ECAI 2012
Luc De Raedt et al. (Eds.)
© 2012 The Author(s).
This article is published online with Open Access by IOS Press and distributed under the terms of the Creative Commons Attribution Non-Commercial License.
doi:10.3233/978-1-61499-098-7-103

Verification of Description Logic Knowledge and Action Bases

Babak Bagheri Hariri[1], **Diego Calvanese**[1], **Giuseppe De Giacomo**[2], **Riccardo De Masellis**[2], **Paolo Felli**[2], **Marco Montali**[1]

Abstract. We introduce description logic (DL) Knowledge and Action Bases (KAB), a mechanism that provides both a semantically rich representation of the information on the domain of interest in terms of a DL KB and a set of actions to change such information over time, possibly introducing new objects. We resort to a variant of *DL-Lite* where UNA is not enforced and where equality between objects may be asserted and inferred. Actions are specified as sets of conditional effects, where conditions are based on epistemic queries over the KB (TBox and ABox), and effects are expressed in terms of new ABoxes. We address the verification of temporal properties expressed in a variant of first-order μ-calculus where a controlled form of quantification across states is allowed. Notably, we show decidability of verification, under a suitable restriction inspired by the notion of weak acyclicity in data exchange.

1 Introduction

Recent work in business processes, services and databases is bringing forward the need of considering both data and processes as first-class citizens in process and service design [14, 29]. In particular, the so called artifact-centric approaches, which advocate a sort of middle ground between a conceptual formalization of dynamic systems and their actual implementation, are promising to be effective in practice [12]. The verification of temporal properties in the presence of data represents a significant research challenge, since data makes the system infinite-state, and neither finite-state model checking [11] nor most of the current technique for infinite-state model checking, which mostly tackle recursion [6], apply to this case. Recently, there have been some advancements on this issue [13, 4, 5], considering a suitably constrained relational database settings.

The motivation for our work comes when we want to enrich data-intensive business processes with a semantic level. This leads us to look into how to combine first-order data, ontologies, and processes, while maintaining basic inference tasks (specifically verification) decidable. In this setting, we capture the domain of interest in terms of a semantically rich language as those provided by ontological languages based on Description Logics (DLs) [2]. Such languages natively deal with incomplete knowledge in the modeled domain. This additional flexibility comes with an added cost, however: differently from relational databases, to evaluate queries we need to resort to logical implication. Moreover incomplete information combined with the ability of evolving the system through actions results in a notoriously difficult setting [30]. In particular, due to the nature of DL assertions (which in general are not definitions but constraints on models), we get one of the most difficult kinds of domain descriptions for reasoning about actions [26], which amounts to dealing with complex forms of state constraints [18, 19]. To overcome this difficulty, virtually all effective solutions presented in the literature are based on a so-called "functional view of knowledge bases" [17]: the KB provides the ability of querying based on logical implication, and the ability of progressing it to a "new" KB through forms of updates [3, 10]. Notice that this functional view is tightly related to an epistemic interpretation of the KB [8]. Indeed our work is also related to that on Epistemic Dynamic Logic [28], and, though out of the scope of this paper, the decidability results presented here could find application in the context of that research as well.

We follow this functional view of KBs. However, a key point of our work is that at each execution step external information is incorporated into the system in form of new objects (denoted by *Skolem terms*), that is, our systems are not closed wrt the available information. This makes our framework particularly interesting and challenging. In particular, the presence of these objects requires a specific treatment of equality, since as the system progresses and new information is acquired, distinct object terms may be inferred to denote the same object.

Specifically, we introduce the so-called *Knowledge and Action Bases* (KABs). A KAB is equipped with a TBox, expressed in a variant of *DL-Lite*$_{\mathcal{A}}$ [9, 1], which extends the core of the Web Ontology Language OWL 2 QL[3] and is particularly well suited for data management. Such a TBox captures intensional information on the domain of interest, similarly to UML class diagrams or other conceptual data models, though as a software component to be used at runtime. The KAB includes also an ABox, which acts as a storage or state. The ABox maintains the data of interest, which are accessed by relying on query answering based on logical implication (certain answers). Notably, our variant of *DL-Lite*$_{\mathcal{A}}$ is without UNA, since, as discussed above, we cannot have UNA for Skolems, and we allow for explicit equality assertions in the ABox. Technically, this breaks the first-order rewritability of *DL-Lite*$_{\mathcal{A}}$ query answering, and requires that, in addition to the rewriting process, inference on equality is performed. As a query language, we use unions of conjunctive queries, possibly composing their certain answers through full FOL constructs. This gives rise to an *epistemic query language* that asks about what is "known" by the current KB [8]. The KAB then contains *actions* whose execution changes the state of the KB, i.e., its ABox. Such actions are specified as sets of conditional effects, where conditions are (epistemic) queries over the KB and effects are expressed in terms of new ABoxes. Actions have no static pre-conditions, instead

[1] Free University of Bozen-Bolzano, Italy, email: *lastname*@inf.unibz.it
[2] Sapienza Università di Roma, email: *lastname*@dis.uniroma1.it

[3] http://www.w3.org/TR/owl2-profiles/

a process is used to specify which actions can be executed at each step. For simplicity, we model such processes as condition/action rules, where the condition is again expressed as a query over the KB.

In this setting, we address the verification of temporal/dynamic properties expressed in a first-order variant of μ-calculus [24, 27], where atomic formulae are queries over the KB that can refer both to known and unknown objects, and where a controlled form of quantification across states is allowed. Notice that all previous decidability results on actions over DL KBs assumed that no information is coming from outside of the system, in the sense that no new objects are added while executing actions [3, 10]. In this paper, instead, we allow for arbitrary introduction of new objects. Unsurprisingly, we show that even for very simple KABs and temporal properties, verification is undecidable. However, we also show that for a very rich class of KABs, verification is indeed decidable and reducible to finite-state model checking. To obtain this result, following [4], we rely on recent results in data exchange on the finiteness of the chase of tuple-generating dependencies [15], though, in our case, we need to extend the technique to deal with *(i)* incomplete information; *(ii)* inference on equality; *(iii)* quantification across states in the verification language. Proofs are dropped for brevity.

2 Knowledge Base Formalism

For expressing knowledge bases, we use *DL-Lite*$_{\text{NU}}$, a variant of the *DL-Lite*$_{\mathcal{A}}$ language [25, 7] in which we drop the *unique name assumption* (UNA) [1]. The syntax of *concept* and role *expressions* in *DL-Lite*$_{\text{NU}}$ is as follows:

$$\begin{array}{lcl} B & \longrightarrow & N \mid \exists R \\ C & \longrightarrow & B \mid \neg B \end{array} \qquad \begin{array}{lcl} R & \longrightarrow & P \mid P^- \\ V & \longrightarrow & R \mid \neg R \end{array}$$

where N denotes a *concept name*, P a *role name*, and P^- an *inverse role*. A *DL-Lite*$_{\text{NU}}$ *knowledge base* (KB) is a pair (T, A), where:

- T is a TBox, i.e., a finite set of *TBox (inclusion and functionality) assertions* of the form $B \sqsubseteq C \mid R \sqsubseteq V \mid (\mathsf{funct}\ R)$, and
- A is an Abox, i.e., a finite set of *ABox (membership and equality) assertions* of the form $N(t_1) \mid P(t_1, t_2) \mid t_1 = t_2$, where t_1, t_2 denote individuals.

As usual in *DL-Lite*, a TBox may contain neither $(\mathsf{funct}\ P)$ nor $(\mathsf{funct}\ P^-)$ if it contains $R \sqsubseteq P$ or $R \sqsubseteq P^-$, for some role R.

We adopt the standard FOL semantics of DLs based on FOL interpretations $\mathcal{I} = (\Delta^\mathcal{I}, \cdot^\mathcal{I})$ such that $c^\mathcal{I} \in \Delta^\mathcal{I}$, $N^\mathcal{I} \subseteq \Delta^\mathcal{I}$, and $P^\mathcal{I} \subseteq \Delta^\mathcal{I} \times \Delta^\mathcal{I}$. The semantics of the construct, of TBox and ABox assertions, and the notions of *satisfaction* and of *model* are as usual. We also say that A is *consistent wrt* T if (T, A) is satisfiable, i.e., admits at least one model.

Next we introduce queries. As usual (cf. OWL 2), answers to queries are formed by terms denoting individuals explicitly mentioned in the ABox. The *domain of an ABox* A, denoted by $\text{ADOM}(A)$, is the (finite) set of terms appearing in A. A *union of conjunctive queries* (UCQ) q over a KB (T, A) is a FOL formula of the form $\exists \vec{y}_1.conj_1(\vec{x}, \vec{y}_1) \vee \cdots \vee \exists \vec{y}_n.conj_n(\vec{x}, \vec{y}_n)$, with free variables $\vec{x}$ and existentially quantified variables $\vec{y}_1, \ldots, \vec{y}_n$. Each $conj_i(\vec{x}, \vec{y}_i)$ in q is a conjunction of atoms of the form $N(z)$, $P(z, z')$, where N and P respectively denote a concept and a role name occurring in T, and z, z' are constants in $\text{ADOM}(A)$ or variables in $\vec{x}$ or $\vec{y}_i$, for some $i \in \{1, \ldots, n\}$. The *(certain) answers* to q over (T, A) is the set $ans(q, T, A)$ of substitutions[4] σ of the free variables of q with constants in $\text{ADOM}(A)$ such that $q\sigma$ evaluates to true in every model of (T, A). If q has no free variables, then it is called *boolean* and its certain answers are either true or false.

[4] As customary, we can view each substitution simply as a tuple of constants, assuming some ordering of the free variables of q.

Theorem 1 ([1]) *Computing* $ans(q, T, A)$ *of an UCQs q over a KBs* (T, A) *is* PTIME*-complete in the size of* T *and* A.

We also consider an extension of UCQs, called ECQs, which are queries of the query language *EQL-Lite*(UCQ) [8], that is, the FOL query language whose atoms are UCQs evaluated according to the certain answer semantics above. An *ECQ* over T and A is a possibly open formula of the form (where q is a UCQ):

$$Q \quad \longrightarrow \quad [q] \mid [x = y] \mid \neg Q \mid Q_1 \wedge Q_2 \mid \exists x.Q$$

The *answer to* Q *over* (T, A), is the set $\text{ANS}(Q, T, A)$ of tuples of constants in $\text{ADOM}(A)$ defined by composing the certain answers $ans(q, T, A)$ of UCQs q through first-order constructs, and interpreting existential variables as ranging over $\text{ADOM}(A)$. Following the line of the proof in [8], but considering Theorem 1 for the basic step of evaluating an UCQ, we get:

Theorem 2 *Computing* $\text{ANS}(Q, T, A)$ *of an ECQs* Q *over a KBs* (T, A) *is* PTIME*-complete in the size of* T *and* A.

We close by recalling that DL-Lite enjoys the *FO rewritability* property, which in our setting says that for every UCQ q, $ans(q, T, A) = ans(rew(q), \emptyset, A)$, where $rew(q)$ is a UCQ computed by the reformulation algorithm in [9]. Notice that, in this way, we have “compiled away” the TBox, though we still need to do logical implication w.r.t. ABox, which contains equality assertions. This result can be extended to ECQs as well [8].

3 Knowledge and Action Bases

A *Knowledge and Action Base (KAB)* is a tuple $\mathcal{K} = (T, A_0, \Gamma, \Pi)$ where T and A_0 form the *knowledge component* (or knowledge base), and Γ and Π form the *action component* (or action base). In practice, $\mathcal{K}$ is a stateful device that stores the information of interest into a KB, formed by a fixed TBox T and an initial ABox A_0, which can evolve by executing actions Γ according to the sequencing established by process Π. We describe such components in detail.

TBox. T is a *DL-Lite*$_{\text{NU}}$ TBox, used to capture the intensional knowledge about the domain of interest. Such a TBox is fixed once and for all, and does not evolve during the execution of the KAB.

ABox. A_0 is a *DL-Lite*$_{\text{NU}}$ ABox, which stores the extensional information of interest. Notice that A_0 is the ABox of the *initial state* of the KAB, and as the KAB evolves due to the effect of actions, the ABox, which is indeed the state of the system, evolves accordingly to store up-to-date information. Through actions, we acquire new information from the external world, which results in new individuals. These individuals are denoted by (ground) *Skolem terms*. The presence of Skolem terms has an impact on the treatment of equality, since in principle we need to close equality w.r.t. congruence, i.e., if $a = b$ holds, then also $f(a) = f(b)$ must hold. Closure w.r.t. congruence generates an infinite number of logically implied equality assertions. However, we are going to keep such assertions implicit, computing them only when needed. Observe that, given two complex terms verifying their equality requires a PTIME computation.

Actions. Γ is a finite set actions. An *action* $\gamma \in \Gamma$ modifies the current ABox A by adding or deleting assertions, thus generating a new ABox A'. γ is constituted by a signature and an effect specification.

The *action signature* is constituted by a name and a list of individual *input parameters*. Such parameters need to be instantiated with individuals for the execution of the action. Given a substitution θ for the input parameters, we denote by $\gamma\theta$ the instantiated action with the *actual* parameters coming from θ. [5] The *effect specification* consists of a set $\{e_1, \ldots, e_n\}$ of effects, which take place simultaneously. An *effect* e_i has the form $[q_i^+] \wedge Q_i^- \rightsquigarrow A_i'$, where

- q_i^+ is an UCQ, and Q_i^- is an arbitrary ECQ whose free variables occur all among the free variables of q_i^+;[6]
- A_i' is a set of facts (over the alphabet of T) which include as terms: individuals in A_0, free variables of q_i^+, and Skolem terms $f(\vec{x})$ having as arguments free variables $\vec{x}$ of q_i^+.

Given the current ABox A of $\mathcal{K}$ and a substitution θ for the parameters of the action γ, the new state A' resulting from firing the action γ with parameters θ on the state A, is computed as follows: *(i)* each effect $e_i \in \gamma$ extracts from A the set $\textsc{ans}(([q_i^+] \wedge Q_i^-)\theta, T, A)$ of tuples of terms in $\textsc{adom}(A)$, and for each such tuple σ asserts a set $A_i'\theta\sigma$ of facts obtained from $A_i'\theta$ by applying the substitution σ for the free variables of q_i^+. For each Skolem term $f(\vec{x})\theta$ appearing in $A_i'\theta$, a new ground term is introduced having the form $f(\vec{x})\theta\sigma$. These terms represent new "constants" denoting "unknown" individuals. We denote by $e_i\theta(A)$ the overall set of facts, i.e., $e_i\theta(A) = \bigcup_{\sigma\in\textsc{ans}(([q_i^+]\wedge Q_i^-)\theta,T,A)} A_i'\theta\sigma$. *(ii)* Moreover, let $\textsc{eq}(A) = \{t_1 = t_2 \mid \langle t_1, t_2\rangle \in \textsc{ans}([x_1 = x_2], T, A)\}$. Observe that, due to the semantics of queries, the terms in $\textsc{eq}(A)$ must appear explicitly in $\textsc{adom}(A)$, that is, the possibly infinite number of equalities due to congruence do not appear in $\textsc{eq}(A)$, though they are logically implied. The overall *effect* of the action γ with parameter substitution θ over A is the new ABox $A' = \textsc{do}(T, A, \gamma\theta)$ where $\textsc{do}(T, A, \gamma\theta) = \textsc{eq}(A) \cup \bigcup_{1\le i\le n} e_i\theta(A)$.

Let us make some observations on such actions. The effects of an action are a form of update of the previous state [7], and not of belief revision [16]. That is, we never learn new facts on the state in which an action is executed, but only on the state resulting from the action execution. Skolem terms introduced by actions effects can be though of witnesses of new information coming from an external user/environment when executing the action. Their presence makes the domain of the ABoxes obtained by executing actions continuously changing. For simplicity, we do not make any persistence (or frame) assumption in our formalization (except for equality) [26]. In principle at every move we substitute the whole old state, i.e., ABox, with a new one. On the other hand, it should be clear that we can easily write effect specifications that *copy* big chunks of the old state into the new one. For example, $P(x, y) \rightsquigarrow P(x, y)$ copies the entire set of assertions involving the role P. We do have a persistence assumption on equalities, we implicitly copy all equalities holding in the current state to the new one. This implies that, as the system evolves, we acquire new information on equalities between terms, but never lose equality information already acquired.

Process. The process component of a KAB is a possibly nondeterministic program that uses the KAB ABoxes to store its (intermediate and final) computation results, and the actions in Γ as atomic instructions. The ABoxes can be arbitrarily queried through the KAB TBox T, while they can be updated only through actions in Γ. There are many ways to specify processes. We adopt a rule-based specification.

A *process* is a finite set Π of condition/action rules. A *condition/action rule* $\pi \in \Pi$ is an expression of the form $Q \mapsto \gamma$, where γ is an action in Γ and Q is an ECQ over T, whose free variables are exactly the parameters of γ. The rule expresses that, for each tuple θ for which condition Q holds, the action γ with actual parameters θ *can* be executed. Processes do not force the execution of actions but constrain them: the user of the process will be able to choose any action that the rules forming the process allow. Moreover, our processes inherit entirely their states from the KAB knowledge component (TBox and ABox), see e.g., [12]. Other choices are also possible: the process could maintain its own state besides the one of the KAB. As long as such an additional state is finite, or embeddable into the KAB itself, the results here would easily extend to such a case.

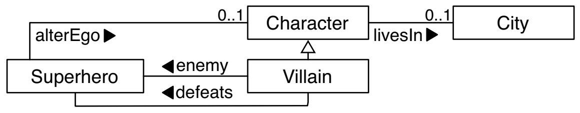

Figure 1. KAB's TBox for Example 1

Example 1 Let us consider a KAB $\mathcal{K} = (T, A_0, \Gamma, \Pi)$ describing a super-heroes comics world, where we have cities in which characters live. Figure 1 shows a UML representation of the TBox T (see [7] for the correspondence between *DL-Lite*$_\mathcal{A}$ and UML). Characters can be superheroes or (super)villains, who fight each other. As in the most classic paradigm, superheroes help the endeavors of law enforcement fighting villains threatening the city they live in. In fact, as a villain reveals himself for perpetrating his nefarious purposes against the city's peace, he consequently becomes a declared enemy of all superheroes living in that city. Each character can live in one city at the time. A common trait of almost all superheroes is a secret identity: a superhero is said to be the alter ego of some character, which is his identity in common life. Hence, the ABox assertion $\mathsf{alterEgo}(s, p)$ means that the superhero s is the alter ego of character p. Villains always try to unmask superheroes, i.e., find their secret identity, in order to exploit such a knowledge to defeat them. Notice the subtle difference here: we use an $\mathsf{alterEgo}(s, p)$ assertion to model the fact that s is the alter ego of p, whereas only by asserting $s = p$ we can capture the knowledge that s and p actually semantically denote the same individual. Γ includes the following actions:

$$\begin{array}{l}\mathsf{BecomeSH}(p, c):\\ \quad\{\ [\mathsf{Character}(p) \wedge \exists v.\mathsf{Villain}(v) \wedge \mathsf{livesIn}(v, c)]\\ \qquad \rightsquigarrow \{\mathsf{Superhero}(f(p)), \mathsf{alterEgo}(f(p), p)\},\quad \mathit{CopyAll}\ \}\end{array}$$

states that if there exists at least one villain living in the city c, a new superhero $f(p)$ is created, with the purpose of protecting c. Such a superhero has p as alter ego. *CopyAll* is a shortcut for copying everything in the new state.

$$\mathsf{Unmask}(s, p): \{\ [\mathsf{alterEgo}(p, s)] \rightsquigarrow \{s = p\},\quad \mathit{CopyAll}\ \}$$

states that superhero s, who is the alter ego of p, gets unmasked by asserting the equality between s and p (it is now known that $s = p$).

$$\begin{array}{l}\mathsf{Fight}(v, s):\\ \quad\{\ \exists p.[\mathsf{Villain}(v) \wedge \mathsf{Character}(p) \wedge \mathsf{alterEgo}(s, p)] \wedge [s = p]\\ \qquad \rightsquigarrow \{\mathsf{defeats}(v, s)\},\quad \mathit{CopyAll}\ \}\end{array}$$

states that when villain v fights superhero s, he defeats s if s has been

[5] We disregard a specific treatment of the output to the user, and assume instead that she can freely pose queries over the KB, extracting implicit or explicit information from the states through which the KAB evolves.

[6] The UCQ-ECQ division is a convenience to have readily available the positive part of the condition.

[7] Our approach sidesteps the semantical and computational difficulties of description logic knowledge base update [20]. Adopting such forms of update in our setting is an interesting research issue.

unmasked, i.e., it is known that s is equal to his alter ego.

$\mathsf{Challenge}(v,s)$:
$\{\ [\mathsf{Villain}(v) \land \mathsf{Superhero}(s) \land \exists p.\mathsf{alterEgo}(s,p) \land \mathsf{livesIn}(p,sc)] \land \neg[\mathsf{defeats}(v,s)] \rightsquigarrow \{\mathsf{livesIn}(v,sc), \mathsf{enemy}(v,s)\},\ \ \mathit{CopyAll}\ \}$

states that when villain v challenges superhero s and has not defeated him, next he lives in the same city as s and is enemy of s.

$\mathsf{ThreatenCity}(v,c)$:
$\{\ \ [\mathsf{Villain}(v) \land \mathsf{Superhero}(s) \land \exists p.\mathsf{alterEgo}(s,p) \land \mathsf{livesIn}(p,c)] \rightsquigarrow \{\mathsf{enemy}(v,s) \land \mathsf{livesIn}(v,c)\},\ \ \mathit{CopyAllExceptEnemy},$
$[\mathsf{Villain}(v) \land \mathsf{enemy}(v',s')] \land \neg[v = v'] \rightsquigarrow \mathsf{enemy}(v',s')\ \ \}$

states that when villain v threatens city c, then he becomes an enemy of all and only superheroes that live in c.

Consider an initial ABox $A_0 = \{\mathsf{Superhero}(\mathsf{batman}), \mathsf{Villain}(\mathsf{joker}), \mathsf{alterEgo}(\mathsf{batman},\mathsf{bruce}), \mathsf{livesIn}(\mathsf{bruce},\mathsf{gotham}), \mathsf{livesIn}(\mathsf{batman},\mathsf{gotham}), \mathsf{livesIn}(\mathsf{joker},\mathsf{city1})\}$. In this state, bruce and batman live in the same city, and batman is the alterego of bruce but it is not known whether they denote the same individual. Executing $\mathsf{Challenge}(\mathsf{joker},\mathsf{batman})$ in A_0 generates a new ABox with added facts $\mathsf{enemy}(\mathsf{joker},\ \mathsf{batman})$, $\mathsf{livesIn}(\mathsf{joker},\ \mathsf{gotham})$, and $\mathsf{gotham} = \mathsf{city1}$ is implied by the functionality on $\mathsf{livesIn}$.

A process Π might include the following rules:

$[\mathsf{Character}(p)] \land \neg[\mathsf{Superhero}(p)] \land [\mathsf{livesIn}(p,c)] \ \mapsto\ \mathsf{BecomeSH}(p,c),$
$[\mathsf{Superhero}(s) \land \mathsf{Character}(c)] \ \mapsto\ \mathsf{Unmask}(s,c),$
$[\mathsf{enemy}(v,s)] \ \mapsto\ \mathsf{Fight}(v,s),$
$[\mathsf{Villain}(v) \land \mathsf{Superhero}(s)] \ \mapsto\ \mathsf{Challenge}(v,s),$
$[\mathsf{Villain}(v) \land \mathsf{City}(c)] \land \neg\exists v'([\mathsf{Villain}(v') \land \mathsf{livesIn}(v',c)] \land \neg[v = v']) \ \mapsto\ \mathsf{ThreatenCity}(v,c)$

For instance, the first one states that a character can become a superhero if it is not already one. ■

4 KAB Semantics

The semantics of KABs is given in terms of possibly infinite transition systems that represents the possible evolutions of the KAB over time as actions are executed according to the process. Notice that such transition systems must be equipped with semantically rich states, since a full KB is associated to them. Technically, a *transition system* Υ is a tuple of the form $(\mathbb{U}, T, \Sigma, s_0, \mathit{abox}, \Rightarrow)$, where: $\mathbb{U}$ is a countably infinite set of terms denoting individuals; T is a TBox; Σ is a set of states; $s_0 \in \Sigma$ is the initial state; abox is a function that, given a state $s \in \Sigma$, returns an ABox associated to s, which has as individuals terms of $\mathbb{U}$ and conforms to T; $\Rightarrow \subseteq \Sigma \times \Sigma$ is a transition relation between pairs of states.

Transition system generated by a KAB. Given a KAB $\mathcal{K} = (T, A_0, \Gamma, \Pi)$, we define its *(generated) transition system* $\Upsilon_\mathcal{K} = (\mathbb{U}, T, \Sigma, s_0, \mathit{abox}, \Rightarrow)$ as follows:

- $\mathbb{U}$ is formed by all constants and all Skolem terms inductively formed starting from $\textsc{adom}(A_0)$ by applying the Skolem functions occurring in the actions in Γ;
- T is the TBox of the KAB;
- abox is the identity function (i.e., each state is simply an ABox);
- $s_0 = A_0$ is the initial state;
- Σ and $\Rightarrow$ are defined by mutual induction as the smallest sets satisfying the following property: if $s \in \Sigma$ then for each rule $Q \mapsto \gamma$, evaluate Q, and for each tuple θ returned, if $\textsc{do}(T, \mathit{abox}(s), \gamma\theta)$ is consistent w.r.t. T, then $s' = \textsc{do}(T, \mathit{abox}(s), \gamma\theta)$ and $s \Rightarrow s'$.

Notice that $\Upsilon_\mathcal{K}$ is an infinite tree in general and, in fact, it is enough to perform infinitely a single action to obtain an infinite tree. Hence the classical results on model checking [11], which are developed for finite transition systems, cannot be applied directly.

5 Verification

To specify dynamic properties over a semantic artifacts, we use a first-order variant of μ-calculus [27, 24]. (Temporal) μ-calculus is virtually the most powerful temporal logic used for model checking of finite-state transition systems, and is able to express both linear time logics such as LTL and PSL, and branching time logics such as CTL and CTL* [11]. The main characteristic of μ-calculus is its ability of expressing directly least and greatest fixpoints of (predicate-transformer) operators formed using formulae relating the current state to the next one. By using such fixpoint constructs one can easily express sophisticated properties defined by induction or co-induction. This is the reason why virtually all logics used in verification can be considered as fragments of μ-calculus. Technically, μ-calculus separates local properties, asserted on the current state or on states that are immediate successors of the current one, and properties talking about states that are arbitrarily far away from the current one [27]. The latter are expressed through the use of fixpoints.

In this work, we use a first-order variant of μ-calculus, where we allow local properties to be expressed as EQL queries, and at the same time we allow arbitrary first-order quantification across states. Given the nature of EQL queries used for formulating local properties, first-order quantification must range over terms denoting individuals. Formally, we introduce the logic $\mu\mathcal{L}_A$ defined as follows:

$$\Phi \ \longrightarrow\ Q \ \mid\ \neg\Phi \ \mid\ \Phi_1 \land \Phi_2 \ \mid\ \exists x.\Phi \ \mid\ \langle - \rangle\Phi \ \mid\ Z \ \mid\ \mu Z.\Phi$$

where Q is a possibly open EQL query, and Z is a second order predicate variable (of arity 0). We make use of the following abbreviations: $\forall x.\Phi = \neg(\exists x.\neg\Phi)$, $\Phi_1 \lor \Phi_2 = \neg(\neg\Phi_1 \land \neg\Phi_2)$, $[-]\Phi = \neg\langle - \rangle\neg\Phi$, and $\nu Z.\Phi = \neg\mu Z.\neg\Phi[Z/\neg Z]$. The formulae $\mu Z.\Phi$ and $\nu Z.\Phi$ respectively denote the least and greatest fixpoint of the formula Φ (seen as the predicate transformer $\lambda Z.\Phi$). As usual in μ-calculus, formulae of the form $\mu Z.\Phi$ (and $\nu Z.\Phi$) must obey to the *syntactic monotonicity* of Φ wrt Z, which states that every occurrence of the variable Z in Φ must be within the scope of an even number of negation symbols. This ensures that the least fixpoint $\mu Z.\Phi$ (as well as the greatest fixpoint $\nu Z.\Phi$) always exists.

The semantics of $\mu\mathcal{L}_A$ formulae is defined over possibly infinite transition systems $\langle \mathbb{U}, T, \Sigma, s_0, \mathit{abox}, \Rightarrow \rangle$. Since $\mu\mathcal{L}_A$ also contains formulae with both individual and predicate free variables, given a transition system Υ, we introduce an individual variable valuation v, i.e., a mapping from individual variables x to $\mathbb{U}$, and a predicate variable valuation V, i.e., a mapping from the predicate variables Z to a subset of Σ. With these three notions in place, we assign meaning to formulae by associating to Υ, v, and V an *extension function* $(\cdot)^\Upsilon_{v,V}$, which maps formulae to subsets of Σ. Formally, the extension function $(\cdot)^\Upsilon_{v,V}$ is defined inductively as follows:

$$\begin{aligned}
(Q)^\Upsilon_{v,V} &= \{s \in \Sigma \mid \textsc{ans}(Qv, T, \mathit{abox}(s)) = \mathit{true}\}\\
(\neg\Phi)^\Upsilon_{v,V} &= \Sigma \setminus (\Phi)^\Upsilon_{v,V}\\
(\Phi_1 \land \Phi_2)^\Upsilon_{v,V} &= (\Phi_1)^\Upsilon_{v,V} \cap (\Phi_2)^\Upsilon_{v,V}\\
(\exists x.\Phi)^\Upsilon_{v,V} &= \{s \in \Sigma \mid \exists t.t \in \textsc{adom}(\mathit{abox}(s)) \text{ and } s \in (\Phi)^\Upsilon_{v[x/t],V}\}\\
(\langle - \rangle\Phi)^\Upsilon_{v,V} &= \{s \in \Sigma \mid \exists s'.s \Rightarrow s' \text{ and } s' \in (\Phi)^\Upsilon_{v,V}\}\\
(Z)^\Upsilon_{v,V} &= V(Z)\\
(\mu Z.\Phi)^\Upsilon_{v,V} &= \bigcap\{\mathcal{E} \subseteq \Sigma \mid (\Phi)^\Upsilon_{v,V[Z/\mathcal{E}]} \subseteq \mathcal{E}\}
\end{aligned}$$

Here Qv stands for the query obtained from Q by substituting its free variables according to v.[8] Intuitively, $(\cdot)^{\Upsilon}_{v,V}$ assigns to such constructs the following meaning:

- The boolean connectives have the expected meaning.
- The quantification of individuals is done over the objects of the "current" ABox.
- The extension of $\langle - \rangle \Phi$ consists of the states s such that for *some* state s' with $s \Rightarrow s'$, we have that Φ holds in s', while the extension of $[-]\Phi$ consists of the states s such that for *all* states s' with $s \Rightarrow s'$, Φ holds in s'.
- The extension of $\mu X.\Phi$ is the *smallest subset* $\mathcal{E}_\mu$ of Σ such that, assigning to Z the extension $\mathcal{E}_\mu$, the resulting extension of Φ is contained in $\mathcal{E}_\mu$. That is, the extension of $\mu X.\Phi$ is the *least fixpoint* of the operator $(\Phi)^{\Upsilon}_{v,V[Z/\mathcal{E}]}$, where $V[Z/\mathcal{E}]$ denotes the predicate valuation obtained from V by forcing the valuation of Z to be $\mathcal{E}$.
- Similarly, the extension of $\nu X.\Phi$ is the *greatest subset* $\mathcal{E}_\nu$ of Σ such that, assigning to X the extension $\mathcal{E}_\nu$, the resulting extension of Φ contains $\mathcal{E}_\nu$. That is, the extension of $\nu X.\Phi$ is the *greatest fixpoint* of the operator $(\Phi)^{\Upsilon}_{v,V[X/\mathcal{E}]}$. Formally, $(\nu Z.\Phi)^{\Upsilon}_{v,V} = \bigcup\{\mathcal{E} \subseteq \Sigma \mid \mathcal{E} \subseteq (\Phi)^{\Upsilon}_{v,V[Z/\mathcal{E}]}\}$.

Example 2 An example of $\mu\mathcal{L}_A$ formula is:

$$\nu X.(\forall x.[\mathsf{Superhero}(x)] \supset \mu Y.([\mathsf{alterEgo}(x,x)] \vee \langle - \rangle Y)) \wedge [-]X$$

It states that, along every path, it is always true, for each superhero, that there exists an evolution that eventually leads to unmask him. ■

When Φ is a closed formula, $(\Phi)^{\Upsilon}_{v,V}$ does not depend on v or V, and we denote the extension of Φ simply by $(\Phi)^{\Upsilon}$. A closed formula Φ holds in a state $s \in \Sigma$ if $s \in (\Phi)^{\Upsilon}$. In this case, we write $\Upsilon, s \models \Phi$. A closed formula Φ holds in Υ, denoted by $\Upsilon \models \Phi$, if $\Upsilon, s_0 \models \Phi$. We call *model checking* verifying whether $\Upsilon \models \Phi$ holds.

Two transitions systems are *behaviourally equivalent* if they satisfy exactly the same $\mu\mathcal{L}_A$ formulas. To formally capture such an equivalence, we make use of the notion of *bisimulation* [23], suitably extended to deal with query answering over KBs.

For transition systems $\Upsilon_1 = \langle \mathbb{U}, T_1, \Sigma_1, s_{01}, abox_1, \Rightarrow_1 \rangle$ and $\Upsilon_2 = \langle \mathbb{U}, T_2, \Sigma_2, s_{02}, abox_2, \Rightarrow_2 \rangle$, a *bisimulation* between Υ_1 and Υ_2 is a relation $\mathcal{B} \subseteq \Sigma_1 \times \Sigma_2$ such that: $(s_1, s_2) \in \mathcal{B}$ implies that:

1. $(T_1, abox(s_1))$ and $(T_2, abox(s_2))$ are logically equivalent, i.e., for each ABox assertion $\alpha_1 \in abox(s_1)$ we have that $(T_2, abox(s_2)) \models \alpha_1$, and for each ABox assertion $\alpha_2 \in abox(s_2)$ we have that $(T_1, abox(s_1)) \models \alpha_2$;
2. if $s_1 \Rightarrow_1 s'_1$ then there exists s'_2 such that $s_2 \Rightarrow_2 s'_2$ and $(s'_1, s'_2) \in \mathcal{B}$;
3. if $s_2 \Rightarrow_2 s'_2$ then there exists s'_1 such that $s_1 \Rightarrow_1 s'_1$ and $(s'_1, s'_2) \in \mathcal{B}$.

We say that two states s_1 and s_2 are *bisimilar*, if there exists a bisimulation $\mathcal{B}$ such that $(s_1, s_2) \in \mathcal{B}$. Two transition systems Υ_1 with initial state s_{01} and Υ_2 with initial state s_{02} are *bisimilar* if $(s_{01}, s_{02}) \in \mathcal{B}$. The following theorem states that the formula evaluation in $\mu\mathcal{L}_A$ is indeed invariant wrt bisimulation, so we can equivalently check any bisimilar transition systems.

Theorem 3 *Let Υ_1 and Υ_2 be two bisimilar transition systems. Then, for two states s_1 of Υ_1 and s_2 of Υ_2 (including the initial ones) that are bisimilar, for all closed $\mu\mathcal{L}_A$ formulas Φ, we have that $s_1 \in (\Phi)^{\Upsilon_1}$ iff $s_2 \in (\Phi)^{\Upsilon_2}$.*

Proof. The proof is analogous to the standard proof of bisimulation invariance of μ-calculus [27], though taking into account our bisimulation, which guarantees that ECQs are evaluated identically over bisimilar states. □

We observe that in this invariance result, we make no use of the restriction on existentially quantified individuals belonging to the "current" active domain. However, we enforce this restriction to be able to talk only about individuals that are explicitly mentioned in the ABoxes of the transition system (i.e., in the active domain of the transition system) and not those that are implicitly present because of the congruence of equalities.

Making use of such a notion of bisimulation, we can redefine the transition system generated by KAB $\mathcal{K} = (T, A_0, \Gamma, \Pi)$ while maintaining bisimilarity, by modifying the definition of $\Upsilon_\mathcal{K} = \langle \mathbb{U}, T, \Sigma, s_0, abox, \Rightarrow \rangle$ given in Section 4 as follows. *(i)* We modify $\textsc{do}()$ so that no Skolem term t' is introduced in the generated ABox A' if in the current ABox[9] A there already is a term t such that $(T, A) \models t = t'$. *(ii)* If the ABox $A' = \textsc{do}(T, abox(s), \gamma\theta)$ obtained from the current state s is logically equivalent to the ABox $abox(s'')$, for some already generate state s'', we do not generate a new state, but simply add $s \Rightarrow s''$ to $\Upsilon_\mathcal{K}$.

6 Weakly Acyclic KABs

Verification of KABs is undecidable in general. Indeed, we have:

Theorem 4 *Verification of CTL reachability-like formulas of the form $\mu Z.(N(a) \vee \langle - \rangle Z)$ (with N an atomic concept and a an individual occurring in A_0) on KABs with empty TBoxes and actions that make use only of UCQs (no negation nor equality) is undecidable.*

Proof. By reduction to answering boolean UCQs in a relational database under a set of tuple-generating dependencies (TGDs), which is undecidable [15]. □

Hence it is of interest to isolate interesting special cases in which verification is decidable. Next, we introduce a notable class of KABs that enjoys such a property. In particular, we show that a suitable syntactic restriction, which resembles the notion of *weak acyclicity* in data exchange [15][10], guarantees boundedness of ABoxes generated by the execution of the nKAB, and in turn decidability of verification.

To do so, we introduce the edge-labeled directed *dependency graph* of a KAB $\mathcal{K} = (T, A_0, \Gamma, \Pi)$, defined as follows. *Nodes*, called *positions*, are obtained from the TBox T: there is a node for every concept name N in T, and two nodes for every role name P in T, corresponding to the domain and to the range of P. *Edges* are drawn by considering every effect specification $[q^+] \wedge Q^- \rightsquigarrow A'$ of each action contained in Γ, tracing how values are copied or contribute to generate new values as the system progresses. In particular, let p be a position corresponding to a concept/role component in the rewriting $rew(q^+)$ of q^+ with variable x. For every position p' in A' with the same variable x, we include a *normal edge* $p \rightarrow p'$. For every position p'' in A' with a Skolem term $f(\vec{t})$ such that $x \in \vec{t}$, we include a *special edge* $p \xrightarrow{*} p''$. We say that $\mathcal{K}$ is *weakly-acyclic* if its dependency graph has no cycle going through a special edge.

[8] Notice that it is built in the semantics of EQL queries that if v substitutes some free variable with an element of $\mathbb{U}$ not occurring in $abox(s)$, then $\textsc{ans}(Qv, T, abox(s)) = false$ (cf. Sec. 2). However, this does not happen for the KAB generated transition system by construction, due to the preservation of equality between domain individuals, which is reflexive.

[9] Note that all terms that are present in the current ABox are preserved in the new ABox, together with equalities between terms.

[10] We use the original definition of weak acyclicity. However, our results depend only on the ability of finding a finite bound for the chase. So, other variants of weak acyclicity, such as [21, 22], can also be adopted.

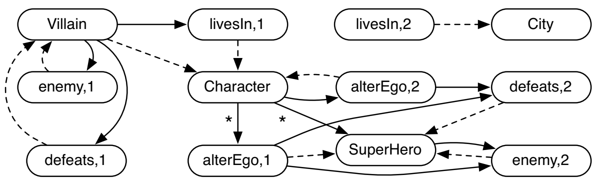

Figure 2. Weakly acyclic dependency graph for Example 1.

Theorem 5 *Verification of $\mu\mathcal{L}_A$ properties for a weakly acyclic KAB is decidable in* ExpTime *in combined complexity.*

Proof. We can define a simplified KAB, such that the size of its ABoxes bounds the size of those of the original KAB, and then relate the execution of such simplified KAB to the chase of a set of TGDs. If the original KAB is weakly acyclic, so is such a set of TGDs, hence we can apply the bound in [15]. Since all ABoxes are bounded, this implies that we can generate a *finite-state* transition system which is bisimilar to $\Upsilon_\mathcal{K}$, and do verification there. The number of states of $\Upsilon_\mathcal{K}$ is at most exponential in the size of the KAB. Hence the result follows, considering the model checking algorithm for μ-calculus on finite-state transition systems [27, 11]. □

Example 3 The KAB of Example 1 is weakly acyclic. Its dependency graph, shown in Figure 2, does not contain any cycle going through special edges. For readability, self-loops are not shown in the Figure (but are present for all nodes), and dashed edges are used to compactly represent the contributions given by the rewriting of the queries. E.g., the dashed edge form Villain to Character denotes that for every outgoing edge from Character, there exists an outgoing edge from Villain with the same type and target. Hence, w.r.t. weak acyclicity dashed edges can be simply replaced by normal edges. ■

The restriction imposed by weak acyclicity (or variants) is not too severe, and in many real cases KABs are indeed weakly acyclic or can be transformed into weakly acyclic ones at cost of redesign. Indeed, if a KAB is not weakly acyclic, it indefinitely generates new values from the old ones, which then depend on a chain of unboundedly many previous values. In other words, current values depend on an unbounded number of old values that are arbitrarily far in the past. If this is not the case, then the KAB can in principle be rewritten into a weakly acyclic one. While such unbounded systems exist in theory, e.g., Turing machines, higher level processes, as those in business process management or service-oriented modeling, do not typically require such an unboundedness in practice. How to systematically transform systems into weakly acyclic ones remains an open issue.

7 Conclusions

In this paper we have studied verification of knowledge and action bases, which are dynamic systems constituted by a knowledge base expressed in description logic, and by an action specification that changes the knowledge base over time. We have obtained an interesting decidability result by relying on the notion of weak acyclicity, based on a connection with the theory of chase of TGDs in relational databases. With this at hand, it becomes of interest to study refined action specifications that guarantee better elaboration tolerance (addressing the frame, ramification, and qualification problems) [26].

Acknowledgements. This research has been partially supported by the ICT Collaborative Project ACSI (Artifact-Centric Service Interoperation), funded by the EU under FP7 ICT Call 5, 2009.1.2, grant agreement No. FP7-257593.

REFERENCES

[1] A. Artale, D. Calvanese, R. Kontchakov, and M. Zakharyaschev, 'The *DL-Lite* family and relations', *J. of Artificial Intelligence Research*, **36**, 1–69, (2009).
[2] *The Description Logic Handbook: Theory, Implementation and Applications*, eds., F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and Peter F. Patel-Schneider, Cambridge University Press, 2003.
[3] F. Baader, S. Ghilardi, and C. Lutz, 'LTL over description logic axioms', *ACM Trans. on Computational Logic*, **13**(3), (2012).
[4] B. Bagheri-Hariri, D. Calvanese, G. De Giacomo, R. De Masellis, and P. Felli, 'Foundations of relational artifacts verification', in *Proc. of BPM*, volume 6896 of *LNCS*, pp. 379–395. Springer, (2011).
[5] F. Belardinelli, A. Lomuscio, and F. Patrizi, 'Verification of deployed artifact systems via data abstraction', in *Proc. of ICSOC*, (2011).
[6] O. Burkart, D. Caucal, F. Moller, and B. Steffen, 'Verification of infinite structures.', in *Handbook of Process Algebra*. Elsevier Science, (2001).
[7] D. Calvanese, G. De Giacomo, Domenico L., M. Lenzerini, A. Poggi, M. Rodríguez-Muro, and R. Rosati, 'Ontologies and databases: The *DL-Lite* approach', in *5th Int. Reasoning Web Summer School*, volume 5689 of *LNCS*, 255–356, Springer, (2009).
[8] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati, 'EQL-Lite: Effective first-order query processing in description logics', in *Proc. of IJCAI*, pp. 274–279, (2007).
[9] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati, 'Tractable reasoning and efficient query answering in description logics: The *DL-Lite* family', *J. of Automated Reasoning*, **39**(3), 385–429, (2007).
[10] D. Calvanese, G. De Giacomo, M. Lenzerini, and R. 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).
[11] E. M. Clarke, O. Grumberg, and D. A. Peled, *Model checking*, The MIT Press, Cambridge, MA, USA, 1999.
[12] D. Cohn and R. Hull, 'Business artifacts: A data-centric approach to modeling business operations and processes', *IEEE Bull. on Data Engineering*, **32**(3), 3–9, (2009).
[13] E. Damaggio, A. Deutsch, and V. Vianu, 'Artifact systems with data dependencies and arithmetic', in *Proc. of ICDT*, pp. 66–77, (2011).
[14] A. Deutsch, R. Hull, F. Patrizi, and V. Vianu, 'Automatic verification of data-centric business processes', in *Proc. of ICDT*, (2009).
[15] R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa, 'Data exchange: Semantics and query answering', *Theoretical Computer Science*, **336**(1), 89–124, (2005).
[16] H. Katsuno and A. Mendelzon, 'On the difference between updating a knowledge base and revising it', in *Proc. of KR*, pp. 387–394, (1991).
[17] H. J. Levesque, 'Foundations of a functional approach to knowledge representation', *Artificial Intelligence*, **23**, 155–212, (1984).
[18] F. Lin and R. Reiter, 'State constraints revisited', *J. of Logic Programming*, **4**(5), 655–678, (1994).
[19] H. Liu, C. Lutz, M. Milicic, and F. Wolter, 'Reasoning about actions using description logics with general TBoxes', in *Proc. of JELIA*, (2006).
[20] H. Liu, C. Lutz, M. Milicic, and F. Wolter, 'Updating description logic ABoxes', in *Proc. of KR*, pp. 46–56, (2006).
[21] B. Marnette and F. Geerts, 'Static analysis of schema-mappings ensuring oblivious termination', in *Proc. of ICDT*, pp. 183–195, (2010).
[22] M. Meier, M. Schmidt, F. Wei, and G. Lausen, 'Semantic query optimization in the presence of types', in *Proc. of PODS*, (2010).
[23] R. Milner, 'An algebraic definition of simulation between programs', in *Proc. of IJCAI*, pp. 481–489, (1971).
[24] D. Park, 'Finiteness is mu-ineffable', *Theoretical Computer Science*, **3**, 173–181, (1976).
[25] A. Poggi, D. Lembo, D. Calvanese, G. De Giacomo, M. Lenzerini, and R. Rosati, 'Linking data to ontologies', *J. on Data Semantics*, (2008).
[26] R. Reiter, *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems*, The MIT Press, 2001.
[27] C. Stirling, *Modal and Temporal Properties of Processes*, Springer, 2001.
[28] H. van Ditmarsch, W. van der Hoek, and B. Kooi, *Dynamic epistemic logic*, Springer, 2007.
[29] V. Vianu, 'Automatic verification of database-driven systems: a new frontier', in *Proc. of ICDT*, pp. 1–13, (2009).
[30] F. Wolter and M. Zakharyaschev, 'Temporalizing description logic', in *Frontiers of Combining Systems*, Studies Press/Wiley, (1999).