Optimizations for Decision Making and Planning in Description Logic Dynamic Knowledge Bases

Michele Stawowy

IMT Institute for Advanced Studies, Lucca, Italy
michele.stawowy@imtlucca.it

Abstract. Artifact-centric models for business processes recently raised a lot of attention, as they manage to combine structural (i.e. data related) with dynamical (i.e. process related) aspects in a seamless way. Many frameworks developed under this approach, although, are not built explicitly for planning, one of the most prominent operations related to business processes. In this paper, we try to overcome this by proposing a framework named Dynamic Knowledge Bases, aimed at describing rich business domains through Description Logic-based ontologies, and where a set of actions allows the system to evolve by modifying such ontologies. This framework, by offering action rewriting and knowledge partialization, represents a viable and formal environment to develop decision making and planning techniques for DL-based artifact-centric business domains.

1 Introduction

Classically, management of business processes always focused on workflows and the actions/interactions that take part in them, an approach called *process-centric*. One of the most prominent operations related to business processes is *planning* [7], namely finding a sequence of operations/actions that allows to reach a desired goal. Lately, such approach has been call into question, as the sole focus on the workflow leaves out the *informational context* in which the workflow is executed.

Artifact-centric models for business processes recently raised a lot of attention [2,6], as they manage to combine structural (i.e. data related) with dynamical (i.e. process related) aspects in a seamless way, thus overcoming the limits of process-centric approach. In this context, we can see the development of the framework called *Knowledge and Actions Bases* [9], the later higher formalization of it named *Description Logic Based Dynamic Systems* [5], and the Golog-based work of [1]. These works all share the same concept: handle the data-layer through a *Description Logic ontology*, while the process-layer, since DLs are only able to give a static representation of the domain of interest, is defined as *actions* that update the ontology (the so-called "functional view of knowledge bases" [10]). The combination of these two elements generates a transition system in which states are represented by DL knowledge bases. They do

also share a similar objective: verification of temporal formulas over the afore-mentioned transition system. Since finding a path that lead to a goal state can be expressed as a reachability temporal formula, these environments can be used for planning purposes, but they are not explicitly meant for this task. From their definition, we are limited to explore the state-space in a forward manner (we could end up having to explore the full state-space) and only by using the full body of the available knowledge, which is not ideal for developing different ways to search the state-space, as well as under a performance point of view.

In this paper we propose an artifact-centric framework, called *Dynamic Knowledge Bases*, aimed at describing data-rich business domains and be a more versatile environment for planning and decision-making: the data-layer is taken care of by a DL knowledge base, while a set of actions allows the system to evolve by adding/removing assertions, as well as introducing new instances to the system. To reach our goals, and overcome the afore-mentioned limitations, our framework relies on few optimizations. First of all, although our framework is based on Description Logic, it is desirable to skip completely the use of the TBox: this would allow us to avoid executing reasoning tasks and only work with facts from the ABox, simplifying especially the transition-building process. We fulfil this aspect with *action rewriting*, which rewrites actions and introduces a *blocking query*: such query (which is fixed for each action) tells if, given a state, we can perform the given action and built the ending state of the transition, or if the action will lead us to an inconsistent state w.r.t. the TBox. These operations are done without calculating the ending state, and without the need of the TBox (while keeping the consistency w.r.t. it).

Secondly, while the totality of the available knowledge is necessary to asses the consistency of the overall system, it bounds us to work with details that might not be of interests immediately. In decision making [8], "*an heuristic is a strategy that ignores part of the information, with the goal of making decisions more quickly, frugally, and/or accurately than more complex methods*". Being able to work with partial information is vital when we deal with systems described by complex ontologies and are composed of millions (if not more) instances. To allow our framework to be used for such strategies we introduce *partialization*, so that users can focus on a chosen subset of knowledge (partial knowledge); it allows to build a transition system which starts from a subset of the original ABox (the facts that describe the complete system), and, for each transition, choose which knowledge to transfer to the next state. Lastly, we demonstrate how, given a path found over the partial knowledge transition system, we can calculate a *global blocking query*, which tells if such path can be performed in the original transition system with no modifications.

The resulting framework constitutes a sound base on top of which researchers can develop new planning techniques useful for all those situations in which is necessary to manipulate both actions and data together (e.g. the decision making process in agents, composition of web services, etc.).

2 Dynamic Knowledge Bases

Dynamic Knowledge Bases (DKBs) are, briefly, a variation of *Knowledge and Action Bases* (KABs) [9], namely *dynamic systems* (more precisely labelled transition systems) in which states are constituted by DL *knowledge bases* (KBs), and a set of *actions* that makes the system evolve by modifying those KBs.

Definition 1. *A* DKB *is a tuple* $\mathcal{D} = (T, A_0, \Gamma)$*, where* (T, A_0) *is a DL-Lite*$_\mathcal{A}$ *KB, while* Γ *is a finite set of actions.*

We adopt a restricted version of DL-Lite$_\mathcal{A}$ knowledge bases [4], which does not use attributes (available in full DL-Lite$_\mathcal{A}$ KBs). DL-Lite$_\mathcal{A}$ employs the *Unique Name Assumption*, thus equality assertions are not allowed. We adopt DL-Lite$_\mathcal{A}$ as it is, like other DL-Lite dialects, quite expressive while maintaining decidability, good complexity results, and enjoys the FOL-rewritability property. In the followings, the set $\textsc{adom}(A)$ identifies the individual constants in the ABox A, which are defined over a countably infinite (object) universe Δ of individuals (it follows that $\textsc{adom}(A) \subseteq \Delta$). $\mathcal{A}_T$ denotes the set of all possible consistent ABoxes w.r.t. T that can be constructed using atomic concept and atomic role names in T, and individuals in Δ. The adopted semantic is the standard one based on first-order interpretations and on the notion of model: a TBox is satisfiable if admits at least one model, an ABox A is consistent w.r.t. a TBox T if (T, A) is satisfiable, and (T, A) logically implies an ABox assertion α (denoted $(T, A) \models \alpha$) if every model of (T, A) is also a model of α.

We define an *action* as:

$$\mathsf{a}\colon q, N \rightsquigarrow E$$

where a is the *action name*, q is a query called *action guard*, N is a set of variables which are used in an *instance creation function*, and E are the *action effects*.
The guard q is a standard *conjunctive query* (CQ) of the type $q = \exists \overrightarrow{y}.conj(\overrightarrow{x}, \overrightarrow{y})$, where $conj(\overrightarrow{x}, \overrightarrow{y})$ is a conjunction of atoms using free variables $\overrightarrow{x}$ and existentially quantified variables $\overrightarrow{y}$, no individuals. Atoms of q uses concepts and roles found in T. $Vars(q)$ represents the variables in q (i.e., $\overrightarrow{x} \cup \overrightarrow{y}$), while $Vars(q)_{\not\exists}$ (resp., $Vars(q)_\exists$) only the set $\overrightarrow{x}$ (resp., $\overrightarrow{y}$).
The set N contains variables which do not appear in q (i.e., $Vars(q) \cap N = \emptyset$), and which are fed to an assignment function m when the action is executed. The set E is a set of atomic effects (i.e., atomic non-grounded ABox assertions) which is divided in two subsets: the set E^- of *negative effects*, and the set E^+ of *positive effects*. All atoms of E^- must use variables that are in $Vars(q)_{\not\exists}$, while the atoms of E^+ uses variables from the set $Vars(q)_{\not\exists} \cup N$. All variables are defined over a countably infinite (object) universe V of variables.

Definition 2. *The transition system* $\Upsilon_\mathcal{D}$ *is defined as a tuple* $(\Delta, T, \Sigma, A_0, \Rightarrow)$*, where: (i)* Δ *is the universe of individual constants; (ii)* T *is a TBox; (iii)* Σ *is a set of states, namely ABoxes from the set* $\mathcal{A}_T$ *(*$\Sigma \subseteq \mathcal{A}_T$*); (iv)* A_0 *is the initial state; (v)* $\Rightarrow \ \subseteq \Sigma \times \mathcal{L} \times \Sigma$ *is a labelled transition relation between states, where* $\mathcal{L} = \Gamma \times \Theta$ *is the set of labels containing an action instantiation* $\mathsf{a}\vartheta$*, where* a *is an action from* Γ *and* ϑ *a variable assignment in* Θ *from* V *to* Δ*.*

The *transition system* $\Upsilon_{\mathcal{D}}$ represent the dynamics of a DKB $\mathcal{D}$. Given a state A and selected an action a, the informal semantic of a transition is:

1. extract the certain answers $\text{ANS}(q, T, A)$ of the guard q from the state A;
2. pick *randomly* one tuple from $\text{ANS}(q, T, A)$ and use it to initiate the variable assignment ϑ_{a} for the variables $Vars(\mathsf{a})$ (at this point we covered only the free variables in $Vars(q)_{\not\exists}$);
3. choose an assignment for the variables in N and use it to extend ϑ_{a}. We define an assignment function $m(N, A) : N \to (\Delta \setminus \text{ADOM}(A))$, which assigns to each variable of N an individual from Δ which does not appear in A;
4. use ϑ_{a} to instantiate the effects E and calculate A_{next} by applying the instantiated effects to A.

The sets Σ and $\Rightarrow$ are thus mutually defined using induction (starting from A_0) as the smallest sets satisfying the following property: for every $A \in \Sigma$ and action $\mathsf{a} \in \Gamma$, if exists an action instantiation $\mathsf{a}\vartheta_{\mathsf{a}}$ s.t.

$$A_{next} = A \setminus sub(ent(E^-, T)\vartheta_{\mathsf{a}}, A) \cup E^+\vartheta_{\mathsf{a}}$$

and $A_{next} \in \mathcal{A}_T$, then $A_{next} \in \Sigma$ and $A \overset{l}{\Rightarrow} A_{next}$, with $l = \mathsf{a}\vartheta_{\mathsf{a}}$. $\mathsf{a}\vartheta_{\mathsf{a}}$ is called an *instantiation* of a.

$ent(E^-, T)$ represents a set of atoms derived from E^-, which represents all the atoms which entail one or more single negative effects e^- in E^- w.r.t. to the TBox T. We take each single negative effect e^- and, by considering e^- as a CQ composed only by one atom, obtain an UCQ $rew_T(e^-)$ by using the *query reformulation algorithm* [3, Chapter 5.2]. Since we consider a single atom at a time, the algorithm produces an UCQ composed only by CQs with a single atom e^-_{rew} in them. Each atom e^-_{rew} either contains variables found in e^- or, in case of a role term, one of the two variables can be a non-distinguished non-shared variable represented by the symbol '_' (never both variables). We add each atom e^-_{rew} to the set $ent(E^-, T)$. Given $ent(E^-, T)$, we calculate the set $sub(ent(E^-, T)\vartheta_{\mathsf{a}}, A)$ in the following way. For each atom e^-_{rew} in $ent(E^-, T)$, we apply the variable transformation ϑ_{a} to it (the symbol '_' remains untouched, as it is not linked to any variable that appears in ϑ_{a}); we then check if it exists in the ABox A an assertion α such that $e^-_{rew}\vartheta_{\mathsf{a}} = \alpha$, assuming that the symbol '_' can be evaluated equal to any individual ($\_ = ind\ , \forall ind \in \text{ADOM}(A)$).

For clarity, from now on we will denote the set $sub(ent(E^-, T)\vartheta_{\mathsf{a}}, A)$ with $E^-_{sub(\vartheta_{\mathsf{a}})}$. Notice that the set $E^-_{sub(\vartheta_{\mathsf{a}})}$ is not uniquely determined, as it depends on the ABox on which it is applied. This behaviour is intentional, as our aim is to have the certainty that an assertion e^- marked for removal will not appear in the next state nor in the ABox A_{next}, nor as an inferable assertion ($\langle T, A_{next}\rangle \not\models e^-$); to reach such goal, we have to remove all possible assertions that entail e^-. The set $ent(E^-, T)$, instead, depends only on E^- and T, thus it's constant and can be calculated only one time at the beginning.

As we see from the definition of A_{next}, actions modify only ABox assertions: it follows that the TBox is fixed, while the ABox changes as the system evolves (thus an ABox A_i is sufficient to identify the state i of the system). The transition system $\Upsilon_{\mathcal{D}}$ clearly can be infinite, as we have the possibility to introduce new

constants. We call a *path* π a (possibly infinite) sequence of transitions over $\Upsilon_{\mathcal{D}}$ that start from A_0 ($\pi = A_0 \overset{\mathsf{a}_1\vartheta_1}{\Rightarrow} \ldots \overset{\mathsf{a}_n\vartheta_n}{\Rightarrow} A_n$).

Example 1. Consider the DKB $\mathcal{D}$ described by the following elements and which models a simple business scenario:

- the TBox $T = \{\mathsf{Employee} \sqsubseteq \neg\mathsf{Product}, \mathsf{Technician} \sqsubseteq \mathsf{Employee}\}$;
- the ABox $A_0 = \{\mathsf{Technician(t1)}, \mathsf{Product(p1)}\}$;
- the action set Γ composed of the following actions:
  create: $\{\mathsf{Employee}(x)\}, \{y\} \rightsquigarrow \{\mathsf{Product}(y)\}^+$
  fire: $\{\mathsf{Employee}(x)\} \rightsquigarrow \{\mathsf{Employee}(x)\}^-$

If we consider A_0 as the initial state in $\Upsilon_{\mathcal{D}}$, then a possible transition is $A_0 \overset{\mathsf{create}\vartheta}{\Rightarrow} A_1$ where: $\vartheta = \{x \mapsto \mathsf{t1}, y \mapsto \mathsf{p2}\}$ (notice that we introduce a new individual p2), and $A_1 = \{\mathsf{Technician(t1)}, \mathsf{Product(p1)}, \mathsf{Product(p2)}\}$.

We could also perform the action fire, as it exists a proper instantiation of it by using the variable assignment $\vartheta_{\mathsf{fire}} = \{x \mapsto \mathsf{t1}\}$. The set $ent(E^-, T)$ for the action fire corresponds to the set $\{\mathsf{Employee}(x), \mathsf{Technician}(x)\}$, thus $E^-_{sub(\vartheta_{\mathsf{fire}})}$ would be equal to $\{\mathsf{Technician(t1)}\}$. Performing the action instantiation would get us to the state $A_2\{\mathsf{Product(p1)}\}$, and it's clear that $\langle T, A_2\rangle \not\models \mathsf{Employee(t1)}$. If we would simply remove the instantiated negative effects in $E^-\vartheta_{\mathsf{fire}}$), we wouldn't achieve the same result (as the assertion Technician(t1) would still appear in the final state), as if the action didn't have any effect at all.

3 Optimizations

3.1 Action Rewriting

The first optimization we bring to the framework regards actions, and, more specifically, the guard q. Using the *query reformulation algorithm* [3, Chapter 5.2], we can transform a query q into an UCQ $rew_T(q)$ such that $\textsc{ans}(q, T, a) = \textsc{ans}(rew_T(q), \emptyset, A)$. We then take every action a, calculate $rew_T(q)$, and, for every CQ $q^{rew} \in rew_T(q)$, create an action $\mathsf{a}^{\mathsf{rew}}$: $q^{rew}, N \rightsquigarrow E$ (with N and E taken from a without modifications). These new actions slightly modify the transition function $\Rightarrow$: the guard is now evaluated without using the TBox, and the variable assignment $\vartheta_{\mathsf{a}^{\mathsf{rew}}}$ must be taken from the certain answers $\textsc{ans}(q^{rew}, \emptyset, A)$, while the rest of the transition function remains the same.

The second optimization regards the ending state of the transition: in the specification of a DKB, actions could lead to inconsistent states. We introduce an additional element called *blocking query* B, a boolean UCQ used as a block test in the state A before performing the action: if B returns *false*, then we can perform the action and have the guarantee that the ending state A_{next} is consistent w.r.t. T. The building of B is based on the *NI-closure of* T (denoted $cln(T)$) defined in [3]. Each positive effect $e^+ \in E^+$ (column 1 in Table 1, we need to change the variables accordingly to the ones in e^+) could take part in a negative inclusion assertion $\alpha \in cln(T)$ (column 2 in Table 1); this mean that we have to look for a possible assertion β (column 3 in Table 1) which

could break α when e^+ is added ($\mathbf{z}$ represents a newly introduced variable, thus $\mathbf{z} \notin Vars(q) \cup N \cup Vars(B)$). To do so, for each possible β we get from e^+ and α, we perform the following steps (we start from $B = \bot$, where $\bot$ indicates a predicate whose evaluation is *false* in every interpretation):

1. we check if β is present in the positive effects E^+ by executing $\text{ANS}(\beta, \emptyset, E^+)$ and retrieve all the certain answers ϕ_{E^+}. For each ϕ_{E^+}, it means it exist an assertion $\beta\phi_{E^+}$ which poses a problem. Since we are dealing with variables (the effects are not instantiated yet), we have to express in B under which conditions $\beta\phi_{E^+}$ would make A_{next} inconsistent; we do this by adding the corresponding CQ β_{E^+} (column 4 in Table 1) to B by *or*-connecting it to the rest of the CQs.
   Notice that we treat $\mathbf{z}$ as an existential variable, as it does not appear in e^+ and thus we have no constrains about it.
2. we check if in E^- there are negative effects that could block β by removing it (thus eliminating the threat of an inconsistency). by executing $\text{ANS}(\beta, \emptyset, ent(E^-, T))$ and retrieve all the certain answers ϑ_{E^-}. For each ϑ_{E^-}, it means it exist an assertion $\beta\phi_{E^-}$ which is removed. Since we are dealing with variables (the effects are not instantiated yet), we have to express in B under which conditions $\beta\phi_{E^-}$ can't block an inconsistency in A_{next}; we do this by adding the corresponding UCQ β_{E^-} (column 5 in Table 1) to B by *or*-connecting it to the rest of the CQs.
3. if E^- can't block any inconsistency (thus $\text{ANS}(\beta, \emptyset, ent(E^-, T)) = \emptyset$), then we have to express in B under which conditions there will be a inconsistency in A_{next} due to an assertion β in A w.r.t e^+; we do so by adding β_A (column 6 in Table 1) to B by *or*-connecting it to the rest of the CQs.

Note that while building the blocking query B, we could have, for the UCQs β_{E^-}, inequalities of the type $x \neq \_$, with $\_$ the non-distinguished non-shared variable generated by $ent(E^-, T)$. Such inequalities always evaluate to *False*.

Definition 3. *Given an action* $\mathsf{a} \in \Gamma$*, its* rewritten action $\mathsf{a}^{\mathsf{rew}}$ *is defined as:*

$$\mathsf{a}^{\mathsf{rew}}\text{: } q^{rew}, N, B \rightsquigarrow E$$

where $q^{rew} \in rew_T(q)$*, and* B *is the* blocking query *of* $\mathsf{a}^{\mathsf{rew}}$*.*

The union of all possible rewritten actions defines the set of actions Γ^{rew}.

Example 2. Let's consider the action create: $\{\mathsf{Employee}(x)\}, \{y\} \rightsquigarrow \{\mathsf{Product}(y)\}^+$. First we calculate $rew_T(q)$, which is the UCQ $\mathsf{Employee}(x) \vee \mathsf{Technician}(x)$.
We can now calculate the blocking query B. We see that the concept term Product of the positive effect $e^+ = \mathsf{Product}(y)$ takes part in the negative-inclusion assertion $\mathsf{Employee} \sqsubseteq \neg\mathsf{Product}$, and, by the definition of $cln(T)$, also in the assertion $\mathsf{Technician} \sqsubseteq \neg\mathsf{Product}$: we thus have two β assertions, $\mathsf{Employee}(y)$, and $\mathsf{Technician}(y)$. By following the procedure for building B, we have no β_{E^+} elements (as $\text{ANS}(\beta, \emptyset, E^+) = \emptyset$, and no β_{E^-} elements (as $\text{ANS}(\beta, \emptyset, ent(E^-, T)) = \emptyset$). The final query is thus composed only of β_A elements, and is

$$B = \mathsf{Employee}(y) \vee \mathsf{Technician}(y)$$

e^+	α	β	$\vartheta_{E^+} \to \beta_{E^+}$	$\vartheta_{E^-} \to \beta_{E^-}$	β_A
$\mathsf{A}(x)$	$\mathsf{A} \sqsubseteq \neg \mathsf{A}_1$ $\mathsf{A}_1 \sqsubseteq \neg \mathsf{A}$	$\mathsf{A}_1(x)$	$\{x \mapsto y\} \to x = y$	$\{x \mapsto y\} \to x \neq y \wedge \mathsf{A}_1(x)$	$\mathsf{A}_1(x)$
$\mathsf{A}(x)$	$\mathsf{A} \sqsubseteq \neg \exists \mathsf{P}$ $\exists \mathsf{P} \sqsubseteq \neg \mathsf{A}$	$\mathsf{P}(x, \mathbf{z})$	$\{x \mapsto y\} \to x = y$	$\{x \mapsto y_1, \mathbf{z} \mapsto y_2\} \to$ $\exists \mathbf{z}.\mathsf{P}(x, \mathbf{z}) \wedge x \neq y_1 \wedge \mathbf{z} \neq y_2$	$\exists \mathbf{z}.\mathsf{P}(x, \mathbf{z})$
$\mathsf{A}(x)$	$\mathsf{A} \sqsubseteq \neg \exists \mathsf{P}^-$ $\exists \mathsf{P}^- \sqsubseteq \neg \mathsf{A}$	$\mathsf{P}(\mathbf{z}, x)$	$\{x \mapsto y\} \to x = y$	$\{x \mapsto y_1, \mathbf{z} \mapsto y_2\} \to$ $\exists \mathbf{z}.\mathsf{P}(\mathbf{z}, x) \wedge x \neq y_1 \wedge \mathbf{z} \neq y_2$	$\exists \mathbf{z}.\mathsf{P}(\mathbf{z}, x)$
$\mathsf{P}(x_1, x_2)$	$\exists \mathsf{P} \sqsubseteq \neg \mathsf{A}$ $\mathsf{A} \sqsubseteq \neg \exists \mathsf{P}$	$\mathsf{A}(x_1)$	$\{x_1 \mapsto y\} \to x_1 = y$	$\{x_1 \mapsto y\} \to x_1 \neq y \wedge \mathsf{A}(x_1)$	$\mathsf{A}(x_1)$
$\mathsf{P}(x_1, x_2)$	$\exists \mathsf{P}^- \sqsubseteq \neg \mathsf{A}$ $\mathsf{A} \sqsubseteq \neg \exists \mathsf{P}^-$	$\mathsf{A}(x_2)$	$\{x_2 \mapsto y\} \to x_2 = y$	$\{x_2 \mapsto y\} \to x_2 \neq y \wedge \mathsf{A}(x_2)$	$\mathsf{A}(x_2)$
$\mathsf{P}(x_1, x_2)$	$\exists \mathsf{P} \sqsubseteq \neg \exists \mathsf{P}_1$ $\exists \mathsf{P}_1 \sqsubseteq \neg \exists \mathsf{P}$	$\mathsf{P}_1(x_1, \mathbf{z})$	$\{x_1 \mapsto y\} \to x_1 = y$	$\{x_1 \mapsto y_1, \mathbf{z} \mapsto y_2\} \to$ $\exists \mathbf{z}.\mathsf{P}(x_1, \mathbf{z}) \wedge x_1 \neq y_1 \wedge \mathbf{z} \neq y_2$	$\exists \mathbf{z}.\mathsf{P}_1(x_1, \mathbf{z})$
$\mathsf{P}(x_1, x_2)$	$\exists \mathsf{P}^- \sqsubseteq \neg \exists {\mathsf{P}_1}^-$ $\exists {\mathsf{P}_1}^- \sqsubseteq \neg \exists \mathsf{P}^-$	$\mathsf{P}_1(\mathbf{z}, x_2)$	$\{x_2 \mapsto y\} \to x_2 = y$	$\{x_2 \mapsto y_1, \mathbf{z} \mapsto y_2\} \to$ $\exists \mathbf{z}.\mathsf{P}(\mathbf{z}, x_2) \wedge x_2 \neq y_1 \wedge \mathbf{z} \neq y_2$	$\exists \mathbf{z}.\mathsf{P}_1(\mathbf{z}, x_2)$
$\mathsf{P}(x_1, x_2)$	$\exists \mathsf{P} \sqsubseteq \neg \exists {\mathsf{P}_1}^-$ $\exists {\mathsf{P}_1}^- \sqsubseteq \neg \exists \mathsf{P}$	$\mathsf{P}_1(\mathbf{z}, x_1)$	$\{x_1 \mapsto y\} \to x_1 = y$	$\{x_1 \mapsto y_1, \mathbf{z} \mapsto y_2\} \to$ $\exists \mathbf{z}.\mathsf{P}(\mathbf{z}, x_1) \wedge x_1 \neq y_1 \wedge \mathbf{z} \neq y_2$	$\exists \mathbf{z}.\mathsf{P}_1(\mathbf{z}, x_1)$
$\mathsf{P}(x_1, x_2)$	$\exists \mathsf{P}^- \sqsubseteq \neg \exists \mathsf{P}_1$ $\exists \mathsf{P}_1 \sqsubseteq \neg \exists \mathsf{P}^-$	$\mathsf{P}_1(x_2, \mathbf{z})$	$\{x_2 \mapsto y\} \to x_2 = y$	$\{x_2 \mapsto y_1, \mathbf{z} \mapsto y_2\} \to$ $\exists \mathbf{z}.\mathsf{P}(x_2, \mathbf{z}) \wedge x_2 \neq y_1 \wedge \mathbf{z} \neq y_2$	$\exists \mathbf{z}.\mathsf{P}_1(x_2, \mathbf{z})$
$\mathsf{P}(x_1, x_2)$	$\mathsf{P} \sqsubseteq \neg \mathsf{P}_1$ $\mathsf{P}_1 \sqsubseteq \neg \mathsf{P}$ $\mathsf{P}^- \sqsubseteq \neg {\mathsf{P}_1}^-$ ${\mathsf{P}_1}^- \sqsubseteq \neg \mathsf{P}^-$	$\mathsf{P}_1(x_1, x_2)$	$\{x_1 \mapsto y_1, x_2 \mapsto y_2\}$ $\to x_1 = y_1 \wedge x_2 = y_2$	$\{x_1 \mapsto y_1, x_2 \mapsto y_2\} \to$ $(x_1 \neq y_1 \wedge \mathsf{P}(x_1, x_2)) \vee$ $(x_2 \neq y_2 \wedge \mathsf{P}(x_1, x_2))$	$\mathsf{P}_1(x_1, x_2)$
$\mathsf{P}(x_1, x_2)$	$\mathsf{P} \sqsubseteq \neg {\mathsf{P}_1}^-$ ${\mathsf{P}_1}^- \sqsubseteq \neg \mathsf{P}$ $\mathsf{P}^- \sqsubseteq \neg \mathsf{P}_1$ $\mathsf{P}_1 \sqsubseteq \neg \mathsf{P}^-$	$\mathsf{P}_1(x_2, x_1)$	$\{x_1 \mapsto y_1, x_2 \mapsto y_2\}$ $\to x_1 = y_1 \wedge x_2 = y_2$	$\{x_1 \mapsto y_1, x_2 \mapsto y_2\} \to$ $(x_1 \neq y_1 \wedge \mathsf{P}(x_2, x_1)) \vee$ $(x_2 \neq y_2 \wedge \mathsf{P}(x_2, x_1))$	$\mathsf{P}_1(x_2, x_1)$
$\mathsf{P}(x_1, x_2)$	funct P	$\mathsf{P}(x_1, \mathbf{z})$ $\wedge\ x_2 \neq \mathbf{z}$	$\{x_1 \mapsto y_1, x_2 \mapsto y_2\}$ $\to x_1 = y_1 \wedge x_2 \neq y_2$	$\{x_1 \mapsto y_1, \mathbf{z} \mapsto y_2\} \to$ $(\exists \mathbf{z}.\mathsf{P}(x_1, \mathbf{z}) \wedge x_1 \neq y_1 \wedge \mathbf{z} \neq x_2) \vee$ $(\exists \mathbf{z}.\mathsf{P}(x_1, \mathbf{z}) \wedge x_1 = y_1 \wedge \mathbf{z} \neq x_2 \wedge \mathbf{z} \neq y_2)$	$\exists \mathbf{z}.\mathsf{P}(x_1, \mathbf{z})$ $\wedge\ x_2 \neq \mathbf{z}$
$\mathsf{P}(x_1, x_2)$	funct P^-	$\mathsf{P}(\mathbf{z}, x_2)$ $\wedge\ x_1 \neq \mathbf{z}$	$\{x_1 \mapsto y_1, x_2 \mapsto y_2\}$ $\to x_2 = y_2 \wedge x_1 \neq y_1$	$\{\mathbf{z} \mapsto y_1, x_2 \mapsto y_2\} \to$ $(\exists \mathbf{z}.\mathsf{P}(\mathbf{z}, x_2) \wedge x_2 \neq y_2 \wedge \mathbf{z} \neq x_1) \vee$ $(\exists \mathbf{z}.\mathsf{P}(\mathbf{z}, x_2) \wedge x_2 = y_2 \wedge \mathbf{z} \neq x_1 \wedge \mathbf{z} \neq y_1)$	$\exists \mathbf{z}.\mathsf{P}(\mathbf{z}, x_2)$ $\wedge\ x_1 \neq \mathbf{z}$

Table 1: Assertions β_{E^+}, β_{E^-}, and β_A for a given positive effect e^+ and assertion α

We get the following two rewritten actions:
$\mathsf{create}_1^{\mathsf{rew}}$: $\{\mathsf{Employee}(x)\}, \{y\}, \{\mathsf{Employee}(y) \vee \mathsf{Technician}(y)\} \rightsquigarrow \{\mathsf{Product}(y)\}^+$
$\mathsf{create}_2^{\mathsf{rew}}$: $\{\mathsf{Technician}(x)\}, \{y\}, \{\mathsf{Employee}(y) \vee \mathsf{Technician}(y)\} \rightsquigarrow \{\mathsf{Product}(y)\}^+$

Theorem 1. *Given a satisfiable KB (T, A), an action $\mathsf{a}^{\mathsf{rew}} \in \Gamma^{rew}$ such that $\vartheta_{\mathsf{a}^{\mathsf{rew}}} \in \text{ANS}(q^{rew}, \emptyset, A)$ and $\text{ANS}(B\vartheta_{\mathsf{a}^{\mathsf{rew}}}, \emptyset, A) = \emptyset$, then the ABox $A_{next} = A \setminus E^-_{sub(\vartheta_{\mathsf{a}^{\mathsf{rew}}})} \cup E^+\vartheta_{\mathsf{a}^{\mathsf{rew}}}$ is consistent w.r.t. T.*

Proof. For the proof of the theorem we remind the reader to the Appendix of the extended version of this paper [11].

Lemma 1. *Given an action $\mathsf{a}^{\mathsf{rew}} \in \Gamma^{rew}$, for every ABox A such that $\vartheta_{\mathsf{a}^{\mathsf{rew}}} \in \text{ANS}(q^{rew}, \emptyset, A)$ and $\text{ANS}(B\vartheta_{\mathsf{a}^{\mathsf{rew}}}, \emptyset, A) = \emptyset$, we can always perform the transition $A \overset{\mathsf{a}^{\mathsf{rew}}\vartheta_{\mathsf{a}^{\mathsf{rew}}}}{\Rightarrow} A_{next}$, with $A_{next} \in \mathcal{A}_T$.*

Thanks to the rewriting of actions, we can build the transition system $\Upsilon_{\mathcal{D}}$ without the need of the TBox T, while still having the guarantee that the system is consistent w.r.t. it.

3.2 Partial Transition System

We now build a *partialization* $\Upsilon^p_{\mathcal{D}}$ of the transition system $\Upsilon_{\mathcal{D}}$, which is built in the same way as $\Upsilon_{\mathcal{D}}$, apart from two points: i) the initial state is a subset of the ABox A_0 ii) it uses a looser transition function.

Definition 4. *A* partial transition system $\Upsilon^p_{\mathcal{D}}$ *is a tuple $(\Delta, T, \Sigma^p, A^p_0, \rightarrow)$, where: (i) Δ is the universe of individual constants; (ii) T is a TBox; (iii) Σ^p is a set of states, namely ABoxes from the set $\mathcal{A}_T$ ($\Sigma^p \subseteq \mathcal{A}_T$); (iv) A^p_0 is a subset of the initial ABox A_0 ($A^p_0 \subseteq A_0$); (v) $\rightarrow \ \subseteq \Sigma^p \times \mathcal{L} \times \Sigma^p$ is a labelled transition relation between states, where $\mathcal{L} = \Gamma^{rew} \times \Theta$ is the set of labels containing an action instantiation $\mathsf{a}^{\mathsf{rew}}\vartheta$, where $\mathsf{a}^{\mathsf{rew}}$ is an action from Γ^{rew} and ϑ a variable assignment in Θ from V to Δ.*

As $A^p_0 \subseteq A_0$, we have the guarantee that $A^p_0 \in \mathcal{A}_T$. The sets Σ^p and $\rightarrow$ are mutually defined using induction (starting from A^p_0) as the smallest sets satisfying the following property: for every $A^p \in \Sigma^p$ and action $\mathsf{a}^{\mathsf{rew}} \in \Gamma^{rew}$, if exists an action instantiation $\mathsf{a}^{\mathsf{rew}}\vartheta_{\mathsf{a}^{\mathsf{rew}}}$ s.t.

$$A^p_{next} \subseteq A^p \setminus E^-_{sub(\vartheta_{\mathsf{a}^{\mathsf{rew}}})} \cup E^+\vartheta_{\mathsf{a}^{\mathsf{rew}}}$$

and $A^p_{next} \in \mathcal{A}_T$, then $A^p_{next} \in \Sigma^p$ and $A^p \overset{l}{\Rightarrow} A^p_{next}$, with $l = \mathsf{a}^{\mathsf{rew}}\vartheta_{\mathsf{a}^{\mathsf{rew}}}$.

Notice that A^p_{next} can be any subset of $A^p \setminus E^-_{sub(\vartheta_{\mathsf{a}^{\mathsf{rew}}})} \cup E^+\vartheta_{\mathsf{a}^{\mathsf{rew}}}$, thus allowing to select which knowledge to focus on, unlike in $\Upsilon_{\mathcal{D}}$ where we transfer all the knowledge from one state to another. We now define the existing relation between the the partial transition system $\Upsilon^p_{\mathcal{D}}$ and the transition system $\Upsilon_{\mathcal{D}}$. Given a path π^p in $\Upsilon^p_{\mathcal{D}}$, we say that π^p is a *proper partialization* of a path π in $\Upsilon_{\mathcal{D}}$ (resp., π is a *proper completion* of π^p) if:
– each state A^p_i is a subset of the relative state A_i ($A^p_i \subseteq A_i$);

– each transition is caused by the same action $\mathsf{a}_\mathsf{i}^{\mathsf{rew}}$ and the related variable assignments are equal ($\vartheta_i^p = \vartheta_i$).

Between $\Upsilon_\mathcal{D}$ and $\Upsilon_\mathcal{D}^p$ there is no relation such as bisimulation or even simulation; this is a clear (and intended) consequence of working with partial knowledge. This also means that we have no immediate way to know if, given a partial path π^p in $\Upsilon_\mathcal{D}^p$, we can use the same actions instantiations in $\Upsilon_\mathcal{D}$, and thus if it exists a path π that is a proper completion π^p. To overcome this problem, we extend the definition of the blocking query B by creating a *global blocking query B_{π^p} w.r.t to a finite partial path π^p*. B_{π^p} is a boolean UCQ that can be evaluated in the complete initial state A_0, and, if it is evaluated *False*, gives us the certainty that we can use the same actions instantiations found in π^p starting from A_0 without generating any inconsistent state w.r.t. T.

B_{π^p} is built by iteratively adding the single instantiated blocking queries $B_i\vartheta_i^p$ of the actions that compose π^p (Algorithm 1, the symbol $\top$ indicates a predicate whose evaluation is *true* in every interpretation). At each step, before adding the i-th instantiated blocking query $B_i\vartheta_i^p$ to B_{π^p}, we perform the following operations:

– check that $\text{ANS}(B_{\pi^p}, \emptyset, E_i^+\vartheta_i^p)$ is *False*;
– remove any CQ β in B_{π^p} that evaluates always *False* (i.e., contains (in)equalities that evaluates always to *False*, like $\mathsf{ind_i} = \mathsf{ind_l}$, or $\mathsf{ind_i} \neq \mathsf{ind_i}$);
– remove from each CQ the (in)equalities that evaluates always to *True*, as they do not influence the ending result. We are sure that no CQ will be left empty, because it would mean the whole CQ would always evaluate to *True*, and this would have blocked the first step;
– for each CQ β, generate a temporary CQ β_{temp} by removing all the (in)equalities and transform existential variables in free ones. Looking at how the blocking query is built, we have that β_{temp} is either empty (β is composed only of (in)equalities) or contains only one atomic assertion with at most one free variable. For example, if $\beta = \exists z.\mathsf{P}(\mathsf{i_1}, z) \wedge \mathsf{i_2} \neq z$, then $\beta_{temp} = \mathsf{P}(\mathsf{i_1}, z)$;
– perform $\text{ANS}(\beta_{temp}, \emptyset, E^-_{sub(\vartheta_i^p)})$:
  - if it evaluates to *True*, then it means that the instantiated negative effects $E^-_{sub(\vartheta_i^p)}$ remove the atom β_{temp}, and in this case we can remove the CQ β from B_{π^p};
  - if it returns answers of the type $\vartheta_{\beta_{temp}} = \{\mathsf{z} \mapsto \mathsf{ind}\}$, then it means that the instantiated negative effects $E^-_{sub(\vartheta_i^p)}$ remove the atom β_{temp} only if z is mapped to the individual ind. We thus add to the CQ β the inequality $z \neq \mathsf{ind}$.

Theorem 2. *Given a DKB $\mathcal{D}$, a finite partial path π^p, and its global blocking query B_{π^p}, if $\text{ANS}(B_{\pi^p}, \emptyset, A_0) = \emptyset$, then it exists a concretion π of π^p such that $\pi \in \Upsilon_\mathcal{D}$.*

Proof. For the proof of the theorem we remind the reader to the Appendix of the extended version of this paper [11].

Example 3. Consider the DKB $\mathcal{D}$ described by the following elements and which models a simple business scenario:

Algorithm 1: The algorithm to build the global blocking query B_{π^p}

input : A partial path π^p
output: An UCQ B_{π^p}

$B_{\pi^p} := \{\bot\}$
$i :=$ n. of transitions in π^p `// counter variable`

while $i > 0$ **do** `// each cycle refers to transition` $A^p_{i-1} \overset{\mathsf{a_i}\vartheta^p_i}{\rightarrow} A^p_i$
  if $\text{ANS}(B_{\pi^p}, \emptyset, E^+_i\vartheta^p_i) \neq \emptyset$ **then**
    $B_{\pi^p} := \top$ `// inconsistency in the i-th transition`
    break
  end
  foreach $\beta \in B_{\pi^p}$ **do**
    if *β contains (in)equalities that are always False* **then**
      $B_{\pi^p} := B_{\pi^p} \setminus \beta$ `// remove CQs that are always` *`False`*
    end
    remove from β (in)equalities that are always *True*
    $\beta_{temp} :=$ β without (in)equalities and existential operator
    if $\text{ANS}(\beta_{temp}, \emptyset, E^-_{sub(\vartheta^p_i)}) = True$ **then**
      $B_{\pi^p} := B_{\pi^p} \setminus \beta$ `//` $E^-_{sub(\vartheta^p_i)}$ `erases the CQ` β_{temp}
    else if $\text{ANS}(\beta_{temp}, \emptyset, E^-_{sub(\vartheta^p_i)}) \neq \emptyset$ **then**
      foreach $\vartheta_{\beta_{temp}} = \{\boldsymbol{z} \mapsto \mathsf{ind}\} \in \text{ANS}(\beta_{temp}, \emptyset, E^-_{sub(\vartheta^p_i)})$ **do**
        $\beta := \beta \wedge \mathbf{z} \neq \mathsf{ind}$ `// update the CQ` β
      end
    end
  end
  $B_{\pi^p} := B_{\pi^p} \cup B_i\vartheta^p_i$ `// add the blocking query of action` $\mathsf{a_i}$
  $i := i - 1$
end

- the TBox $T = \{\mathsf{Stored} \sqsubseteq \neg\mathsf{Shipped}\}$;
- the ABox $A_0 = \{\mathsf{Product(p1)}, \mathsf{Stored(p1)}, \mathsf{Product(p2)}\}$;
- the action set Γ composed by the following actions:
  pack: $\{\mathsf{Product}(x)\} \rightsquigarrow \{\mathsf{Packed}(x)\}^+$,
  ship: $\{\mathsf{Packed}(x)\} \rightsquigarrow \{\mathsf{Shipped}(x)\}^+$
  which becomes the set Γ^{rew} composed of the actions:
  $\mathsf{pack}^{\mathsf{rew}}$: $\{\mathsf{Product}(x)\} \rightsquigarrow \{\mathsf{Packed}(x)\}^+$,
  $\mathsf{ship}^{\mathsf{rew}}$: $\{\mathsf{Packed}(x)\}, \{\mathsf{Stored}(x)\} \rightsquigarrow \{\mathsf{Shipped}(x)\}^+$

At this point, we develop a partial transition system $\widehat{\Upsilon}_{\mathcal{D}}$ by considering the partial initial state $A^p_0 = \{\mathsf{Product(p1)}\}$. We can perform the sequence of transitions $\pi^p = A^p_0 \overset{\mathsf{pack}\vartheta}{\rightarrow} A^p_1 \overset{\mathsf{ship}\vartheta}{\rightarrow} A^p_2$, where: $\vartheta = \{x \mapsto \mathsf{p1}\}$, $A^p_1 = \{\mathsf{Packed(p1)}\}$, and $A^p_2 = \{\mathsf{Shipped(p1)}\}$. The global blocking query B_{π^p} is $\mathsf{Stored(p1)}$, and we see that, if we try to transpose π^p in the original ABox A_0, we have $\text{ANS}(B_{\pi^p}, \emptyset, A_0) \neq \emptyset$, thus meaning that π^p doesn't have a proper concretion π (indeed if we perform the two actions, we would end up having an inconsistent state A_2).

If we would consider instead the partial initial state $A_0^p = \{\mathsf{Product(p2)}\}$, instead, we woould be able to find a proper completion of π^p, as B_{π^p} would be $\mathsf{Stored(p2)}$ and $\text{ANS}(B_{\pi^p}, \emptyset, A_0) = \emptyset$.

Given a finite partial path π^p and its global blocking query B_{π^p}, we have a way to know if we can transform π^p into a complete path π without actually calculating it, only by performing an UCQ over the initial state A_0. Notice also that this result can be applied to all possible ABoxes, not only A_0; as long as A_0^p is contained in an ABox A, and $\text{ANS}(B_{\pi^p}, \emptyset, A) = \emptyset$, then it exists a path π which starts from A and is a proper concretion of π^p.

4 Conclusions

In this paper we formalize a framework, called Dynamic Knowledge Bases, aimed at modelling the dynamics of artifact-centric business processes. Such framework is represented by a transition system where states are defined by DL-Lite$_\mathcal{A}$ knowledge bases, and where a set of actions allows the system to evolve by adding or removing assertions, along with the possibility to introduce new instances. The expressive power and reasoning services of Description Logics are very helpful to describe and manage the domain knowledge, but constitute a difficult environment to deal with when it comes to the dynamics of the processes. To tackle this problem, we introduce two optimizations, namely action rewriting and the partialization of the transition system related to a Dynamic Knowledge Base: these optimizations give us a framework where we can work with partial knowledge and where the TBox is not needed, still guaranteeing that the resulting system is consistent with it. Given a path valid for the partial transition system, we can calculate its global blocking query, and know if it can be transferred to the complete transition system without any change, and without the need to do any other calculation.

Our work does not aim to propose a planning technique, neither try to give a solution w.r.t. the decidability/undecidability problem of plan research in our environment (since it is possible to generate an infinite transition system), but to create a framework that can be used as a formal domain-independent base to develop planning and decision making techniques for data-rich business domains by taking full advantage of the DL-Lite reasoning power.

We are currently working to further expand this framework in various directions. Under the theoretical side, we are already developing an abstraction of the transition system, in particular by expressing the needed knowledge by using only queries, which can be then used over the complete transition system. Under the practical side, we intend to propose a backward planning algorithm, which takes advantage of the abstract transition system and the possibility to work with partial knowledge to return all plans of interest w.r.t. a goal.

Although further investigation is surely needed, Dynamic Knowledge Bases are a promising framework that can be usefully employed to tackle the problem of planning and decision making in artifact-centric business domains.

References

1. Baader, F., Zarrieß, B.: Verification of Golog programs over description logic actions. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) 8152 LNAI, 181–196 (2013)
2. Bhattacharya, K., Gerede, C., Hull, R.: Towards formal analysis of artifact-centric business process models. Business Process Management pp. 288–304 (2007)
3. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Poggi, A., Rodriguez-Muro, M., Rosati, R.: Ontologies and Databases: the DL-Lite Approach 5689, 255–356 (2009)
4. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Poggi, A., Rosati, R.: Linking data to ontologies: The description logic DL-Lite$_{\mathcal{A}}$. CEUR Workshop Proceedings 216 (2006)
5. Calvanese, D., De Giacomo, G., Montali, M., Patrizi, F.: Verification and Synthesis in Description Logic Based Dynamic Systems, Lecture Notes in Computer Science, vol. 7994. Springer Berlin Heidelberg, Berlin, Heidelberg (2013)
6. Cohn, D., Hull, R.: Business artifacts: A data-centric approach to modeling business operations and processes. IEEE Data Eng. Bull 32(3), 3–9 (2009)
7. Ghallab, M., Nau, D.S., Traverso, P.: Automated planning - theory and practice. Elsevier (2004)
8. Gigerenzer, G., Gaissmaier, W.: Heuristic decision making. Annual review of psychology 62, 451–482 (2011)
9. Hariri, B.B., Calvanese, D., Montali, M., De Giacomo, G., Masellis, R.D., Felli, P.: Description Logic Knowledge and Action Bases. J. Artif. Intell. Res. (JAIR) 46, 651–686 (2013)
10. Levesque, H.J.: Foundations of a Functional Approach to Knowledge Representation. Artif. Intell. 23(2), 155–212 (1984)
11. Stawowy, M.: Optimizations for decision making and planning in description logic based dynamic knowledge bases (2015), `http://arxiv.org/abs/1502.04665`