Verification of Context-Sensitive Knowledge and Action Bases

Diego Calvanese[1], İsmail İlkan Ceylan[2], Marco Montali[1], and Ario Santoso[1]

[1] Free University of Bozen-Bolzano, Italy
lastname@inf.unibz.it
[2] Technische Universität Dresden, Germany
ceylan@tcs.inf.tu-dresden.de

Abstract. Knowledge and Action Bases (KABs) have been recently proposed as a formal framework to capture the dynamics of systems which manipulate Description Logic (DL) Knowledge Bases (KBs) through action execution. In this work, we enrich the KAB setting with contextual information, making use of different context dimensions. On the one hand, context is determined by the environment using context-changing actions that make use of the current state of the KB and the current context. On the other hand, it affects the set of TBox assertions that are relevant at each time point, and that have to be considered when processing queries posed over the KAB. Here we extend to our enriched setting the results on verification of rich temporal properties expressed in μ-calculus, which had been established for standard KABs. Specifically, we show that under a run-boundedness condition, verification stays decidable and does not incur in any additional cost in terms of worst-case complexity. We also show how to adapt syntactic conditions ensuring run-boundedness so as to account for contextual information, taking into account context-dependent activation of TBox assertions.

1 Introduction

Recent work in the areas of knowledge representation, databases, and business processes [15,26,4,10,19] has identified the need for integrating static and dynamic aspects in the design and maintenance of complex information systems. The *static* aspects are characterized on the one hand by the data manipulated by the system, and on the other hand by possibly complex domain knowledge that may vary during the evolution of the system. Instead, *dynamic* aspects are affected by the processes that operate over the system, by executing actions that manipulate the state of the system. In such a setting, in which new data may be imported into the system from the outside environment, the system becomes infinite-state in general, and the verification of temporal properties becomes more challenging: indeed, neither finite-state model checking [14] nor most of the current techniques for infinite-state model checking apply to this case.

Knowledge and action bases (KABs) [4] have been introduced recently as a mechanism for capturing systems in which knowledge, data, and processes are combined and treated as first-class citizens. In particular, KABs provide a mechanism to represent semantically rich information in terms of a description logic (DL) [1] knowledge base (KB) and a set of actions that manipulate such a KB over time. Additionally, actions allow one to import into the system fresh values from the outside, via service calls. In this setting, the problem of verification of rich temporal properties expressed over KABs in a first-order variant of the μ-calculus has been studied. Decidability has been established under the assumptions that in the properties first-order quantification across states is restricted, and that the system satisfies a so-called *run-boundedness* condition. Intuitively, these ensure that along each run the system cannot encounter (and hence manipulate) an unbounded number of distinct objects. In KABs, the intensional knowledge about the domain, expressed in terms of a DL TBox, is assumed to be fixed along the evolution of the system, i.e., independent of the actual state. However, this assumption is in general too restrictive, since specific knowledge might hold or be applicable only in specific, *context-dependent* circumstances. Ideally, one should be able to form statements that are known to be true in certain cases, but not necessarily in all.

Work on representing and formally reasoning over contexts dates back to work on generality in AI see [20]. Since then, there has been some effort in knowledge representation and in DLs to devise context-sensitive formalisms, ranging from multi-context systems [5] to many-dimensional logics [18]. An important aspect in modeling context is related to the choice of which kind of information is considered to be fixed and which context dependent. Specifically, for DLs, one can define the assertions in the TBox [2,13], the concepts [5], or both [24,18] as context-dependent. Each choice addresses different needs, and results in differences in the complexity of reasoning.

We follow here the approach of [2,13], and introduce *contextualized TBoxes*, in which each inclusion assertion is adorned with context information that determines under which circumstances the inclusion assertion is considered to hold. The relation among contexts is described by means of a lattice in [2] and by means of a directed acyclic graph in [13]. In our case, we represent context using a finite set of context dimensions, each characterized by a finite set of domain values that are organized in a tree structure. If for a context dimension d, a value v_2 is placed below v_1 in the tree (i.e., v_2 is a descendant of v_1), then the context associated to v_1 is considered to be more general than the one for v_2, and hence whenever context dimension d is in value v_2, it is also in value v_1.

Starting from this representation of contexts, we enrich KABs towards *context-sensitive KABs* (CKABs), by representing the intensional information about the domain using a contextualized TBox, in place of an ordinary one. Moreover, the action component of KABs, which specifies how the states of the system evolve, is extended in CKABs with *context changing actions*. Such actions determine values for context dimensions in the new state, based on the data and the context in the current state. In addition, also regular state-changing actions can query, besides the state, also the context, and hence be enabled or disabled according to the context. Notably, we show that verification of a very rich temporal logic, which can be used to query the system evolution, contexts, and data, is decidable for run-bounded CKABs. We also discuss how to recast the syntactic condition of *weak acyclicity* [4], which ensures run-boundedness, to the case of CKABs.

2 Preliminaries

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

For expressing knowledge bases, we use the lightweight Description Logic (DL) [1] *DL-Lite*$_\mathcal{A}$ [9,7]. The syntax for *concept* and *role* expressions in *DL-Lite*$_\mathcal{A}$ is as follows:

$$B ::= N \mid \exists R \qquad\qquad R ::= P \mid P^-$$

where N denotes a *concept name*, B a *basic concept*, P a *role name*, P^- an *inverse role*, and R a *basic role*. A *DL-Lite*$_\mathcal{A}$ *knowledge base* (KB) is a tuple $\mathcal{O} = \langle T, A\rangle$, where:

- T is a TBox, containing a finite set of assertion of the form:

$$B_1 \sqsubseteq B_2 \qquad R_1 \sqsubseteq R_2 \qquad B_1 \sqsubseteq \neg B_2 \qquad R_1 \sqsubseteq \neg R_2 \qquad (\mathsf{funct}\ R)$$

  From left to right, assertions of the first two columns respectively denote *positive inclusions* between basic concepts and basic roles; assertions of the third and fourth columns denote *negative inclusions* between basic concepts and basic roles; assertions of the last column denote *functionality* on roles.
- A is an Abox, i.e., a finite set of *ABox membership assertions* of the form $N(c_1)$ or $P(c_1, c_2)$, where c_1, c_2 denote individuals (constants).

We use the standard 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 *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., [9]). We also say that A is T*-consistent* if $\mathcal{O} = \langle T, A\rangle$ is satisfiable, i.e., admits at least one model.

Queries. We are interested to query the KB, i.e., retrieving relevant constants in the ABox based on the query. We denote with $\textsc{adom}(A)$ the *set of constants appearing in* A. A *union of conjunctive queries* (UCQ) q over a KB $\mathcal{O} = \langle T, A\rangle$ is a FOL formula of the form $\bigvee_{1\leq i\leq n} \exists \vec{y_i}.conj_i(\vec{x}, \vec{y_i})$ 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 $\textsc{adom}(A)$ or variables in $\vec{x}$ or $\vec{y_i}$, for some $1 \leq i \leq n$.

The *(certain) answers* of q over $\mathcal{O} = \langle T, A\rangle$ are defined as the set $ans(q, T, A)$ of substitutions σ which substitute the free variables of q with constants from $\textsc{adom}(A)$ such that $q\sigma$ evaluates to true in every model of $\mathcal{O} = \langle T, A\rangle$. If q has no free variables, then it is called *boolean* and its certain answers are either true or false.

We also consider an extension of UCQs, namely *EQL-Lite*(UCQ) [8] (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:

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

where q is a UCQ over T. The *certain answers* $\textsc{ans}(Q, T, A)$ *of an ECQ* Q *over* $\mathcal{O} = \langle T, A\rangle$ are obtained by first computing the certain answers over $\mathcal{O} = \langle T, A\rangle$ of each UCQs embedded in Q, then evaluating them through the first-order part of Q, and interpreting existential variables as ranging over $\text{ADOM}(A)$. As stated in [8], the reformulation algorithm for answering query q over *DL-Lite*$_{\mathcal{A}}$ KB $\mathcal{O} = \langle T, A\rangle$ which allows us to "compile away" the TBox (i.e., $ans(q, T, A) = ans(rew(q), \emptyset, A)$, where $rew(q)$ is a UCQ computed by the algorithm in [7]) can be extended to ECQs.

2.2 Knowledge and Action Bases

In the following, we make use of a countably infinite set Δ of *constants*, and a finite set $\mathcal{F}$ of *functions* representing *service calls*, which can be used to introduce fresh values from Δ into the system.

A *knowledge and action base* (KAB) is a tuple $\mathcal{K} = \langle T, A_0, \Gamma, \Pi\rangle$ where: *(i)* T is a *DL-Lite*$_{\mathcal{A}}$ TBox capturing the domain of interest, *(ii)* A_0 is the initial *DL-Lite*$_{\mathcal{A}}$ ABox, which intuitively represents the initial data of the system, *(iii)* Γ is a finite set of actions that characterize the evolution of the system, *(iv)* Π is a finite set of condition-action rules forming a process that intuitively specifies when and how an action can be executed. T and A_0 together form the *knowledge base* while Γ and Π form the *action base*.

An *action* $\alpha \in \Gamma$ represents the progression mechanism that changes the ABox in the current state and hence generates a new ABox for the successor state. Formally, an action $\alpha \in \Gamma$ is represented as $\alpha(p_1, \ldots, p_n) : \{e_1, \ldots, e_m\}$ where *(i)* α is the *action name*, *(ii)* $p_1, \ldots, p_n$ are the *input parameters*, and *(iii)* $\{e_1, \ldots, e_m\}$ is the set of *effects*. Each effect e_i is of the form $[q_i^+] \wedge Q_i^- \rightsquigarrow A_i$, where: (a) q_i^+ is an UCQ, and Q_i^- is an arbitrary ECQ whose free variables occur all among the free variables of q_i^+. (b) A_i is a set of facts (over the alphabet of T) which includes as terms: constants in $\text{ADOM}(A_0)$, input parameters, free variables of q_i^+, and Skolem terms representing service calls formed by applying a function $f \in \mathcal{F}$ to one of the previous kinds of terms. Intuitively, q_i^+, together with Q_i^- acting as a filter, selects the values that instantiate the facts listed in A_i. Collectively, the instantiated facts produced from all the effects of α constitute the newly generated ABox, once the ground service calls are substituted with corresponding results. The *process* Π is formally defined as a finite set of *condition-action rules* of the form $Q(\vec{x}) \mapsto \alpha(\vec{x})$, where: *(i)* $\alpha \in \Gamma$ is an action, and *(ii)* $Q(\vec{x})$ is an ECQ over T, which has the parameters of α as free variables $\vec{x}$, and quantified variables or values in $\text{ADOM}(A_0)$ as additional terms.

Notice that KABs are a pristine action specification framework, aimed at understanding the interaction between the static and dynamic components of systems evolving over time, towards general decidability results for verification. On top of KABs, several abstractions typical of reasoning about actions in AI can be built, see, e.g., [22].

KABs Execution Semantics. The execution semantics of a KAB is defined in terms of a possibly infinite-state transition system. Formally, given a KAB $\mathcal{K} = \langle T, A_0, \Gamma, \Pi\rangle$, we define its semantics by the *transition system* $\Upsilon_{\mathcal{K}} = \langle \Delta, T, \Sigma, s_0, abox, \Rightarrow\rangle$, 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. Intuitively, the transitions system $\Upsilon_{\mathcal{K}}$ of KAB $\mathcal{K}$ captures all possible evolutions of the system by the actions in accordance with the process rules.

During the execution, an action can issue service calls. In this paper, we assume that the semantics of service calls is *deterministic*, i.e., along a run of the system, whenever a service is called with the same input parameters, it will return the same value. To enforce this semantics, the transition system remembers the results of previous service calls in a so-called service call map that is part of the system state. Formally, a *service call map* is defined as a partial function $m : \mathbb{SC} \rightarrow \Delta$, where $\mathbb{SC}$ is the set $\{f(v_1, \ldots, v_n) \mid f/n \in \mathcal{F}$ and $\{v_1, \ldots, v_n\} \subseteq \Delta\}$ of (skolem terms representing) *service calls*. Each state $s \in \Sigma$ of the transition system $\Upsilon_\mathcal{K}$ is a tuple $\langle A, m\rangle$, where A is an ABox and m is a service call map.

The semantics of an *action execution* is as follows: Given a state $s = \langle A, m\rangle$, let $\alpha \in \Gamma$ be an action of the form $\alpha(p_1, \ldots, p_n) : \{e_1, \ldots, e_m\}$ with $e_i = [q_i^+] \wedge Q_i^- \rightsquigarrow A_i$, and let σ be a *parameter substitution* for $p_1, \ldots, p_n$ with values taken from Δ. We say that α *is executable in state* s *with parameter substitution* σ, if there exists a condition-action rule $Q(\vec{x}) \mapsto \alpha(\vec{x}) \in \Pi$ s.t. $\textsc{ans}(Q\sigma, T, A)$ is $\mathtt{true}$. The result of the application of α to an ABox A using a parameter substitution σ is captured by the following function:

$$\textsc{do}(T, A, \alpha\sigma) = \bigcup_{[q_i^+] \wedge Q_i^- \rightsquigarrow A_i \text{ in } \alpha} \quad \bigcup_{\rho \in \textsc{ans}(([q_i^+] \wedge Q_i^-)\sigma, T, A)} A_i\sigma\rho$$

Intuitively, the result of the evaluation of α is obtained by combining the contribution of each effect of α, which in turn is obtained by grounding the facts A_i in the head of the effect with all the certain answers of the query $[q_i^+] \wedge Q_i^-$ over $\langle T, A\rangle$.

The result of $\textsc{do}(T, A, \alpha\sigma)$ is in general not a proper ABox, because it could contain (ground) Skolem terms, attesting that in order to produce the ABox, some service calls have to be issued. We denote by $\textsc{calls}(\textsc{do}(T, A, \alpha\sigma))$ the set of such ground service calls, and by $\textsc{evals}(T, A, \alpha\sigma)$ the set of substitutions that replace such calls with concrete values taken from Δ. Specifically, $\textsc{evals}(T, A, \alpha\sigma)$ is defined as

$$\textsc{evals}(T, A, \alpha\sigma) = \{\theta \mid \theta : \textsc{calls}(\textsc{do}(T, A, \alpha\sigma)) \rightarrow \Delta \text{ is a total function}\}.$$

With all these notions in place, we can now recall the execution semantics of a KAB $\mathcal{K} = \langle T, A_0, \Gamma, \Pi\rangle$. To do so, we first introduce a transition relation $\textsc{exec}_\mathcal{K}$ that connects pairs of ABoxes and service call maps due to action execution. In particular, $\langle\langle A, m\rangle, \alpha\sigma, \langle A', m'\rangle\rangle \in \textsc{exec}_\mathcal{K}$ if the following holds: *(i)* α is *executable* in state $s = \langle A, m\rangle$ with parameter substitution σ; *(ii)* there exists $\theta \in \textsc{evals}(T, A, \alpha\sigma)$ s.t. θ and m "agree" on the common values in their domains (in order to realize the deterministic service call semantics); *(iii)* $A' = \textsc{do}(T, A, \alpha\sigma)\theta$; *(iv)* $m' = m \cup \theta$ (i.e., updating the history of issued service calls).

The transition system $\Upsilon_\mathcal{K}$ of $\mathcal{K}$ is then defined as $\langle \Delta, T, \Sigma, s_0, abox, \Rightarrow\rangle$ where $s_0 = \langle A_0, \emptyset\rangle$, and Σ and $\Rightarrow$ are defined by simultaneous induction as the smallest sets satisfying the following properties: *(i)* $s_0 \in \Sigma$; *(ii)* if $\langle A, m\rangle \in \Sigma$, then for all actions $\alpha \in \Gamma$, for all substitutions σ for the parameters of α and for all $\langle A', m'\rangle$ s.t. $\langle\langle A, m\rangle, \alpha\sigma, \langle A', m'\rangle\rangle \in \textsc{exec}_\mathcal{K}$ and A' is T-consistent, we have $\langle A', m'\rangle \in \Sigma$, $\langle A, m\rangle \Rightarrow \langle A', m'\rangle$. A *run* of $\Upsilon_\mathcal{K}$ is a (possibly infinite) sequence $s_0 s_1 \cdots$ of states of $\Upsilon_\mathcal{K}$ such that $s_i \Rightarrow s_{i+1}$, for all $i \geq 0$.

3 Contextualizing Knowledge Bases

Following [21], we formalize context as a mathematical object. Basically, we follow the approach in [24] of contextualizing knowledge bases by adopting the metaphor of considering context as a box [6,17]. Specifically, this means that the knowledge represented by the TBox (together with the ABox) in a certain context is affected by the values of parameters used to characterize the context itself.

Formally, to define the context, we fix a set of variables $\mathbb{C}_{dim} = \{d_1, \ldots, d_n\}$ called *context dimensions* . Each context dimension $d_i \in \mathbb{C}_{dim}$ comes with its own tree-shaped finite *value domain* $\langle Dom(d_i), \prec_{d_i}\rangle$, where $Dom(d_i)$ represents the finite set of domain values, and $\prec_{d_i}$ represents the predecessor relation forming the tree. We denote the domain value in the root of the tree with $\top_{d_i}$. Intuitively, $\top_{d_i}$ is the most general value in the tree-shaped value hierarchy of $Dom(d_i)$. We denote the fact that a context dimension d is in value v by $[d \rightsquigarrow v]$, and call this a *context dimension assignment*.

A *context* C over a set $\mathbb{C}_{dim}$ of context dimensions is defined as a set $\{[d_1 \rightsquigarrow v_1], \ldots, [d_n \rightsquigarrow v_n]\}$ of context dimension assignments such that for each context dimension $d \in \mathbb{C}_{dim}$, there exists exactly one assignment $[d \rightsquigarrow v] \in C$.

To predicate over contexts, we introduce a *context expression language* $\mathcal{L}_{cx}$ over $\mathbb{C}_{dim}$, which corresponds to propositional logic where the propositional letters are context dimension assignments over $\mathbb{C}_{dim}$. The syntax of $\mathcal{L}_{cx}$ is as follows:

$$\varphi_C \ ::= \ [d \rightsquigarrow v] \ \mid \ \varphi_C \wedge \varphi'_C \ \mid \ \neg\varphi_C$$

where $d \in \mathbb{C}_{dim}$, and $v \in Dom(d)$. We adopt the standard propositional logic semantics and the usual abbreviations. The notion of *satisfiability* and *model* are as usual. We call a formula expressed in $\mathcal{L}_{cx}$ a *context expression*.

Observe that a context $C = \{[d_1 \rightsquigarrow v_1], \ldots, [d_n \rightsquigarrow v_n]\}$, being a set of (atomic) formulas in $\mathcal{L}_{cx}$, can be considered as a propositional theory. The semantics of value domains in $\mathbb{C}_{dim}$ can also be characterized by a $\mathcal{L}_{cx}$ theory. Specifically, we define the theory $\Phi_{\mathbb{C}_{dim}}$ as the smallest set of context expressions satisfying the following conditions. For every context dimension $d \in \mathbb{C}_{dim}$, we have:

- For all values $v_1, v_2 \in Dom(d)$ s.t. $v_1 \prec_d v_2$, we have that $\Phi_{\mathbb{C}_{dim}}$ contains the expression $[d \rightsquigarrow v_1] \rightarrow [d \rightsquigarrow v_2]$. Intuitively, this states that the value v_2 is more general than v_1, and hence, whenever we have $[d \rightsquigarrow v_1]$ we can infer that $[d \rightsquigarrow v_2]$.
- For all values $v_1, v_2, v \in Dom(d)$ s.t. $v_1 \prec_d v$ and $v_2 \prec_d v$, we have that $\Phi_{\mathbb{C}_{dim}}$ contains the expression $[d \rightsquigarrow v_1] \rightarrow \neg[d \rightsquigarrow v_2]$. Intuitively, this expresses that sibling values v_1 and v_2 are disjoint.

Example 1. Consider an online retail enterprise (e.g., amazon.com) with many warehouses. A simple order processing scenario is as follows: (i) The customer submits the order. (ii) The central processing office receives the order. (iii) The *assembler* collects the ordered product. For each product that is not available in the central warehouse, the assembler makes a request to one of the warehouses having that product. (iv) The *wrapper* wraps the ordered product. (v) The *quality controller (QC)* checks the prepared order. (vi) The *delivery team* delivers the order to the delivery service. In this scenario we consider $\mathbb{C}_{dim} = \{\texttt{PP}, \texttt{S}\}$, where PP stands for *processing plan*, and S stands for *season*. $Dom(\texttt{PP}) = \{\text{WE, ME, RE, N, AP}\}$ (WE stands for *worker efficiency*, ME stands for *material efficiency*, RE stands for *resource efficiency*, N stands for *normal processing* *plan*, and AP stands for *any processing plan.*), where (i) WE $\prec_{\mathtt{PP}}$ RE, (ii) ME $\prec_{\mathtt{PP}}$ RE, (iii) RE $\prec_{\mathtt{PP}}$ AP, (iv) N $\prec_{\mathtt{PP}}$ AP, For example, WE $\prec_{\mathtt{PP}}$ RE means that *worker efficiency* is a form of *resource efficiency*. $Dom(\mathtt{S}) = \{\mathrm{WH}, \mathrm{PS}, \mathrm{LS}, \mathrm{NS}, \mathrm{AS}\}$ (WH stands for *winter holiday*, PS stands for *peak season*, LS stands for *low season*, NS stands for *normal season*, and AS stands for *any season.*), where (i) WH $\prec_{\mathtt{S}}$ PS, (ii) PS $\prec_{\mathtt{S}}$ AS, (iii) NS $\prec_{\mathtt{S}}$ AS, (iv) LS $\prec_{\mathtt{S}}$ AS.

Context-Sensitive Knowledge Bases. We define a *context-sensitive knowledge base* (CKB) $\mathcal{O}_{cx}$ over $\mathbb{C}_{dim}$ as a standard DL knowledge base in which the TBox assertions are contextualized. Formally, a *contextualized TBox* T_{cx} over $\mathbb{C}_{dim}$ is a finite set of assertions of the form $\langle t : \varphi \rangle$, where t is a TBox assertion and φ is a context expression over $\mathbb{C}_{dim}$. Intuitively, $\langle t : \varphi \rangle$ expresses that the TBox assertion t holds in all those contexts satisfying φ, taking into account the theory $\Phi_{\mathbb{C}_{dim}}$. Given a contextualized TBox T_{cx}, we denote with $\mathrm{VOC}(T_{cx})$ the set of all concept and role names appearing in T_{cx}, independently from the context.

Given a CKB $\mathcal{O}_{cx} = \langle T_{cx}, A \rangle$ and a context C, both over $\mathbb{C}_{dim}$, we define the *KB* $\mathcal{O}_{cx}$ *in context* C as the KB $\mathcal{O}^C_{cx} = \langle T^C_{cx}, A \rangle$, where $T^C_{cx} = \{t \mid \langle t : \varphi \rangle \in T_{cx}$ and $C \cup \Phi_{\mathbb{C}_{dim}} \models \varphi\}$.

Example 2. Continuing our example, in a normal situation, to guarantee a suitable service quality, *wrapper* and *assembler* must not be the *QC*. However, in the situation (context) where we have either *peak season* ($[\mathtt{S} \rightsquigarrow \mathrm{PS}]$) or the company wants to promote *worker efficiency* ($[\mathtt{PP} \rightsquigarrow \mathrm{WE}]$), the *wrapper* and the *assembler* act also as *QC*. This situation can be encoded as follows:

$$\langle \mathsf{Assembler} \sqsubseteq \neg \mathsf{QC} : [\mathtt{PP} \rightsquigarrow \mathrm{N}] \wedge [\mathtt{S} \rightsquigarrow \mathrm{NS}] \rangle \quad \langle \mathsf{Assembler} \sqsubseteq \mathsf{QC} : [\mathtt{PP} \rightsquigarrow \mathrm{WE}] \vee [\mathtt{S} \rightsquigarrow \mathrm{PS}] \rangle$$
$$\langle \mathsf{Wrapper} \sqsubseteq \neg \mathsf{QC} : [\mathtt{PP} \rightsquigarrow \mathrm{N}] \wedge [\mathtt{S} \rightsquigarrow \mathrm{NS}] \rangle \quad \langle \mathsf{Wrapper} \sqsubseteq \mathsf{QC} : [\mathtt{PP} \rightsquigarrow \mathrm{WE}] \vee [\mathtt{S} \rightsquigarrow \mathrm{PS}] \rangle$$

4 Context-Sensitive Knowledge and Action Bases

We now enhance KABs with context-related information, introducing in particular *context-sensitive knowledge and action bases* (CKABs), which consist of: *(i)* a context-sensitive knowledge base (CKB), which maintains the information of interest, *(ii)* an action base, which characterizes the system evolution, and *(iii)* context information that evolves over time, capturing changing circumstances. Differently from KABs, where the TBox is fixed a-priori and remains rigid during the evolution of the system, in CKABs the TBox changes depending on the current context. Alongside the evolution mechanism for data borrowed from KABs, CKABs include also a progression mechanism for the context itself, giving raise to a system in which data and context evolve simultaneously.

4.1 Formalization of CKABs

As for standard KABs, in addition to Δ and $\mathcal{F}$, we fix the set $\mathbb{C}_{dim} = \{d_1, \ldots, d_n\}$ of *context dimensions.* A CKAB is a tuple $\mathcal{K}_{cx} = \langle T_{cx}, A_0, \Gamma, \Pi, C_0, \Pi_C \rangle$ where:

- T_{cx} is a *DL-Lite*$_{\mathcal{A}}$ *contextualized TBox* capturing the domain of interest.
- A_0 and Γ are as in a KAB.

- $\varPi$ is a finite set of condition-action rules that extend those of KABs by including, in the precondition, a context expression. Such context expression implicitly selects those contexts in which the corresponding action can be executed. Specifically, each condition-action rule has the form $\langle Q(\vec{x}), \varphi_C \rangle \mapsto \alpha(\vec{x})$, where *(i)* $\alpha \in \varGamma$ is an action, *(ii)* $Q(\vec{x})$ is an ECQ over T_{cx} whose free variables $\vec{x}$ correspond exactly to the parameters of α, and *(iii)* φ_C is a context expression over $\mathbb{C}_{dim}$.
- C_0 is the initial context over $\mathbb{C}_{dim}$.
- $\varPi_C$ is a finite set of context-evolution rules, each of which determines the configuration of the new context depending on the current context and data. Each *context-evolution rule* has the form $\langle Q, \varphi_C \rangle \mapsto C_{new}$, where: *(i)* Q is a boolean ECQ over T_{cx}, *(ii)* φ_C is a context expression, and *(iii)* C_{new} is a finite set of context dimension assignments such that for each context dimension $d \in \mathbb{C}_{dim}$, there exists *at most one* context dimension assignment $[d \rightsquigarrow v] \in C$. If a context variable is not assigned by C_{new}, it maintains the assignment of the previous state.

Example 3. In our running example, suppose the company has *warehouses* in a remote area (*remote warehouses*), each of which is expected to guarantee a certain *time to delivery* (TTD) for products. During the *low season*, the company is free to set the TTD for all its remote warehouses, which we model as a $\mathsf{chgTTD}()$ action. The execution of this action is controlled by the condition-action rule $\langle \exists \mathsf{w}.\mathsf{RemWH}(\mathsf{w}), [\mathsf{S} \rightsquigarrow \mathrm{LS}] \rangle \mapsto \mathsf{chgTTD}()$. Assuming that the company maintains the TTD for a remote warehouse in the relation hasTTD, the $\mathsf{chgTTD}()$ action can be specified as follows:

$$\mathsf{chgTTD}() : \{\ \mathsf{RemWH}(x) \land \mathsf{hasTTD}(x,y) \rightsquigarrow \{\mathsf{RemWH}(x), \mathsf{hasTTD}(x, \mathrm{newTTD}(x,y))\}\}$$

Intuitively, the unique effect in hasTTD updates the TTD of a remote warehouse x, by issuing a service call $\mathrm{newTTD}(x,y)$, which also takes into account the current TTD y of x.

Example 4. An example of context-evolution rule is $\langle \mathrm{true}, [\mathsf{S} \rightsquigarrow \mathrm{PS}] \rangle \mapsto [\mathsf{S} \rightsquigarrow \mathrm{NS}]$. It models the transition from *peak season* to *normal season*, independently from the data.

4.2 CKAB Execution Semantics

We are interested in verifying temporal properties over the evolution of CKABs, in particular "robust" properties that the system is required to guarantee independently from context changes. Towards this goal, we define the execution semantics of CKABs in terms of a possibly infinite-state transition system that simultaneously captures all possible evolutions of the system as well as all possible context changes.

Each state in the execution of a CKAB is a tuple $\langle id, A, m, C \rangle$, where id is a state identifier, A is an ABox maintaining the current data, m is a service call map accounting for the service call results obtained so far, and C is the current context. The context univocally selects which are the axioms of the contextual TBox that currently hold, in turn determining the current KB.

Formally, given a CKAB $\mathcal{K}_{cx} = \langle T_{cx}, A_0, \varGamma, \varPi, C_0, \varPi_C \rangle$, we define its semantics in terms of a *context-sensitive transition system* $\Upsilon_{\mathcal{K}_{cx}} = \langle \varDelta, T_{cx}, \varSigma, s_0, abox, ctx, \Rightarrow \rangle$, where: *(i)* T_{cx} is a contextualized TBox; *(ii)* $\varSigma$ is a set of states; *(iii)* $s_0 \in \varSigma$ is the initial state; *(iv)* $abox$ is a function that, given a state $s \in \varSigma$, returns the ABox associated to

s; *(v)* ctx is a function that, given a state $s \in \Sigma$, returns the context associated to s; *(vi)* $\Rightarrow \subseteq \Sigma \times \Sigma$ is a transition relation between pairs of states.

Starting from the initial state s_0, $\Upsilon_{\mathcal{K}_{cx}}$ accounts for all the possible (simultaneous) data and context transitions. To single out the dynamics of the system as opposed to those of the context, the transition system is built by repeatedly alternating between system and context transitions. Technically, we revise the notion of executability for KABs by taking into account context expressions, as well as the context evolution. Given an action $\alpha \in \Gamma$, we say that α is *executable* in state s with parameter substitution σ if there exists a condition-action rule $\langle Q(\vec{x}), \varphi_C \rangle \mapsto \alpha(\vec{x})$ in Π s.t. $\vec{x}\sigma \in \textsc{ans}(Q, T_{cx}^{ctx(s)}, abox(s))$ and $ctx(s) \cup \Phi_{\mathbb{C}_{dim}} \models \varphi_C$.

We then introduce an *action transition relation* $\textsc{exec}_{\mathcal{K}_{cx}}$, where $\langle \langle A, m, C \rangle, \alpha\sigma, \langle A', m', C' \rangle \rangle \in \textsc{exec}_{\mathcal{K}_{cx}}$ if the following holds:

- Action α is *executable* in state $\langle A, m, C \rangle$ with parameter substitution σ.
- There exists $\theta \in \textsc{evals}(T_{cx}^C, A, \alpha\sigma)$ s.t. θ and m "agree" on the common values in their domains;
- $A' = \textsc{do}(T_{cx}^C, A, \alpha\sigma)\theta$;
- $m' = m \cup \theta$;
- $C' = C$, i.e., the context does not change.

Alongside the action transition relation, we also define a *context transition relation* $\textsc{cexec}_{\mathcal{K}_{cx}}$, where $\langle \langle A, m, C \rangle, \langle A', m', C' \rangle \rangle \in \textsc{cexec}_{\mathcal{K}_{cx}}$ if the following holds:

- $A' = A$, i.e., the ABox does not change;
- $m' = m$, i.e., the service call map does not change;
- there exists a context rule $\langle Q, \varphi_C \rangle \mapsto C_{new}$ in Π_C s.t.: *(i)* $\textsc{ans}(Q, T_{cx}^C, A)$ is true; *(ii)* $C \cup \Phi_{\mathbb{C}_{dim}} \models \varphi_C$; *(iii)* for every context dimension $d \in \mathbb{C}_{dim}$ s.t. $[d \rightsquigarrow v] \in C_{new}$, we have $[d \rightsquigarrow v] \in C'$; *(iv)* for every context dimension $d \in \mathbb{C}_{dim}$ s.t. $[d \rightsquigarrow v] \in C$, and there does not exist any v_2 s.t. $[d \rightsquigarrow v_2] \in C_{new}$, we have $[d \rightsquigarrow v] \in C'$.

Given these, we can now define how $\Upsilon_{\mathcal{K}_{cx}}$ is constructed, by suitably alternating the action and context transitions. In order to single out the states obtained by applying just an action transition and for which the context transition has not taken place yet, we introduce a special marker $\mathsf{State}(\mathsf{inter})$, which is an ABox assertion with a fresh concept name State and a fresh constant inter. When $\mathsf{State}(\mathsf{inter})$ is present, it means that the state has been produced by an action execution, and that the next transition will represent a context change. Such states can be considered as intermediate, in the sense that the overall change both of the ABox facts and of the context has not taken place yet.

Formally, given a CKAB $\mathcal{K}_{cx} = \langle T_{cx}, A_0, \Gamma, \Pi, C_0, \Pi_C \rangle$, the context-sensitive transition system $\Upsilon_{\mathcal{K}_{cx}} = \langle \Delta, T_{cx}, \Sigma, s_0, abox, ctx, \Rightarrow \rangle$ is defined as follows:

- $s_0 = \langle id_0, A_0, \emptyset, C_0 \rangle$;
- Σ and $\Rightarrow$ are defined by simultaneous induction as the smallest sets satisfying the following properties: *(i)* $s_0 \in \Sigma$; *(ii)* if $\langle id, A, m, C \rangle \in \Sigma$ and $\mathsf{State}(\mathsf{inter}) \notin A$, then for all actions $\alpha \in \Gamma$, for all substitutions σ for the parameters of α, and for all A', m' s.t. $\langle \langle A, m, C \rangle, \alpha\sigma, \langle A', m', C \rangle \rangle \in \textsc{exec}_{\mathcal{K}_{cx}}$, let

$$\begin{aligned} S = \{ \langle id'', A', m', C' \rangle \mid\ & id'' \text{ is a fresh identifier, and there is } \langle A', m', C \rangle \\ & \text{such that } \langle \langle A', m', C \rangle, \langle A', m', C' \rangle \rangle \in \textsc{cexec}_{\mathcal{K}_{cx}} \}. \end{aligned}$$

If for some $\langle id'', A', m', C' \rangle \in S$, we have that A' is $T_{cx}^{C'}$-consistent, then $s' \in \Sigma$ and $\langle id, A, m, C \rangle \Rightarrow s'$, where $s' = \langle id', A' \cup \{\mathsf{State(inter)}\}, m', C \rangle$ and id' is a fresh identifier. Moreover, in this case, for each $s'' = \langle id'', A', m', C' \rangle \in S$ such that A' is $T_{cx}^{C'}$-consistent, we have that $s'' \in \Sigma$ and $s' \Rightarrow s''$.

Notice that, if at some point in the above inductive construction, for no $\langle id'', A', m', C' \rangle \in S$ we have that A' is $T_{cx}^{C'}$-consistent, then neither the state s' nor any state in S becomes part of Σ.

5 Verifying Temporal Properties over CKAB

Given a CKAB $\mathcal{K}_{cx}$, we are interested in verifying whether the evolution of $\mathcal{K}_{cx}$, which is represented by $\Upsilon_{\mathcal{K}_{cx}}$, complies with some given temporal property. The challenge is that in general the transition system is infinite due to the presence of services calls, which can introduce arbitrary fresh values into the system.

5.1 Verification Formalism: Context-Sensitive FO-Variant of μ-Calculus

In order to specify temporal properties over CKABs, we use a first-order variant of μ-calculus [25,23], one of the most powerful temporal logics, which subsumes LTL, PSL, and CTL* [14]. In particular, we introduce the language $\mu\mathcal{L}_{\text{CTX}}$ of *context-sensitive temporal properties*, which is based on $\mu\mathcal{L}_{A}^{\text{EQL}}$ defined in [4]. Basically, we exploit ECQs to query the states, and support a first-order quantification across states, where the quantification ranges over the constants in the current active domain. Additionally, we augment ECQs with context expressions, which allows us to check also context information while querying states. Formally, $\mu\mathcal{L}_{\text{CTX}}$ is defined as follows:

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

where Q is a possibly open EQL query that can make use of the distinguished constants in $\textsc{adom}(A_0)$, φ_C is a context expression over $\mathcal{L}_{cx}$, and Z is a second order predicate variable (of arity 0). We adopt the usual abbreviations of FOL, and also $[-]\Phi = \neg\langle-\rangle\neg\Phi$ and $\nu Z.\Phi = \neg\mu Z.\neg\Phi[Z/\neg Z]$. Hence $\langle-\rangle\langle-\rangle\Phi = \neg[-][-]\neg\Phi$ and $[-]\langle-\rangle\Phi = \neg\langle-\rangle[-]\neg\Phi$.

Notice that $\langle-\rangle[-]\Phi$ and $[-][-]\Phi$ are used in $\mu\mathcal{L}_{\text{CTX}}$ to quantify over the successor states of the current state, obtained after a state-changing transition followed by a context-changing one. This allows one to separately control how the property quantifies over state and context changes. Furthermore, due to the fact that the diamond and box operators can be only used in pairs, the local queries that inspect the data and the context maintained by the states are never issued over intermediate states, but only over those resulting from the combination of an action and context transition.

The semantics of $\mu\mathcal{L}_{\text{CTX}}$ is defined over a transition system $\Upsilon = \langle \Delta, T_{cx}, \Sigma, s_0, abox, ctx, \Rightarrow \rangle$. Since $\mu\mathcal{L}_{\text{CTX}}$ 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 Δ, and a predicate variable valuation V, i.e., a mapping from predicate variables Z to subsets of Σ. The semantics of $\mu\mathcal{L}_{\text{CTX}}$ follows the standard μ-calculus semantics, except for the semantics of queries and of quantification. We assign meaning to $\mu\mathcal{L}_{\text{CTX}}$ formulas by associating to Υ and V an *extension function* $(\cdot)^{\Upsilon}_{v,V}$, which maps $\mu\mathcal{L}_{\text{CTX}}$ formulas to subsets of Σ. The extension function $(\cdot)^{\Upsilon}_{v,V}$ is defined inductively as follows:

$$\begin{aligned}
(Q)^{\Upsilon}_{v,V} &= \{s \in \Sigma \mid \text{ANS}(Qv, T^{C}_{cx}, abox(s)) = true\} \\
(\varphi_C)^{\Upsilon}_{v,V} &= \{s \in \Sigma \mid ctx(s) \cup \Phi_{\mathbb{C}_{dim}} \models \varphi_C\} \\
(\exists x.\Phi)^{\Upsilon}_{v,V} &= \{s \in \Sigma \mid \exists d. d \in \text{ADOM}(abox(s)) \text{ and } s \in (\Phi)^{\Upsilon}_{v[x/d],V}\} \\
(Z)^{\Upsilon}_{v,V} &= V(Z) \subseteq \Sigma \\
(\neg\Phi)^{\Upsilon}_{v,V} &= \Sigma - (\Phi)^{\Upsilon}_{v,V} \\
(\Phi_1 \vee \Phi_2)^{\Upsilon}_{v,V} &= (\Phi_1)^{\Upsilon}_{v,V} \cup (\Phi_2)^{\Upsilon}_{v,V} \\
(\langle - \rangle\Phi)^{\Upsilon}_{v,V} &= \{s \in \Sigma \mid \exists s'.\ s \Rightarrow s' \text{ and } s' \in (\Phi)^{\Upsilon}_{v,V}\} \\
(\mu Z.\Phi)^{\Upsilon}_{v,V} &= \bigcap\{\mathcal{E} \subseteq \Sigma \mid (\Phi)^{\Upsilon}_{v,V[Z/\mathcal{E}]} \subseteq \mathcal{E}\}
\end{aligned}$$

where Qv is the query obtained from Q by substituting its free variables according to v. For a closed formula Φ (for which $(\Phi)^{\Upsilon}_{v,V}$ does not depend on v or V), we denote with $(\Phi)^{\Upsilon}$ the extension of Φ in Υ, and we say that Φ holds in a state $s \in \Sigma$ if $s \in (\Phi)^{\Upsilon}$.

Model checking is the problem of checking whether $s_0 \in (\Phi)^{\Upsilon}$, denoted by $\Upsilon \models \Phi$. We are interested in *verification* of $\mu\mathcal{L}_{\text{CTX}}$ properties over CKABs, i.e., given a CKAB $\mathcal{K}_{cx}$, and a $\mu\mathcal{L}_{\text{CTX}}$ property Φ, check whether $\Upsilon_{\mathcal{K}_{cx}} \models \Phi$.

Example 5. In our running example, the property $\nu Z.(\forall x.\mathsf{CustOrder}(x) \wedge [\mathtt{S} \rightsquigarrow \mathtt{PS}] \rightarrow \mu Y.(\mathsf{Delivered}(x) \vee [-][-]Y)) \wedge [-][-]Z$ checks that every customer order placed during peak season will be eventually delivered, independently on how the context and the state evolve.

5.2 Decidability of Verification

In general, verification of temporal properties over CKABs is undecidable, even for properties as simple as reachability, which can be expressed in much weaker languages than $\mu\mathcal{L}_{\text{CTX}}$. This follows immediately from the fact that CKABs generalize KABs [4].

In order to establish decidability of verification, we need to pose restrictions on the form of CKABs. We adopt the semantic restriction of *run-boundedness* identified in [4], which intuitively imposes that along every run the number of distinct values cumulatively appearing in the ABoxes of the states in the run is bounded. Formally, given a CKAB $\mathcal{K}_{cx}$, a run $\tau = s_0 s_1 \cdots$ of $\Upsilon_{\mathcal{K}_{cx}}$ is *bounded* if there exists a finite bound b s.t. $\left|\bigcup_{s \text{ state of } \tau} \text{ADOM}(abox(s))\right| < b$. We say that $\mathcal{K}_{cx}$ is *run-bounded* if there exists a bound b s.t. every run τ in $\Upsilon_{\mathcal{K}_{cx}}$ is bounded by b. The following result shows that the decidability of verification for run-bounded KABs can be lifted to CKABs as well.

Theorem 1. *Verification of $\mu\mathcal{L}_{\text{CTX}}$ properties over run-bounded CKABs is decidable, and can be reduced to finite-state model checking.*

Proof (sketch). For a run-bounded CKAB $\mathcal{K}_{cx}$, we construct a faithful finite-state abstraction for $\Upsilon_{\mathcal{K}_{cx}}$, that is, a finite-state transition system $\theta_{\mathcal{K}_{cx}}$ s.t., for every $\mu\mathcal{L}_{\text{CTX}}$ property Φ, we have that $\theta_{\mathcal{K}_{cx}} \models \Phi$ if and only if $\Upsilon_{\mathcal{K}_{cx}} \models \Phi$.

We observe that, thanks to run-boundedness, the number of distinct states appearing along each run of $\Upsilon_{\mathcal{K}_{cx}}$ is finite. Hence, the only source of infinity present in $\Upsilon_{\mathcal{K}_{cx}}$ is due to infinite branching. A distinctive feature of CKABs is that distinct states may differ not only in the ABox, but also in the TBox. However, the possible TBoxes that can be encountered during the system evolution depend only on the contexts, and not on the data contained in the ABoxes. Since contexts are propositional, only a finite number of distinct TBoxes will appear in $\Upsilon_{\mathcal{K}_{cx}}$. This, in turn, shows that infinite branching is only caused by the possibly infinite number of distinct values returned by the service calls. Hence, the source of infinity in CKABs is analogous to that of KABs, and we can adopt the same *pruning strategy* as for KABs [12]: we have shown that two successor states whose ABoxes are isomorphic w.r.t. values not present in $\textsc{adom}(A_0)$ cannot be distinguished by $\mu\mathcal{L}_A$ formulas, and therefore it is sufficient to keep only one of them in the faithful abstraction. The claim follows since $\mu\mathcal{L}_{\text{CTX}}$ is a fragment of $\mu\mathcal{L}_A$. □

We close by observing that, due to the "alternating" nature between action and context transitions in $\Upsilon_{\mathcal{K}_{cx}}$, we can interpret $\Upsilon_{\mathcal{K}_{cx}}$ as a game structure in which the system is the "good" player and the context is the "bad" player. In this light, $\mu\mathcal{L}_{\text{CTX}}$ formulas that are in negation-normal form and only make use of temporal operators $\langle - \rangle [-]$ and $[-][-]$ can express properties that the system is required to guarantee independently on how the context evolves. Thanks to Theorem 1, and by observing that CKABs meet the so-called *genericity property* in the sense of [11], we can not only verify whether there exists a system strategy to enforce a property of this kind, but also effectively extract such strategy, following the metaphor of *synthesis via model checking*.

6 Weakly Acyclic CKABs

Even though run-boundedness guarantees decidability of $\mu\mathcal{L}_{\text{CTX}}$ verification over CKABs, it is a semantic property, which is undecidable to check [3]. To mitigate this problem, [3] provides a sufficient condition for run-boundedness. Such condition leverages on the notion of *weak-acyclicity* in data exchange [16], and is syntactically checked over a dependency graph that over-approximates the transfer of values from relation components to other relation components, according to the specification of the system actions.

Intuitively, weak-acyclicity checks for the presence of service calls that can feed themselves, either directly or indirectly, through a chain of other service calls. This cyclic dependency gives raise to runs in which infinitely many distinct service calls are issued, and possibly return infinitely many distinct values, thus making those runs unbounded.

In [4], the notion of weak-acyclicity has been suitably recast in the context of KABs, taking advantage from first-order rewritability of EQL queries over *DL-Lite* ontologies, and from the fact that KABs have a TBox that is fixed, i.e., independent of the state. The idea is to construct the dependency graph approximating the behavior of the KAB action component, by considering the contribution of the TBox.

The main difficulty in lifting weak-acyclicity to our setting, is that due to the presence of the context, the TBox changes over time. To tackle this issue, we observe that the current TBox is determined by the current context, and that each action α in a CKAB can be executed only in those contexts that match with the context expressions contained in the pre-conditions of condition-action rules having α in their head. Therefore, when analyzing the contribution of α to the dependency graph, we consider all the possible finitely many contexts in which α can be applied, and consider the application of α with all corresponding TBoxes.

Formally, given a CKAB $\mathcal{K}_{cx} = \langle T_{cx}, A_0, \mathit{\Gamma}, \mathit{\Pi}, C_0, \mathit{\Pi}_C \rangle$, we define its *dependency graph* $G = \langle V, E \rangle$ as follows.

The set V of nodes is created from the concepts and roles in $\text{VOC}(T_{cx})$, as the smallest set satisfying the following conditions: (a) for each concept N in $\text{VOC}(T_{cx})$, V contains one node $\langle N, 1 \rangle$; (b) for each role R in $\text{VOC}(T_{cx})$, V contains two nodes $\langle R, 1 \rangle$ and $\langle R, 2 \rangle$, respectively reflecting the first and second component of R.

The set E of edges is created based on the condition-action rules in $\mathit{\Pi}$ and the actions in $\mathit{\Gamma}$. Each edge represents a possible data transfer from one node (i.e., concept/role component) to another node, due to some action effect. In particular, a *normal* edge represents a value transfer, whereas a *special* edge represents that the source node is part of the input for a service call whose result is stored in the target node. Specifically, E is the smallest set satisfying the following conditions (we consider the contribution of concepts, the case of role components is analogous):

1. E contains an ordinary edge $\langle N_1, 1 \rangle \dashrightarrow \langle N_2, 1 \rangle$ if there exist *(i)* an action $\alpha \in \mathit{\Gamma}$, *(ii)* an effect $[q^+] \wedge Q^- \rightsquigarrow A'$ in α, *(iii)* a condition-action rule $\langle Q(\vec{x}), \varphi_C \rangle \mapsto \alpha(\vec{x})$, and *(iv)* a variable x, s.t. $N_1(x)$ appears in $\mathit{rew}(q^+, T_{cx}^{C_\alpha})$ (i.e., in the perfect rewriting of q^+ w.r.t. $T_{cx}^{C_\alpha}$), and $N_2(x)$ appears in A'.
2. E contains a special edge $\langle N_1, 1 \rangle \overset{*}{\dashrightarrow} \langle N_2, 1 \rangle$ if there exist *(i)* an action $\alpha \in \mathit{\Gamma}$, *(ii)* an effect $[q^+] \wedge Q^- \rightsquigarrow A'$ in α, *(iii)* a condition-action rule $\langle Q(\vec{x}), \varphi_C \rangle \mapsto \alpha(\vec{x})$, and *(iv)* a variable x, s.t. $N_1(x)$ appears in $\mathit{rew}(q^+, T_{cx}^{C_\alpha})$, and $N_2(f(\ldots, x, \ldots))$ appears in A'.

A CKAB $\mathcal{K}_{cx}$ is *weakly acyclic* if its dependency graph has no cycle going through a special edge. Such a cycle witnesses that the same service call (in)directly feeds itself. The following result shows that such "context-aware" dependency graph can be effectively used as a sufficient condition for checking whether a CKAB is run-bounded.

Theorem 2. *Given a weakly acyclic CKAB $\mathcal{K}_{cx}$, we have that $\Upsilon_{\mathcal{K}_{cx}}$ is run-bounded.*

Proof (sketch). The proof is obtained by observing that the dependency graph construction for CKABs corresponds to that of standard KABs, imagining that the context is "compiled away", and that each (contextualized) action α of the CKAB under study is translated into a set of actions $\alpha_1, \ldots, \alpha_n$, each corresponding to the execution of α in one of the possible contexts in which α can be applied. Observe that n is finite and, in the worst case, it corresponds to the overall number of contexts that can be encountered in the system. In standard KABs, the contribution of each action to the dependency graph is obtained by compiling away the TBox and by considering the rewritten queries in the action effects. Hence, there is no difference between a normal KAB and a CKAB in which each of the aforementioned α_i is rewritten using the TBox obtained from the context to which α_i corresponds. This is exactly what the dependency graph construction provided above does. We can therefore recast Theorem 6.1 in [12] to obtain the claim. $\square$

From Theorems 1 and 2, we finally obtain:

Corollary 1. *Verification of $\mu\mathcal{L}_{\text{CTX}}$ properties over weakly acyclic CKABs is decidable, and can be reduced to finite-state model checking.*

7 Conclusion

We have introduced context-sensitive KABs, which extend KABs with contextual information. In this enriched setting, we make use of context-sensitive temporal properties based on a FOL variant of μ-calculus, and establish decidability of verification for such logic over CKABs in which the data values encountered along each run are bounded.

In this work, we adopt a simplistic approach to deal with inconsistency, based on simply rejecting inconsistent states. This approach is particularly critical in the presence of contextual information, which could lead to an inconsistent state simply due to a context change. In this light, it is particularly interesting to merge the approach presented in this paper with the one in [12], where inconsistency is treated in a more sophisticated way, based on the notion of repairs.

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 DFG within the Research Training Group "RoSI" (GRK 1907).

References

1. Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Patel-Schneider, P.F. (eds.): The Description Logic Handbook: Theory, Implementation and Applications. Cambridge University Press (2003)
2. Baader, F., Knechtel, M., Peñaloza, R.: Context-dependent views to axioms and consequences of semantic web ontologies. John Wiley & Sons 12–13, 22–40 (2012)
3. Bagheri Hariri, B., Calvanese, D., De Giacomo, G., Deutsch, A., Montali, M.: 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), pp. 163–174 (2013)
4. Bagheri Hariri, B., Calvanese, D., Montali, M., De Giacomo, G., De Masellis, R., Felli, P.: Description logic knowledge and action bases. J. of Artificial Intelligence Research 46, 651–686 (2013)
5. Borgida, A., Serafini, L.: Distributed description logics: Assimilating information from peer sources. J. on Data Semantics 1, 153–184 (2003)
6. Bozzato, L., Ghidini, C., Serafini, L.: Comparing contextual and flat representations of knowledge: A concrete case about football data. In: Proc. of the 7th Int. Conf. on Knowledge Capture (K-CAP), pp. 9–16. ACM Press (2013)
7. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Poggi, A., Rodríguez-Muro, M., Rosati, R.: Ontologies and databases: The *DL-Lite* approach. In: Tessaris, S., Franconi, E., Eiter, T., Gutierrez, C., Handschuh, S., Rousset, M.-C., Schmidt, R.A. (eds.) Reasoning Web. LNCS, vol. 5689, pp. 255–356. Springer, Heidelberg (2009)
8. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Rosati, R.: EQL-Lite: Effective first-order query processing in description logics. In: Proc. of the 20th Int. Joint Conf. on Artificial Intelligence (IJCAI), pp. 274–279 (2007)

9. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Rosati, R.: Tractable reasoning and efficient query answering in description logics: The *DL-Lite* family. J. of Automated Reasoning 39(3), 385–429 (2007)
10. Calvanese, D., De Giacomo, G., Lembo, D., Montali, M., Santoso, A.: Ontology-based governance of data-aware processes. In: Krötzsch, M., Straccia, U. (eds.) RR 2012. LNCS, vol. 7497, pp. 25–41. Springer, Heidelberg (2012)
11. Calvanese, D., De Giacomo, G., Montali, M., Patrizi, F.: Verification and synthesis in description logic based dynamic systems. In: Faber, W., Lembo, D. (eds.) RR 2013. LNCS, vol. 7994, pp. 50–64. Springer, Heidelberg (2013)
12. Calvanese, D., Kharlamov, E., Montali, M., Santoso, A., Zheleznyakov, D.: Verification of inconsistency-aware knowledge and action bases. In: Proc. of the 23rd Int. Joint Conf. on Artificial Intelligence, IJCAI (2013)
13. Ceylan, İ.İ., Peñaloza, R.: The Bayesian description logic $\mathcal{BEL}$. In: Demri, S., Kapur, D., Weidenbach, C. (eds.) IJCAR 2014. LNCS, vol. 8562, pp. 480–494. Springer, Heidelberg (2014)
14. Clarke, E.M., Grumberg, O., Peled, D.A.: Model checking. The MIT Press, Cambridge (1999)
15. Deutsch, A., Hull, R., Patrizi, F., Vianu, V.: Automatic verification of data-centric business processes. In: Proc. of the 12th Int. Conf. on Database Theory (ICDT), pp. 252–267 (2009)
16. Fagin, R., Kolaitis, P.G., Miller, R.J., Popa, L.: Data exchange: Semantics and query answering. Theoretical Computer Science 336(1), 89–124 (2005)
17. Giunchiglia, F., Bouquet, P.: Introduction to contextual reasoning. an artificial intelligence perspective. In: Perspectives on Cognitive Science, pp. 138–159. NBU Press (1997)
18. Klarman, S., Gutiérrez-Basulto, V.: $\mathcal{ALC}_{\mathcal{ALC}}$: A context description logic. In: Janhunen, T., Niemelä, I. (eds.) JELIA 2010. LNCS, vol. 6341, pp. 208–220. Springer, Heidelberg (2010)
19. Limonad, L., De Leenheer, P., Linehan, M., Hull, R., Vaculín, R.: Ontology of dynamic entities. In: Atzeni, P., Cheung, D., Ram, S. (eds.) ER 2012. LNCS, vol. 7532, pp. 345–358. Springer, Heidelberg (2012)
20. McCarthy, J.: Generality in artificial intelligence. Commun. ACM 30(12), 1030–1035 (1987)
21. McCarthy, J.: Notes on formalizing context. In: Proc. of the 13th Int. Joint Conf. on Artificial Intelligence (IJCAI), pp. 555–560 (1993)
22. Montali, M., Calvanese, D., De Giacomo, G.: Verification of data-aware commitment-based multiagent systems. In: Proc. of the 13th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2014), pp. 157–164 (2014)
23. Park, D.M.R.: Finiteness is Muineffable. Theoretical Computer Science 3(2), 173–181 (1976)
24. Serafini, L., Homola, M.: Contextualized knowledge repositories for the semantic web. J. of Web Semantics 12, 64–87 (2012)
25. Stirling, C.: Modal and Temporal Properties of Processes. Springer (2001)
26. Vianu, V.: Automatic verification of database-driven systems: A new frontier. In: Proc. of the 12th Int. Conf. on Database Theory (ICDT), pp. 1–13 (2009)