Adding Context to Knowledge and Action Bases

Diego Calvanese\textsuperscript{1}, İsmail İlkan Ceylan\textsuperscript{2}, Marco Montali\textsuperscript{1}, and Ario Santoso\textsuperscript{1}

\textsuperscript{1} Free University of Bozen-Bolzano, \texttt{lastname@inf.unibz.it}
\textsuperscript{2} Technische Universität Dresden, \texttt{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 mu-calculus, which had been established for standard KABs. Specifically, we show that under a run-boundedness condition, verification stays decidable.

1 Introduction

Recent work in the areas of knowledge representation, databases, and business processes \cite{21,29,15} 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 \cite{12} nor most of the current techniques for infinite-state model checking apply to this case.

Knowledge and action bases (KABs) \cite{2} 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) 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 \(\mu\)-calculus has been studied. Decidability has been established under the assumptions that in the properties first-order quantification across states is

\* This paper is an abridged version of a paper published in the proceeding of JELIA 2014 \cite{5}.
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 [16]. Since then, there has been some effort in knowledge representation and in DLs to devise context-sensitive formalisms, ranging from multi-context systems [3] to many-dimensional logics [14]. 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 [11], the concepts [3], or both [19] as context-dependent. Each choice addresses different needs, and results in differences in the complexity of reasoning.

We follow here the approach of [11], 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 [1] and by means of a directed acyclic graph in [11]. 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.

2 Preliminaries

**DL-Lite\(_A\).** For expressing knowledge bases, we use the lightweight DL DL-Lite\(_A\) [6]. The syntax for concept and role expressions in DL-Lite\(_A\) is as follows:

\[
B ::= N \mid \exists R \\
R ::= P \mid P^\neg
\]

where \( N \) denotes a concept name, \( B \) a basic concept, \( P \) a role name, \( P^\neg \) an inverse role, and \( R \) a basic role. A DL-Lite\(_A\) knowledge base (KB) is a tuple \( O = (T, A) \), where:
We use the standard semantics of DLs based on FOL interpretations $I$ which intuitively represents the initial data of the system, where $q$ is a UCQ over $T$ such that $q$ is a UCQ over $T$. The certain answers $\text{ans}(q, T, A)$ of an ECQ $q$ over $O = \langle T, A \rangle$ are obtained by first computing the certain answers over $O = \langle T, A \rangle$ of each UCQs embedded in $q$, then evaluating them through the first-order part of $Q$ and interpreting existential variables as ranging over ADOM. As stated in [7], the reformulation algorithm for answering query $q$ over DL-Lite$_A$ KB $O = \langle T, A \rangle$ which allows us to “compile away” the TBox (i.e., $\text{ans}(q, T, A) = \text{ans}(\text{rew}(q), \emptyset, A)$, where $\text{rew}(q)$ is a UCQ computed by the algorithm in [6]) can be extended to ECQs.

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

A knowledge and action base (KAB) is a tuple $K = \langle T, A_0, \Gamma, \Pi \rangle$ where: (i) $T$ is a DL-Lite$_A$ TBox capturing the domain of interest, (ii) $A_0$ is the initial DL-Lite$_A$ ABox, which intuitively represents the initial data of the system, (iii) $\Gamma$ is a finite set of actions that characterize the evolution of the system, (iv) $\Pi$ 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 $\Gamma$ and $\Pi$ 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) $\alpha$ 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 $q_i^+ \land Q_i^- \rightarrow A_i$, 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 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 $\alpha$ constitute the newly generated ABox, once the ground service calls are substituted with corresponding results. The process $\Pi$ is formally defined as a finite set of condition-action rules of the form $Q(\vec{x}) \rightarrow \alpha(\vec{x})$, where: (i) $\alpha \in \Gamma$ is an action, and (ii) $Q(\vec{x})$ is an ECQ over $T$, which has the parameters of $\alpha$ as free variables $\vec{x}$, and quantified variables or values in ADOM($A_0$) as additional terms.

**KABs Execution Semantics.** The execution semantics of a KAB is defined in terms of a possibly infinite-state transition system. Formally, given a KAB $\mathcal{K} = (T, A_0, \Gamma, \Pi)$, we define its semantics by the transition system $\mathcal{T}_{\mathcal{K}} = (\Delta, T, \Sigma, s_0, \text{abox}, \rightarrow)$, where: (i) $T$ is a DL-Lite$_A$ TBox; (ii) $\Sigma$ is a (possibly infinite) set of states; (iii) $s_0 \in \Sigma$ is the initial state; (iv) $\text{abox}$ 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 $\mathcal{T}_{\mathcal{K}}$ 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 $m : \mathbb{S} \mathcal{C} \rightarrow \Delta$, where $\mathbb{S} \mathcal{C}$ is the set $\{f(v_1, \ldots, v_n) \mid f/n \in \mathcal{F} \text{ and } \{v_1, \ldots, v_n\} \subseteq \Delta\}$ of (skolem terms representing) service calls. Each state $s \in \Sigma$ of the transition system $\mathcal{T}_{\mathcal{K}}$ 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 $e_i = [q_i^+] \land Q_i^- \rightarrow A_i$, and let $\sigma$ be a parameter substitution for $p_1, \ldots, p_n$ with values taken from $\Delta$. We say that $\alpha$ is executable in state $s$ with parameter substitution $\sigma$, if there exists a condition-action rule $Q(\vec{x}) \rightarrow \alpha(\vec{x}) \in \Pi$ s.t. $\text{ANS}(Q\sigma, T, A)$ is true. The result of the application of $\alpha$ to an ABox $A$ using a parameter substitution $\sigma$ is captured by the following function:

\[
\text{DO}(T, A, \alpha\sigma) = \bigcup A_i \sigma \rho \quad \text{where } \rho \in \text{ANS}([q_i^+] \land Q_i^- \rightarrow A_i, \alpha, T, A)
\]
Intuitively, the result of the evaluation of $\alpha$ is obtained by combining the contribution of each effect of $\alpha$, which in turn is obtained by grounding the facts $A_1$ in the head of the effect with all the certain answers of the query $[q^+] \land Q^-$ over $\langle T, A \rangle$.

The result of $\text{DO}(T, A, \alpha\sigma)$ 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 $\text{CALLS}(\text{DO}(T, A, \alpha\sigma))$ the set of such ground service calls, and by $\text{EVALS}(T, A, \alpha\sigma)$ the set of substitutions that replace such calls with concrete values taken from $\Delta$. Specifically, $\text{EVALS}(T, A, \alpha\sigma)$ is defined as

$$\text{EVALS}(T, A, \alpha\sigma) = \{ \theta : \theta : \text{CALLS}(\text{DO}(T, A, \alpha\sigma)) \rightarrow \Delta \text{ is a total function} \}.$$ 

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

The transition system $T_K$ of $K$ is then defined as $\langle \Delta, T, \Sigma, s_0, \text{abox}, \Rightarrow \rangle$, where $s_0 = \langle A_0, \emptyset \rangle$, and $T$ and $\Sigma$ 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 $\alpha \in T$, for all substitutions $\sigma$ for the parameters of $\alpha$ and for all $\langle A', m' \rangle$ s.t. $\langle a, m, \alpha\sigma, \langle A', m' \rangle \rangle \in \text{EXEC}_K$ 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 $T_K$ is a (possibly infinite) sequence $s_0s_1\cdots$ of states of $T_K$ such that $s_i \Rightarrow s_{i+1}$, for all $i \geq 0$.

## 3 Contextualizing Knowledge Bases

Following [17], we formalize context as a mathematical object. Basically, we follow the approach in [19] of contextualizing knowledge bases by adopting the metaphor of considering context as a box [413]. 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 $C_{\text{dim}} = \{d_1, \ldots, d_n\}$ called context dimensions. Each context dimension $d_i \in C_{\text{dim}}$ comes with its own tree-shaped finite value domain $\langle \text{Dom}(d_i), \prec_d \rangle$, where $\text{Dom}(d_i)$ represents the finite set of domain values, and $\prec_d$ represents the predecessor relation forming the tree. We denote the domain value in the root of the tree with $\top_d$. Intuitively, $\top_d$ is the most general value in the tree-shaped value hierarchy of $\text{Dom}(d_i)$. We denote the fact that a context dimension $d$ is in value $v$ by $[d \sim v]$, and call this a context dimension assignment.

A context $C$ over a set $C_{\text{dim}}$ of context dimensions is defined as a set $\{[d_1 \sim v_1], \ldots, [d_n \sim v_n]\}$ of context dimension assignments such that for each context dimension $d \in C_{\text{dim}}$, there exists exactly one assignment $[d \sim v] \in C$. To predicate over contexts, we introduce a context expression language $L_{\text{cx}}$ over $C_{\text{dim}}$, which corresponds
to propositional logic where the propositional letters are context dimension assignments over $\mathcal{C}_{\text{dim}}$. The syntax of $\mathcal{L}_{\text{ctx}}$ is as follows:

$$\varphi_{\text{ctx}} := [d \sim v] \mid \varphi_{\text{ctx}} \land \varphi'_{\text{ctx}} \mid \neg \varphi_{\text{ctx}}$$

where $d \in \mathcal{C}_{\text{dim}}$, and $v \in \text{Dom}(d)$. 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 $\mathcal{L}_{\text{ctx}}$ a context expression.

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

- For all values $v_1, v_2 \in \text{Dom}(d)$ s.t. $v_1 \prec_d v_2$, we have that $\Phi_{\mathcal{C}_{\text{dim}}}$ contains the expression $[d \sim v_1] \rightarrow [d \sim v_2]$. Intuitively, this states that the value $v_2$ is more general than $v_1$, and hence, whenever we have $[d \sim v_1]$ we can infer that $[d \sim v_2]$.
- For all values $v_1, v_2, v \in \text{Dom}(d)$ s.t. $v_1 \prec_d v$ and $v_2 \prec_d v$, we have that $\Phi_{\mathcal{C}_{\text{dim}}}$ contains the expression $[d \sim v_1] \rightarrow \neg[d \sim v_2]$. 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 $\mathcal{C}_{\text{dim}} = \{\text{PP}, \text{S}\}$, where PP stands for processing plan, and S stands for season. $\text{Dom}(\text{PP}) = \{\text{WE}, \text{ME}, \text{RE}, \text{N}, \text{AP}\}$ (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 $\prec_{\text{PP}}$ RE, (ii) ME $\prec_{\text{PP}}$ RE, (iii) RE $\prec_{\text{PP}}$ AP, (iv) N $\prec_{\text{PP}}$ AP. For example, WE $\prec_{\text{PP}}$ RE means that worker efficiency is a form of resource efficiency. $\text{Dom}(\text{S}) = \{\text{WH}, \text{PS}, \text{LS}, \text{NS}, \text{AS}\}$ (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 $\prec_{\text{S}}$ PS, (ii) PS $\prec_{\text{S}}$ AS, (iii) NS $\prec_{\text{S}}$ AS, (iv) LS $\prec_{\text{S}}$ AS.

### Context-Sensitive Knowledge Bases

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

Given a CKB $\mathcal{O}_{\text{ctx}} = (T_{\text{ctx}}, A)$ and a context $C$, both over $\mathcal{C}_{\text{dim}}$, we define the KB $\mathcal{O}_{\text{ctx}}$ in context $C$ as the KB $\mathcal{O}_{\text{ctx}}^C = (T_{\text{ctx}}^C, A)$, where $T_{\text{ctx}}^C = \{t \mid (t : \varphi) \in T_{\text{ctx}} \land C \cup \Phi_{\mathcal{C}_{\text{dim}}} \models \varphi\}$.

**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
We now enhance KABs with context-related information, introducing in particular ∆. As for standard KABs, in addition to we model as a chgTTD. 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 dimension assignment d dimension assignments such that for each context dimension d ∈ Cdim, there exists at most one context dimension assignment [d ∼ v] ∈ C. If a context variable is not assigned by Cnew, it maintains the assignment of the previous state.

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 ∆ and F, we fix the set Cdim = {d1, . . . , dn} of context dimensions. A CKAB is a tuple Kcx = ⟨Tcx, A0, Γ, Π, C0, ΠC⟩ where:

- Tcx is a DL-LiteA contextualized TBox capturing the domain of interest.
- A0 and Γ are as in a KAB.
- Π 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 ⟨Q(⃗x), φC⟩ → α(⃗x), where (i) α ∈ Γ is an action, (ii) Q(⃗x) is an ECQ over Tcx whose free variables ⃗x correspond exactly to the parameters of α, and (iii) φC is a context expression over Cdim.
- C0 is the initial context over Cdim.
- ΠC 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 ⟨Q, φC⟩ → Cnew, where: (i) Q is a boolean ECQ over Tcx, (ii) φC is a context expression, and (iii) Cnew is a finite set of context dimension assignments such that for each context dimension d ∈ Cdim, there exists at most one context dimension assignment [d ∼ v] ∈ C. If a context variable is not assigned by Cnew, 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 chgTTD() action. The execution of this action is controlled by the condition-action rule ⟨∃w. RemWH(w), [S ∼ LS]⟩ → chgTTD(). Assuming that the company maintains the TTD for a remote warehouse in the relation hasTTD, the chgTTD() action can be specified as follows: chgTTD() : { RemWH(x) ∧ hasTTD(x, y) → {RemWH(x), hasTTD(x, newTTD(x, y))} }

Intuitively, the unique effect in hasTTD updates the TTD of a remote warehouse x, by issuing a service call newTTD(x, y), which also takes into account the current TTD y of x.
Example 4. An example of context-evolution rule is \( \langle \text{true}, [S \sim \text{PS}] \rangle \mapsto [S \sim \text{NS}] \). 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 \( K_{cx} = \{ T_{cx}, A_0, \Gamma, \Pi, C_0, \Pi_C \} \), we define its semantics in terms of a context-sensitive transition system \( T_{K_{cx}} = \{ \Delta, T_{cx}, \Sigma, s_0, \text{abox, ctx} \} \), where: (i) \( T_{cx} \) is a contextualized TBox; (ii) \( \Sigma \) is a set of states; (iii) \( s_0 \in \Sigma \) is the initial state; (iv) \( \text{abox} \) is a function that, given a state \( s \in \Sigma \), returns the ABox associated to \( s \); (v) \( \text{ctx} \) 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 \( s_0 \), \( T_{K_{cx}} \) 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 \( \alpha \) is executable in state \( s \) with parameter substitution \( \sigma \) if there exists a condition-action rule \( \langle Q(\vec{x}), \varphi_C \rangle \mapsto \alpha(\vec{x}) \) in \( \Pi \) s.t. \( \vec{x}\sigma \in \text{ANS}(Q, T_{cx}^{\text{ctx}}(s), \text{abox}(s)) \) and \( \text{ctx}(s) \cup \Phi_{\text{ctx}} \models \varphi_C \).

We then introduce an action transition relation \( \text{EXEC}_{K_{cx}} \), where \( \langle \langle A, m, C \rangle, \alpha\sigma, \langle A', m', C' \rangle \rangle \in \text{EXEC}_{K_{cx}} \) if the following holds:

- Action \( \alpha \) is executable in state \( \langle A, m, C \rangle \) with parameter substitution \( \sigma \);
- There exists \( \theta \in \text{EVALS}(T_{cx}^{\text{ctx}}, A, \alpha\sigma) \) s.t. \( m \) “agree” on the common values in their domains;
- \( A' = \text{DO}(T_{cx}^{\text{ctx}}, A, \alpha\sigma)\theta \);
- \( m' = m \cup \theta \);
- \( C' = C \), i.e., the context does not change.

Alongside the action transition relation, we also define a context transition relation \( \text{EXCEX}_{K_{cx}} \), where \( \langle \langle A, m, C \rangle, \langle A', m', C' \rangle \rangle \in \text{EXCEX}_{K_{cx}} \) 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 \( \langle Q, \varphi_C \rangle \mapsto C_{\text{new}} \) in \( \Pi_C \) s.t.: (i) \( \text{ANS}(Q, T_{cx}^{\text{ctx}}, A) \) is true; (ii) \( C \cup \Phi_{\text{ctx}} \models \varphi_C \); (iii) for every context dimension \( d \in C_{\text{dim}} \) s.t. \( [d \sim v] \in C_{\text{new}} \), we have \( [d \sim v] \in C' \); (iv) for every context dimension \( d \in C_{\text{dim}} \) s.t. \( [d \sim v] \notin C \), and there does not exist any \( v_2 \) s.t. \( [d \sim v_2] \in C_{\text{new}} \), we have \( [d \sim v] \notin C' \).
Given these, we can now define how $\mathcal{T}_{\text{ctx}}$ 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 State(inter), which is an ABox assertion with a fresh concept name State and a fresh constant inter. When State(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 $\mathcal{K}_{\text{ctx}} = \langle T_{\text{ctx}}, A_0, \Gamma,\Pi, C_0, \Pi_C \rangle$, the context-sensitive transition system $\mathcal{T}_{\text{ctx}} = \langle \Delta, T_{\text{ctx}}, \Sigma, s_0, \text{abox, ctx}, \Rightarrow \rangle$ is defined as follows:

- $s_0 = \langle \text{id}_0, A_0, \emptyset, C_0 \rangle$;
- $\Sigma$ and $\Rightarrow$ are defined by simultaneous induction as the smallest sets satisfying the following properties: (i) $s_0 \in \Sigma$; (ii) if $\langle \text{id}, A, m, C \rangle \in \Sigma$ and State(inter) $\notin A$, then for all actions $\alpha \in \Gamma$, for all substitutions $\sigma$ for the parameters of $\alpha$, and for all $A', m'$ s.t. $\langle A, m, C \rangle, \alpha \sigma, (A', m', C) \rangle \in \text{EXEC}_{\text{ctx}}$, let

$$S = \{ \langle \text{id}'', A', m', C' \rangle | \text{id}'' \text{ is a fresh identifier, and there is } \langle A', m', C \rangle \text{ such that } \langle \langle A', m', C \rangle, (A', m', C') \rangle \in \text{CEEXEC}_{\text{ctx}} \}.$$ 

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

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

## 5 Verifying Temporal Properties over CKAB

Given a CKAB $\mathcal{K}_{\text{ctx}}$, we are interested in verifying whether the evolution of $\mathcal{K}_{\text{ctx}}$, which is represented by $\mathcal{T}_{\text{ctx}}$, 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 $\mu$-Calculus

In order to specify temporal properties over CKABs, we use a first-order variant of $\mu$-calculus [20,18], one of the most powerful temporal logics, which subsumes LTL, PSL, and CTL* [12]. In particular, we introduce the language $\mu L_{\text{CTX}}$ of context-sensitive temporal properties, which is based on $\mu L_{\text{AO}}$ defined in [2]. 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, $\mu L_{\text{CTX}}$ is defined as follows:

$$\Phi ::= Q \mid \varphi_C \mid \neg \Phi \mid \Phi_1 \land \Phi_2 \mid \exists x.\Phi \mid \langle \rangle[\cdot]\Phi \mid \lceil\cdot\rceil\Phi \mid Z \mid \mu Z.\Phi$$
where $Q$ is a possibly open EQL query that can make use of the distinguished constants in $\text{ADOM}(A_0)$, $\varphi_C$ is a context expression over $L_{ctx}$, and $Z$ is a second order predicate variable (of arity 0). We adopt the usual abbreviations of FOL, and also $\mu Y$. Hence $\langle \cdots \rangle \Phi = \neg \langle \cdots \rangle \neg \Phi$ and $\nu Z. \Phi = \mu Z. \neg \Phi[Z/-Z]$. Hence $\langle \cdots \rangle \neg \Phi = \neg \langle \cdots \rangle \neg \Phi$ and $\langle \cdots \rangle \Phi = \neg \langle \cdots \rangle \neg \Phi$.

Notice that $\langle \cdots \rangle \neg \Phi$ and $\langle \cdots \rangle \neg \Phi$ are used in $\mu L_{ctx}$ 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 $\mu L_{ctx}$ is defined over a transition system $\mathcal{T} = (\Delta, T_{ctx}, \Sigma, s_0, abox, ctxt, \Rightarrow)$. Since $\mu L_{ctx}$ contains formulae with both individual and predicate free variables, given a transition system $\mathcal{T}$, we introduce an individual variable valuation $v$, i.e., a mapping from individual variables $x$ to $\Delta$, and a predicate variable valuation $V$, i.e., a mapping from predicate variables $Z$ to subsets of $\Sigma$. The semantics of $\mu L_{ctx}$ follows the standard $\mu$-calculus semantics, except for the semantics of queries and of quantification. We assign meaning to $\mu L_{ctx}$ formulas by associating to $\mathcal{T}$ and $V$ an extension function $(\cdot)^T_{v,V}$, which maps $\mu L_{ctx}$ formulas to subsets of $\Sigma$. The extension function $(\cdot)^T_{v,V}$ is defined inductively as follows:

\[
(Q)^T_{v,V} = \{ s \in \Sigma \mid \text{ANS}(Qv, T_{ctx}^S, abox(s)) = \text{true} \}
\]

\[
(\varphi)^T_{v,V} = \{ s \in \Sigma \mid \text{ctx}(s) \cup \varphi_{\text{dim}} \models \varphi_C \}
\]

\[
(\exists x. \Phi)^T_{v,V} = \{ s \in \Sigma \mid \exists d. d \in \text{ADOM}(abox(s)) \text{ and } s \in (\Phi)^T_{v[x/d],V} \}
\]

\[
(Z)^T_{v,V} = V(Z) \subseteq \Sigma
\]

\[
(\neg \Phi)^T_{v,V} = \Sigma - (\Phi)^T_{v,V}
\]

\[
(\Phi_1 \vee \Phi_2)^T_{v,V} = (\Phi_1)^T_{v,V} \cup (\Phi_2)^T_{v,V}
\]

\[
(\langle \cdots \rangle \Phi)^T_{v,V} = \{ s \in \Sigma \mid \exists s', s \Rightarrow s' \text{ and } s' \in (\Phi)^T_{v',V} \}
\]

\[
(\mu Z. \Phi)^T_{v,V} = \bigcap \{ E \subseteq \Sigma \mid (\Phi)^T_{v,V[Z/E]} \subseteq E \}
\]

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

**Model checking** is the problem of checking whether $s_0 \in (\Phi)^T$, denoted by $\mathcal{T} \models \Phi$.

We are interested in verification of $\mu L_{ctx}$ properties over CKABs, i.e., given a CKAB $K_{ctx}$, and a $\mu L_{ctx}$ property $\Phi$, check whether $\mathcal{T}_{K_{ctx}} \models \Phi$.

**Example 5.** In our running example, the property $\nu Z.(\forall x. \text{CustOrder}(x) \land [S \sim PS] \rightarrow \mu Y.(\text{Delivered}(x) \lor [\neg \neg Y]) \land [\neg \neg \neg Z]$ 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 $\mu L_{ctx}$. This follows immediately from the fact that CKABs generalize KABs [2].
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 [2], 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 $K_{\text{cx}}$, a run $\tau = s_0 s_1 \cdots$ of $T_{K_{\text{cx}}}$ is bounded if there exists a finite bound $b$ s.t. $|\bigcup_{s \text{ state of } \tau} \text{ADOM}(\text{abox}(s))| < b$. We say that $K_{\text{cx}}$ is run-bounded if there exists a bound $b$ s.t. every run $\tau$ in $T_{K_{\text{cx}}}$ 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 $\mu L_{\text{CTX}}$ properties over run-bounded CKABs is decidable, and can be reduced to finite-state model checking.

**Theorem 2.** Given a weakly acyclic CKAB $K_{\text{cx}}$, we have that $T_{K_{\text{cx}}}$ is run-bounded.

From Theorems 1 and 2, we finally obtain:

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

### 6 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 $\mu$-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 here with the one in [10], where inconsistency is treated in a more sophisticated way.

**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


