T$, $q^f_{\mathsf{unsat}}((\mathsf{funct}\ R), x, y, z) \rightsquigarrow \{\mathbf{del}\ \{R(x,y), R(x,z)\}\}$
- for each assertion $B_1 \sqsubseteq \neg B_2$ s.t. $T \models B_1 \sqsubseteq \neg B_2$, $q^n_{\mathsf{unsat}}(B_1 \sqsubseteq \neg B_2, x) \rightsquigarrow \{\mathbf{del}\ \{B_1(x), B_2(x)\}\}$;
- similarly for negative role inclusions.

Notice that all effects are guarded by queries that extract only individuals involved in an inconsistency. Hence, other facts are kept unaltered, which also means that α_c^T is a no-op when applied over a T-consistent ABox. We define a translation τ_C that, given a C-GKAB $\mathcal{G} = \langle T, A_0, \Gamma, \delta\rangle$, generates an S-GKAB $\tau_C(\mathcal{G}) = \langle T_p, A_0, \Gamma \cup \{\alpha_c^T\}, \delta'\rangle$, which, as for B-GKABs, only maintains positive inclusion assertions of T. Program δ' is obtained from δ by replacing each occurrence of an atomic action invocation of the form **pick** $Q(\vec{p}).\alpha(\vec{p})$ with **pick** $Q(\vec{p}).\alpha(\vec{p})$; **pick** $\mathsf{true}.\alpha_c^T()$. This attests that each transition in $\mathcal{G}$ corresponds to a sequence of two transitions in $\tau_C(\mathcal{G})$: the first mimics the action execution, while the second computes the c-repair of the obtained ABox.

A $\mu\mathcal{L}_A^{\mathrm{EQL}}$ property Φ over $\mathcal{G}$ can then be recast as a corresponding property over $\tau_C(\mathcal{G})$ that substitutes each subformula $\langle - \rangle\Psi$ of Φ with $\langle - \rangle\langle - \rangle\Psi$ (similarly for $[-]\Phi$). By denoting this translation with t_{dup}, we get $\Upsilon_{\mathcal{G}}^{f_C} \models \Phi$ iff $\Upsilon_{\tau_C(\mathcal{G})}^{f_S} \models t_{dup}(\Phi)$.

4.3 From E-GKABs to S-GKABs

Differently from the case of B-GKABs and C-GKABs, E-GKABs pose two challenges: *(i)* when applying an atomic action (and managing the possibly arising inconsistency) it is necessary to distinguish those facts that are newly introduced by the action from those already present in the system; *(ii)* the evolution semantics can be applied only if the facts to be added are consistent with the TBox, and hence an additional check is required to abort the action execution if this is not the case. To this aim, given a TBox T, we duplicate concepts and roles in T, introducing a fresh concept name N^n for every concept name N in T (similarly for roles). The key idea is to insert those individuals that are added to N also in N^n, so as to trace that they are part of the update.

The first issue described above is then tackled by compiling the bold evolution semantics into a 0-ary *evolution action* α_e^T, where $\textsc{Eff}(\alpha_e^T)$ is the smallest set of effects containing:

- for each assertion $(\mathsf{funct}\ R) \in T$, $\exists z.q^f_{\mathsf{unsat}}((\mathsf{funct}\ R), x, y, z) \wedge R^n(x,y) \rightsquigarrow \{\mathbf{del}\ \{R(x,z)\}\}$
- for each assertion $B_1 \sqsubseteq \neg B_2$ s.t. $T \models B_1 \sqsubseteq \neg B_2$, $q^n_{\mathsf{unsat}}(B_1 \sqsubseteq \neg B_2, x) \wedge B_1^n(x) \rightsquigarrow \{\mathbf{del}\ \{B_2(x)\}\}$;
- similarly for negative role inclusion assertions;
- for each concept name N, $N^n(x) \rightsquigarrow \{\mathbf{del}\ \{N^n(x)\}\}$;
- similarly for role names.

These effects mirror those of Section 4.2, with the difference that they asymmetrically remove old facts when inconsistency arises. The last two bullets guarantee that the content of concept and role names tracking the newly added facts are flushed. We then define a translation τ_E that, given an E-GKAB $\mathcal{G} = \langle T, A_0, \Gamma, \delta\rangle$, generates an S-GKAB $\tau_E(\mathcal{G}) = \langle T_p \cup T^n, A_0, \Gamma' \cup \{\alpha_e^T\}, \delta'\rangle$, where:

- T^n is obtained from T by renaming each concept name N in T into N^n (similarly for roles). In this way, the original concepts/roles are only subject in $\tau_E(\mathcal{G})$ to the positive inclusion assertions of T, while concepts/roles tracking newly inserted facts are subject also to negative constraints. This blocks the generation of the successor state when the facts to be added to the current ABox are T-inconsistent.
- Γ' is obtained by translating each action in $\alpha(\vec{p}) \in \Gamma$ into action $\alpha'(\vec{p})$, such that for each effect $Q \rightsquigarrow \mathbf{add}\ F^+, \mathbf{del}\ F^- \in \textsc{Eff}(\alpha)$, we have $Q \rightsquigarrow \mathbf{add}\ F^+ \cup F^{+^n}, \mathbf{del}\ F^- \in \textsc{Eff}(\alpha')$ where F^{+^n} duplicates F^+ by using the vocabulary for newly introduced facts.
- δ' is obtained from δ by replacing each action invocation **pick** $Q(\vec{p}).\alpha(\vec{p})$ with **pick** $Q(\vec{p}).\alpha'(\vec{p})$; **pick** $\mathsf{true}.\alpha_e^T()$.

By exploiting the same $\mu\mathcal{L}_A^{\mathrm{EQL}}$ translation used in Section 4.2, we obtain that $\Upsilon_{\mathcal{G}}^{f_E} \models \Phi$ iff $\Upsilon_{\tau_E(\mathcal{G})}^{f_S} \models t_{dup}(\Phi)$.

5 From Golog to Standard KABs

We close our tour by showing that S-GKABs can be compiled into the normal S-KABs of [Bagheri Hariri *et al.*, 2013b; Calvanese *et al.*, 2013b].

Theorem 3. *Verification of $\mu\mathcal{L}_A^{\mathrm{EQL}}$ properties over S-GKABs can be recast as verification over S-KABs.*

Proof sketch. We introduce a translation from S-GKABs to S-KABs, and from $\mu\mathcal{L}_A^{\mathrm{EQL}}$ properties over S-GKABs to corresponding properties over S-KABs, in such a way that verification in the first setting can be reduced to verification in the second setting. The translation is quite involved, for space reasons we refer to [Calvanese *et al.*, 2015] for details. □

From Theorems 1 and 3, we obtain that S-KABs and S-GKABs are expressively equivalent. From Theorems 2 and 3, we get our second major result: inconsistency-management can be compiled into an S-KAB by concatenating the two translations from I-GKABs to S-GKABs, and then to S-KABs.

Theorem 4. *Verification of $\mu\mathcal{L}_A^{\mathrm{EQL}}$ properties over I-GKABs can be recast as verification over S-KABs.*

Even more interesting is the fact that the semantic property of *run-boundedness* [Bagheri Hariri *et al.*, 2013a; 2013b] is preserved by all translations presented in this paper. Intuitively, run-boundedness requires that every run of the system cumulatively encounters at most a bounded number of individuals. Unboundedly many individuals can still be present in the overall system, provided that they do not accumulate in the same run. Thanks to the preservation of run-boundedness, and to the compilation of I-GKABs into S-KABs, we get:

Theorem 5. *Verification of $\mu\mathcal{L}_A^{\mathrm{EQL}}$ properties over run-bounded I-GKABs is decidable, and reducible to standard μ-calculus finite-state model checking.*

Proof sketch. The claim follows by combining the fact that all translations preserve run-boundedness, Theorem 4, and the results in [Bagheri Hariri *et al.*, 2013a; 2013b] for run-bounded S-KABs. □

6 Conclusion

We introduced GKABs, which extend KABs with Golog-inspired high-level programs, and provided a parametric execution semantics supporting an elegant treatment of inconsistency. We have shown that verification of rich temporal properties over (inconsistency-aware) GKABs can be recast as verification over standard KABs, by encoding the semantics of inconsistency in terms of Golog programs and specific inconsistency-management actions, and Golog programs into standard KAB condition-action rules. An overview of our reductions is depicted below. Our approach is very general, and can be seamlessly extended to account for other mechanisms for handling inconsistency, and more in general data cleaning.

$$\begin{tikzcd} \text{B-GKABs} \arrow[dr] & \text{C-GKABs} \arrow[d] & \text{E-GKABs} \arrow[dl] \\ & \text{S-GKABs} \arrow[r, shift left] & \text{S-KABs} \arrow[l, shift left] \end{tikzcd}$$

Acknowledgments. This research has been partially supported by the EU IP project Optique (*Scalable End-user Access to Big Data*), grant agreement n. FP7-318338, and by the UNIBZ internal project KENDO (*Knowledge-driven ENterprise Distributed cOmputing*).

References

[Baader *et al.*, 2003] Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press, 2003.

[Bagheri Hariri *et al.*, 2013a] Babak Bagheri Hariri, Diego Calvanese, Giuseppe De Giacomo, Alin Deutsch, and Marco Montali. Verification of relational data-centric dynamic systems with external services. In *Proc. of the 32nd ACM SIGACT SIGMOD SIGAI Symp. on Principles of Database Systems (PODS)*, pages 163–174, 2013.

[Bagheri Hariri *et al.*, 2013b] Babak Bagheri Hariri, Diego Calvanese, Marco Montali, Giuseppe De Giacomo, Riccardo De Masellis, and Paolo Felli. Description logic Knowledge and Action Bases. *J. of Artificial Intelligence Research*, 46:651–686, 2013.

[Belardinelli *et al.*, 2012] Francesco Belardinelli, Alessio Lomuscio, and Fabio Patrizi. An abstraction technique for the verification of artifact-centric systems. In *Proc. of the 13th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR)*, pages 319–328, 2012.

[Bertossi, 2006] Leopoldo E. Bertossi. Consistent query answering in databases. *SIGMOD Record*, 35(2):68–76, 2006.

[Calvanese *et al.*, 2007a] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati. EQL-Lite: Effective first-order query processing in description logics. In *Proc. of the 20th Int. Joint Conf. on Artificial Intelligence (IJCAI)*, pages 274–279, 2007.

[Calvanese *et al.*, 2007b] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati. Tractable reasoning and efficient query answering in description logics: The *DL-Lite* family. *J. of Automated Reasoning*, 39(3):385–429, 2007.

[Calvanese *et al.*, 2009] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, Antonella Poggi, Mariano Rodríguez-Muro, and Riccardo Rosati. Ontologies and databases: The *DL-Lite* approach. In *Reasoning Web. Semantic Technologies for Informations Systems – 5th Int. Summer School Tutorial Lectures (RW)*, volume 5689 of *LNCS*, pages 255–356. Springer, 2009.

[Calvanese *et al.*, 2010] Diego Calvanese, Evgeny Kharlamov, Werner Nutt, and Dmitriy Zheleznyakov. Evolution of *DL-Lite* knowledge bases. In *Proc. of the 9th Int. Semantic Web Conf. (ISWC)*, volume 6496 of *LNCS*, pages 112–128. Springer, 2010.

[Calvanese *et al.*, 2011] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Riccardo Rosati. Actions and programs over description logic knowledge bases: A functional approach. In *Knowing, Reasoning, and Acting: Essays in Honour of Hector Levesque*. College Publications, 2011.

[Calvanese *et al.*, 2013a] Diego Calvanese, Giuseppe De Giacomo, and Marco Montali. Foundations of data aware process analysis: A database theory perspective. In *Proc. of the 32nd ACM SIGACT SIGMOD SIGAI Symp. on Principles of Database Systems (PODS)*, 2013.

[Calvanese *et al.*, 2013b] Diego Calvanese, Evgeny Kharlamov, Marco Montali, Ario Santoso, and Dmitriy Zheleznyakov. Verification of inconsistency-tolerant knowledge and action bases. In *Proc. of the 23rd Int. Joint Conf. on Artificial Intelligence (IJCAI)*, 2013.

[Calvanese *et al.*, 2015] Diego Calvanese, Marco Montali, and Ario Santoso. Verification of generalized inconsistency-aware knowledge and action bases (extended version). CoRR Technical Report arXiv:1504.08108, arXiv.org e-Print archive, 2015. Available at `http://arxiv.org/abs/1504.08108`.

[Deutsch *et al.*, 2009] Alin Deutsch, Richard Hull, Fabio Patrizi, and Victor Vianu. Automatic verification of data-centric business processes. In *Proc. of the 12th Int. Conf. on Database Theory (ICDT)*, pages 252–267, 2009.

[Eiter and Gottlob, 1992] Thomas Eiter and Georg Gottlob. On the complexity of propositional knowledge base revision, updates and counterfactuals. *Artificial Intelligence*, 57:227–270, 1992.

[Hull, 2008] Richard Hull. Artifact-centric business process models: Brief survey of research results and challenges. In *Proc. of the 7th Int. Conf. on Ontologies, DataBases, and Applications of Semantics (ODBASE)*, volume 5332 of *LNCS*, pages 1152–1163. Springer, 2008.

[Lembo *et al.*, 2010] Domenico Lembo, Maurizio Lenzerini, Riccardo Rosati, Marco Ruzzi, and Domenico Fabio Savo. Inconsistency-tolerant semantics for description logics. In *Proc. of the 4th Int. Conf. on Web Reasoning and Rule Systems (RR)*, pages 103–117, 2010.

[Levesque *et al.*, 1997] H. J. Levesque, R. Reiter, Y. Lesperance, F. Lin, and R. Scherl. GOLOG: A logic programming language for dynamic domains. *J. of Logic Programming*, 31:59–84, 1997.

[Levesque, 1984] Hector J. Levesque. Foundations of a functional approach to knowledge representation. *Artificial Intelligence*, 23:155–212, 1984.

[Montali *et al.*, 2014] Marco Montali, Diego Calvanese, and Giuseppe De Giacomo. Verification of data-aware commitment-based multiagent systems. In *Proc. of the 13th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 157–164, 2014.

[Vianu, 2009] Victor Vianu. Automatic verification of database-driven systems: a new frontier. In *Proc. of the 12th Int. Conf. on Database Theory (ICDT)*, pages 1–13, 2009.