Verification and Synthesis in Description Logic Based Dynamic Systems

Diego Calvanese[1], Giuseppe De Giacomo[2], Marco Montali[1], and Fabio Patrizi[2]

[1] Free University of Bozen-Bolzano, Piazza Domenicani 3, 39100 Bolzano, Italy
lastname@inf.unibz.it
[2] Sapienza Università di Roma, Via Ariosto, 25, 00185 Rome, Italy
lastname@dis.uniroma1.it

Abstract. In this paper, we devise a general framework for formalizing Description Logic Based Dynamic Systems that is parametric w.r.t. the description logic knowledge base and the progression mechanism of interest. Using this framework we study verification and adversarial synthesis for specifications expressed in a variant of first-order μ-calculus, with a controlled form of quantification across successive states. We provide key decidability results for both verification and synthesis, under a "bounded-state" assumption. We then study two instantiations of this general framework, where the description logic knowledge base is expressed in *DL-Lite* and $\mathcal{ALCQI}$, respectively.

1 Introduction

Integrating semantic mechanisms like Description Logics (DLs), to describe static knowledge, and full-fledged mechanisms like transition systems, to describe dynamics is of great interest, especially in the context of a semantic view of data-aware and artifact-based processes [28,10,19]. The tight combination of these two aspects into a single logical formalism is notoriously difficult [33,35]. In particular, due to the nature of DL assertions we get one of the most difficult kinds of static constraints for reasoning about actions [32,27]. Technically this combination gives rise to a semantics based on two-dimensional structures (DL domain + dynamics/time), which easily leads to undecidability [35,21]. To regain decidability, one has to restrict the combination only to concepts (vs. roles) [3,23,24]. This limitation is unsatisfactory in all those applications where we want to progress a full-fledged knowledge base (KB) representing a shared conceptualization of a domain of interest, such as an ontology, (a formalization of) a UML Class Diagram [9], or an Ontology Based Data Access System [12].

Recently, to overcome such difficulties, a looser coupling of the static and dynamic representation mechanisms has been proposed, giving rise to a rich body of research [4,16,13,6]. Virtually all such work is implicitly or explicitly based on Levesque's functional approach [25], where a KB is seen as a system that provides the ability of querying its knowledge in terms of logical implication/certain answers ("ask" operation), and the ability of progressing it through forms of updates ("tell" operation). As a consequence, the knowledge description formalism becomes decoupled from the formalism that describes the progression: we can define the dynamics through a transition system, whose

states are DL KBs, and transitions are labeled by the action (with object parameters) that causes the transition. The key issue in this context is that such transition systems are *infinite* in general, and hence some form of faithful abstraction is needed. Note that, if for any reason we learn that the number of states in this transition system is finite, then verifying dynamic properties over such systems amounts to a form a finite-state model checking [7].

In this paper, we follow this approach and devise a general framework for *DL Based Dynamic Systems*, which is parametric w.r.t. the DL used for the knowledge base and the mechanism used for progressing the state of the system. Using this framework, we study verification and (adversarial) synthesis for specifications expressed in a variant of first-order μ-calculus, with a controlled form of quantification across successive states. We recall that μ-calculus subsumes virtually all logics used in verification, including LTL, CTL and CTL*. Adversarial synthesis for μ-calculus captures a wide variety of synthesis problems, including conditional planning with full-observability [22], behaviour composition [18], and several other sophisticated forms of synthesis and planning [17].

We provide key decidability results for verification under a "bounded-state" assumption. Such assumption states that while progressing, the system can change arbitrarily the stored individuals but their total number at each time point cannot exceed a certain bound. Notice that along an infinite run (and hence in the overall transition system), the total numer of individuals can still be infinite. We then turn to adversarial synthesis, where we consider the system engaged in a sort of game with an adversarial environment. The two agents (the system and the environment) move in alternation, and the problem is to synthesize a strategy for the system to force the evolution of the game so as to satisfy a given synthesis specification. Such a specification is expressed in the above first-order μ-calculus, using the temporal operators to express that the system is able to force a formula Φ in the next state regardless of the environment moves, as in the strategy logic ATL [2]. We show again decidability under the "bounded-state" assumption (this time for the game structure).

The rest of the paper is organized as follows. Section 2 introduces the general framework of DL Based Dynamic Systems. In Section 3, we introduce the verification formalism, based on first-order μ-calculus with a controlled form of quantification across states. In Section 4, we show the general decidability result of model checking such a variant of μ-calculus against DL Based Dynamic Systems. In Section 5, we turn to adversarial synthesis and show its decidability in our setting. In Section 6, we study the instantiation of the framework in which the DL knowledge base is expressed in *DL-Lite* or $\mathcal{ALCQI}$, and the progression mechanism is that of [6]. In Section 7, we draw final conclusions.

2 Framework

In this paper we follow Leveque's functional approach [25]. We introduce Description Logic Based Dynamic Systems (DLDSs), which are systems constituted by a DL knowledge base, consisting of a fixed TBox and an ABox that changes as the system evolves, and a set of actions that step-wise progress the knowledge base by changing the ABox. We formalize such systems in a general form, referring neither to any specific DL nor

to any specific action representation formalism, and making only minimal assumptions on the various components. Later, in Section 6, we show concrete instantiations of the framework.

Object universe. We fix a countably infinite (object) universe Δ of individuals. These constants, which act as standard names [26], allow us to refer to individuals across distinct timepoints.

DL knowledge bases. A (DL) *knowledge base* (KB) (T, A) is constituted by a TBox T representing the intensional knowledge about the domain of interest in terms of concepts and roles, and an ABox A representing extensional knowledge about individuals. The TBox T is a finite set of universal assertions (we don't allow nominals, to avoid confusion between intensional and extensional levels). The ABox A is a finite set of *assertions* consisting of facts, i.e., atomic formulas of the form $N(d)$ and $P(d, d')$, where N is a concept name, P is a role name (we use binary roles, but all our results can be extended to n-ary relations), and d, d' are individual constants in Δ. We use $\text{ADOM}(A)$ to denote the set of constants actually appearing in A. We adopt the standard DL semantics based on first-order interpretations and on the notion of (first-order) model. Naturally, the only TBoxes of interest for us are those that are *satisfiable*, i.e., admit at least one model. We say that an ABox A is *consistent w.r.t. a TBox* T if (T, A) is satisfiable, and that (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. As usual in DLs, we assume that reasoning, i.e., checking KB satisfiability and logical implication are decidable tasks.

Description Logic based Dynamic Systems. A *Description Logic based Dynamic System* (*DLDS*) is a tuple $\mathcal{S} = (T, A_0, \Gamma)$, where (T, A_0) is a KB and Γ is a finite set of *actions*. The set $\text{ADOM}(A_0)$ of constants in A_0 are called *distinguished* and we denoted them by C, i.e., $C = \text{ADOM}(A_0)$. Such constants play a special role because they are the only ones that can be used in verification formulas (see Section 4). Actions are parametrized, and when an action is executed formal parameters are substituted with individual constants from Δ. Formal parameters are taken from a countably infinite set P of *parameter names*. Interestingly, we do not assume that the number of formal parameters of an action is fixed a priory, but we allow it to depend on the KB (actually, the ABox, since the TBox is fixed) on which the action is executed.

Notice that this mechanism is more general than what typically found in procedures/functions of programming languages, where the number of parameters is fixed by the signature of the procedure/function. Our mechanism is directly inspired by web systems, in which input forms are dynamically constructed and customized depending on data already acquired. For example, when inserting author data in a conference submission system, the input fields that are presented to the user depend on the (previously specified) number of authors.

Once formal parameters are substituted by actual ones, executing the action has the effect of generating a new ABox. We use $\mathcal{A}_T$ to denote the set of all ABoxes that can be constructed using concept and role names in T, and individuals in Δ. Formally, each action in Γ has the form (π, τ), where

- $\pi : \mathcal{A}_T \to 2^P$ is a *parameter selection function* that, given an ABox A, returns the *finite* set $\pi(A) \subseteq P$ of parameters of interest for τ w.r.t. A (see below);
- $\tau : \mathcal{A}_T \times \Delta^P \mapsto \mathcal{A}_T$, is a (partial) *effect function* that, given an ABox A and a *parameter assignment* $m : \pi(A) \to \Delta$, returns (if defined) the ABox $A' = \tau(A, m)$, which *(i)* is consistent wrt T, and *(ii)* contains only constants in $\textsc{adom}(A) \cup \textsc{im}(m)$.[3]

Observe that since $\pi(A)$ is finite, so is $\textsc{im}(m)$, thus only finitely many new individuals, w.r.t. A, can be added to A'.

In fact, we focus our attention on DLDS's that are generic, which intuitively means that the two functions constituting an action are invariant w.r.t. renaming of individuals[4]. Genericity is a natural assumption that essentially says that the properties of individuals are only those that can be inferred from the KB.

To capture genericity, we first introduce the notion of equivalence of ABoxes modulo renaming. Specifically, given two ABoxes A_1, A_2, sets $S_1 \supseteq \textsc{adom}(A_1) \cup C$ and $S_2 \supseteq \textsc{adom}(A_2) \cup C$, and a bijection $h : S_1 \to S_2$ that is the identity on C, we say that A_1 and A_2 are *logically equivalent modulo renaming h w.r.t. a TBox T*, written $A_1 \cong^h_T A_2$, if:

1. for each assertion α_1 in A_1, $(T, A_2) \models h(\alpha_1)$;
2. for each assertion α_2 in A_2, $(T, A_1) \models h^{-1}(\alpha_2)$;

where $h(\alpha_1)$ (resp. $h^{-1}(\alpha_2)$) is a new assertion obtained from α_1 (resp., α_2), by replacing each occurrence of an individual $d \in \Delta$ with $h(d)$ (resp., $h^{-1}(d)$). We say that A_1 and A_2 are *logically equivalent modulo renaming w.r.t.* T, written $A_1 \cong_T A_2$, if $A_1 \cong^h_T A_2$ for some h. We omit T when clear from the context.

A DLDS $\mathcal{S} = (T, A_0, \Gamma)$ is *generic*, if for every $(\pi, \tau) \in \Gamma$ and every $A_1, A_2 \in \mathcal{A}_T$ s.t. $A_1 \cong_T A_2$, we have that *(i)* $\pi(A_1) = \pi(A_2)$, and *(ii)* for every two parameter assignments $m_1 : \pi(A_1) \to \Delta$ and $m_2 : \pi(A_2) \to \Delta$, if there exists a bijection $h : \textsc{adom}(A_1) \cup C \cup \textsc{im}(m_1) \to \textsc{adom}(A_2) \cup C \cup \textsc{im}(m_2)$ s.t. $A_1 \cong^h_T A_2$, then, whenever $A'_1 = \tau(A_1, m_1)$ is defined then also $A'_2 = \tau(A_2, m_2)$ is defined and vice-versa, and moreover $A'_1 \cong^h_T A'_2$.

DLDS Transition System. The dynamics of a DLDS $\mathcal{S}$ is characterized by the *transition system* $\Upsilon_\mathcal{S}$ it generates. The kind of transition systems we consider here have the general form $\Upsilon = (U, T, \Sigma, s_0, abox, \Rightarrow)$, where: *(i)* U is the universe of individual constants, which includes the distinguished constants C; *(ii)* T is a TBox; *(iii)* Σ is a set of states; *(iv)* $s_0 \in \Sigma$ is the initial state; *(v)* $abox$ is a function that, given a state $s \in \Sigma$, returns an ABox associated with s, which has terms of Δ as individuals, and which conforms to T; *(vi)* $\Rightarrow \subseteq \Sigma \times \mathcal{L} \times \Sigma$ is a labeled transition relation between states, where $\mathcal{L}$ is the set of labels.

With a little abuse of notation we also introduce an unlabeled transition relation $\Rightarrow \subseteq \Sigma \times \Sigma$ obtained from the labeled transition relation by projecting out the labels (we use this notion when dealing with verification properties). Given a DLDS $\mathcal{S} = (T, A_0, \Gamma)$, its *(generated) transition system* $\Upsilon_\mathcal{S} = (U, T, \Sigma, s_0, abox, \Rightarrow)$ is defined as: *(i)* $U = \Delta$; *(ii)* $abox$ is the identity function (thus $\Sigma \subseteq \mathcal{A}_T$); *(iii)* $s_0 = A_0$; *(iv)* $\Rightarrow \subseteq \Sigma \times \mathcal{L} \times \Sigma$ is a labeled transition relation where $\mathcal{L} = \Gamma \times \mathcal{M}$, with $\mathcal{M}$ the domain of action parameter

[3] By $\textsc{im}(\cdot)$ we denote the *image* of a function.

[4] This name is due to the notion of genericity in databases [1], also called *uniformity* in [8].

assignments, is a set of labels containing one pair (a, m) for every action $a \in \Gamma$ and corresponding parameter assignment $m \in \mathcal{M}$; *(v)* Σ and $\Rightarrow$ are defined by mutual induction as the smallest sets satisfying the following property: if $A \in \Sigma$ then for every $(\pi, \tau) \in \Gamma$, $m : \pi(A) \to \Delta$, and $A' \in \mathcal{A}_T$, s.t. $A' = \tau(A, m)$, we have $A' \in \Sigma$ and $A \overset{\ell}{\Rightarrow} A'$, s.t. $\ell = ((\pi, \tau), m)$.

3 Specification Logic μDL_p

To specify dynamic properties over DLDSs, we use a first-order variant of μ-calculus [34,30]. μ-calculus is virtually the most powerful temporal logic used for model checking of finite-state transition systems, and is able to express both linear time logics such as LTL and PSL, and branching time logics such as CTL and CTL* [15]. The main characteristic of μ-calculus is its ability of expressing directly least and greatest fixpoints of (predicate-transformer) operators formed using formulae relating the current state to the next one. By using such fixpoint constructs one can easily express sophisticated properties defined by induction or co-induction. This is the reason why virtually all logics used in verification can be considered as fragments of μ-calculus. Technically, μ-calculus separates local properties, asserted on the current state or on states that are immediate successors of the current one, from properties talking about states that are arbitrarily far away from the current one [34]. The latter are expressed through the use of fixpoints.

In our variant of μ-calculus, we allow local properties to be expressed as queries in any language for which query entailment for DL KBs is decidable [11,14]. Specifically, given a KB (T, A), a query Q, and an assignment v for the free variables of Q, we say that Qv *is entailed by* (T, A), if $(T, A) \models Qv$, i.e., (T, A) logically implies the formula Qv obtained from Q by substituting its free variables according to v. Notice that the set of v such that $(T, A) \models Qv$ are the so-called *certain answers*.

At the same time we allow for a controlled form of first-order quantification across states, inspired by [5], where the quantification ranges over individual objects across time only as long as such object persist in the active domain. Formally, we define the logic μDL_p as:

$$\begin{aligned}\Phi ::= Q \mid \neg\Phi \mid \Phi_1 \wedge \Phi_2 \mid \exists x.\textsc{live}(x) \wedge \Phi \mid \\ \textsc{live}(\boldsymbol{x}) \wedge \langle\!-\!\rangle\Phi \mid \textsc{live}(\boldsymbol{x}) \wedge [\!-\!]\Phi \mid Z \mid \mu Z.\Phi\end{aligned}$$

where Q is a (possibly open) query as described above, in which the only constants that may appear are those in C, Z is a second order predicate variable (of arity 0), and, with a slight abuse of notation, we write $\textsc{live}(x_1, \ldots, x_n) = \bigwedge_{i \in \{1,\ldots,n\}} \textsc{live}(x_i)$. For μDL_p, the following assumption holds: in $\textsc{live}(\boldsymbol{x}) \wedge \langle\!-\!\rangle\Phi$ and $\textsc{live}(\boldsymbol{x}) \wedge [\!-\!]\Phi$, the variables $\boldsymbol{x}$ are exactly the free variables of Φ, once we substitute to each bounded predicate variable Z in Φ its bounding formula $\mu Z.\Phi'$. We use the usual abbreviations, including: $\textsc{live}(\boldsymbol{x}) \to \langle\!-\!\rangle\Phi = \neg(\textsc{live}(\boldsymbol{x}) \wedge [\!-\!]\neg\Phi)$ and $\textsc{live}(\boldsymbol{x}) \to [\!-\!]\Phi = \neg(\textsc{live}(\boldsymbol{x}) \wedge \langle\!-\!\rangle\neg\Phi)$. Intuitively, the use of $\textsc{live}(\cdot)$ in μDL_p ensures that individuals are only considered if they persist along the system evolution, while the evaluation of a formula with individuals that are not present in the current database trivially leads to false or true.

$$
\begin{aligned}
(Q)^{\Upsilon}_{v,V} &= \{s \in \Sigma \mid (T, abox(s)) \models Qv\} \\
(\neg\Phi)^{\Upsilon}_{v,V} &= \Sigma \setminus (\Phi)^{\Upsilon}_{v,V} \\
(\Phi_1 \wedge \Phi_2)^{\Upsilon}_{v,V} &= (\Phi_1)^{\Upsilon}_{v,V} \cap (\Phi_2)^{\Upsilon}_{v,V} \\
(\exists x.\textsc{live}(x) \wedge \Phi)^{\Upsilon}_{v,V} &= \{s \in \Sigma \mid \exists d \in \textsc{adom}(abox(s)).s \in (\Phi)^{\Upsilon}_{v[x/d],V}\} \\
(\textsc{live}(\boldsymbol{x}) \wedge \langle\!-\!\rangle\Phi)^{\Upsilon}_{v,V} &= \{s \in \Sigma \mid \boldsymbol{x}/\boldsymbol{d} \in v \text{ implies } \boldsymbol{d} \subseteq \textsc{adom}(abox(s)) \\
&\qquad \text{and } \exists s'.s \Rightarrow s' \text{ and } s' \in (\Phi)^{\Upsilon}_{v,V}\} \\
(\textsc{live}(\boldsymbol{x}) \wedge [\!-\!]\Phi)^{\Upsilon}_{v,V} &= \{s \in \Sigma \mid \boldsymbol{x}/\boldsymbol{d} \in v \text{ implies } \boldsymbol{d} \subseteq \textsc{adom}(abox(s)) \\
&\qquad \text{and } \forall s'.s \Rightarrow s' \text{ implies } s' \in (\Phi)^{\Upsilon}_{v,V}\} \\
(Z)^{\Upsilon}_{v,V} &= V(Z) \\
(\mu Z.\Phi)^{\Upsilon}_{v,V} &= \bigcap\{\mathcal{E} \subseteq \Sigma \mid (\Phi)^{\Upsilon}_{v,V[Z/\mathcal{E}]} \subseteq \mathcal{E}\}
\end{aligned}
$$

Fig. 1. Semantics of μDL_p.

The formula $\mu Z.\Phi$ denotes the least fixpoint of the formula Φ (seen as the predicate transformer $\lambda Z.\Phi$). As usual in μ-calculus, formulae of the form $\mu Z.\Phi$ must obey to the *syntactic monotonicity* of Φ w.r.t. Z, which states that every occurrence of the variable Z in Φ must be within the scope of an even number of negation symbols. This ensures that the least fixpoint $\mu Z.\Phi$ always exists.

The semantics of μDL_p formulae is defined over possibly infinite transition systems of the form $(U, T, \Sigma, s_0, abox, \Rightarrow)$ seen above. Since μDL_p also contains formulae with both individual and predicate free variables, given a transition system Υ, we introduce an *individual variable valuation* v, i.e., a mapping from individual variables x to U, and a *predicate variable valuation* V, i.e., a mapping from the predicate variables Z to subsets of Σ. With these three notions in place, we assign meaning to formulae by associating to Υ, v, and V an *extension function* $(\cdot)^{\Upsilon}_{v,V}$, which maps formulae to subsets of Σ. Formally, the extension function $(\cdot)^{\Upsilon}_{v,V}$ is defined inductively as shown in Figure 1. Intuitively, $(\cdot)^{\Upsilon}_{v,V}$ assigns to such constructs the following meaning. *(i)* The boolean connectives have the expected meaning. *(ii)* The quantification of individuals is done over the individuals of the "current" ABox, using the special $\textsc{live}(\cdot)$ predicate (*active domain quantification*). Notice that such individuals can be referred in a later state, provided that they persist in between (see below). *(iii)* The extension of $\langle\!-\!\rangle\Phi$ consists of the states s s.t.: for *some* successor state s' of s, Φ holds in s' under v. *(iv)* The extension of $\mu Z.\Phi$ is the *smallest subset* $\mathcal{E}_\mu$ of Σ s.t., assigning to Z the extension $\mathcal{E}_\mu$, the resulting extension of Φ (under v) is contained in $\mathcal{E}_\mu$. That is, the extension of $\mu Z.\Phi$ is the *least fixpoint* of the operator $(\Phi)^{\Upsilon}_{v,V[Z/\mathcal{E}]}$, where $V[Z/\mathcal{E}]$ denotes the predicate valuation obtained from V by forcing the valuation of Z to be $\mathcal{E}$. When Φ is a closed formula, $(\Phi)^{\Upsilon}_{v,V}$ does not depend on v or V, and we denote the extension of Φ simply by $(\Phi)^{\Upsilon}$. A closed formula Φ holds in a state $s \in \Sigma$ if $s \in (\Phi)^{\Upsilon}$. In this case, we write $\Upsilon, s \models \Phi$. A closed formula Φ holds in Υ, denoted by $\Upsilon \models \Phi$, if $\Upsilon, s_0 \models \Phi$. Given a DLDS $\mathcal{S}$, we say that $\mathcal{S} \models \Phi$ if $\Upsilon_{\mathcal{S}} \models \Phi$. We call *model checking* the problem of verifying whether $\Upsilon \models \Phi$ holds.

The bisimulation relation that captures μDL_p is as follows. Let $\Upsilon_1 = (U_1, T, \Sigma_1, s_{01}, abox_1, \Rightarrow_1)$ and $\Upsilon_2 = (U_2, T, \Sigma_2, s_{02}, abox_2, \Rightarrow_2)$ be transition systems, s.t. $C \subseteq U_1 \cap U_2$, and H the set of partial bijections between U_1 and U_2, which

are the identity over C. A *persistence preserving bisimulation* between Υ_1 and Υ_2 is a relation $\mathcal{B} \subseteq \Sigma_1 \times H \times \Sigma_2$ such that $(s_1, h, s_2) \in \mathcal{B}$ implies that:

1. $abox_1(s_1) \cong_T^h abox_2(s_2)$;
2. for each s_1', if $s_1 \Rightarrow_1 s_1'$ then there exists an s_2' with $s_2 \Rightarrow_2 s_2'$ and a bijection h' that extends h restricted on $\textsc{adom}(abox_1(s_1)) \cap \textsc{adom}(abox_1(s_1'))$, such that $(s_1', h', s_2') \in \mathcal{B}$;
3. for each s_2', if $s_2 \Rightarrow_2 s_2'$ then there exists an s_1' with $s_1 \Rightarrow_1 s_1'$ and a bijection h' that extends h restricted on $\textsc{adom}(abox_1(s_1)) \cap \textsc{adom}(abox_1(s_1'))$, such that $(s_1', h', s_2') \in \mathcal{B}$.

We say that a state $s_1 \in \Sigma_1$ is *persistence preserving bisimilar* to $s_2 \in \Sigma_2$ *wrt a partial bijection* h, written $s_1 \sim_h s_2$, if there exists a persistence preserving bisimulation $\mathcal{B}$ between Υ_1 and Υ_2 such that $(s_1, h, s_2) \in \mathcal{B}$. A transition system Υ_1 is *persistence preserving bisimilar* to Υ_2, written $\Upsilon_1 \sim \Upsilon_2$, if there exists a partial bijection h_0 and a persistence preserving bisimulation $\mathcal{B}$ between Υ_1 and Υ_2 s.t. $(s_{01}, h_0, s_{02}) \in \mathcal{B}$. A suitable bisimulation-invariance theorem hold:

Theorem 1. *Consider two transition systems Υ_1 and Υ_2 s.t. $\Upsilon_1 \sim \Upsilon_2$. Then for every μDL_p closed formula Φ, we have that $\Upsilon_1 \models \Phi$ if and only if $\Upsilon_2 \models \Phi$.*

The proof follows the line of an analogous one in [5]. The key difference is that the local condition for bisimulation is replaced by logical equivalence modulo renaming. This condition guarantees the preservation of certain answers, which is a sufficient condition for preservation of μDL_p formulae, as their evaluation depends only on certain answers.

4 Verification

Next we focus on checking a μDL_p formula against a DLDS. It is easy to show, by reduction from the halting problem, that this is in general undecidable, even under the assumption of genericity (see, e.g., [6]). On the other hand, it can be shown that for finite Δ, the problem is decidable. Indeed, in such a case, only finitely many ABoxes exist, thus, by quantifier elimination, one can can reduce the problem to model checking of propositional μ-calculus.

Undecidability calls for the identification of decidable classes of the problem. To this end, we focus our attention on a particular class of DLDS, which we call *state-bounded*. Under the assumption of genericity, we are able to prove decidability of verification for state-bounded DLDS. A *state-bounded* DLDS $\mathcal{K}$ is one for which there exists a finite bound b s.t., for each state s of $\Upsilon_\mathcal{K}$, $|\textsc{adom}(abox(s))| < b$. When this is the case, we say that $\mathcal{K}$ is *b-bounded*. Observe that state-bounded DLDS contain in general infinitely many states, and that a DLDS $\mathcal{K}$ can be state-unbounded even if, for every state s of $\Upsilon_\mathcal{K}$, $|\textsc{adom}(abox(s))|$ is finite (but not bounded). W.l.o.g., for state-bounded DLDS, we assume that the maximum number n of parameters that actions may request is known.[5]

We prove now that model checking of μDL_p over state-bounded, generic DLDS is decidable, by showing how it can be reduced to model checking of propositional

[5] This number can be obtained in PSPACE by constructing all the ABoxes of size $\leq b$, up to logical equivalence modulo renaming, and by applying all actions to them, so as to obtain the corresponding parameters.

μ-calculus over finite-state transition systems. The crux of the proof, outlined below, is the construction of a finite-state transition system $\Upsilon_{\mathcal{K}}^{D}$ s.t. $\Upsilon_{\mathcal{K}}^{D} \sim \Upsilon_{\mathcal{K}}$, where $\Upsilon_{\mathcal{K}}$ is the transition system generated by $\mathcal{K}$. This is done through an abstraction technique inspired by that of [5]. Differently from that setting, we deal with DL knowledge bases, instead of relational databases.

Observe that the infiniteness of $\Upsilon_{\mathcal{K}}$ comes from that of Δ, which yields a potential infinite number of ABoxes and assignments to action parameters, thus infinite branching. As a first step, we show that $\Upsilon_{\mathcal{K}}$ can be "pruned", so as to obtain a persistence-bisimilar finite-branching transition system $\Theta_{\mathcal{K}}$, and that any transition system obtained in this way is persistence-bisimilar to $\Upsilon_{\mathcal{K}}$.

To define such *prunings* we introduce the notion of *equality commitment*, i.e., a set of equality constraints involving parameter assignments and distinguished individuals. An *equality commitment* over a finite set $S \subset \Delta \cup P$ of individuals and parameters, is a partition $H = \{H_1, \ldots, H_n\}$ of S, s.t. every H_i contains at most one $d \in \Delta$. Given $(\pi, \tau) \in \Gamma$, an ABox A, and an equality commitment H over $\textsc{adom}(A) \cup \pi(A) \cup C$, a parameter assignment $m : \pi(A) \to \Delta$ *respects* H if, for every two parameters $p_1, p_2 \in \pi(A)$, $m(p_1) = m(p_2)$ iff p_1 and p_2 belong to the same $H_i \in H$, and $m(p_1) = m(p_2) = d$, iff $d \in H_i$. Observe that, being S finite, the number of possible equality commitments over S, is finite, too.

A *pruning* of $\Upsilon_{\mathcal{K}}$ is a transition system $\Theta_{\mathcal{K}} = (\Delta, T, \Sigma, s_0, abox, \Rightarrow)$, s.t.: *(i)* $abox$ is the identity; *(ii)* $s_0 = A_0 \in \Sigma$; *(iii)* Σ and $\Rightarrow$ are defined as smallest sets constructed by mutual induction from s_0 as follows: if $A \in \Sigma$, then for every $(\pi, \tau) \in \Gamma$ and every equality commitment H over $\textsc{adom}(A) \cup \pi(A) \cup C$, nondeterministically choose finitely many parameter assignments $m_i : \pi(A) \to \Delta$ respecting H, and add transitions $A \stackrel{\ell}{\Rightarrow} A'$ where $A' = \tau(A, m_i)$ and $\ell = ((\pi, \tau), m_i)$.

Intuitively, a pruning is obtained from $\Upsilon_{\mathcal{K}}$, by executing, at each reachable state A, an arbitrary, though *finite*, set of instantiated actions. This set is required to cover Γ and all the possible equality commitments over the parameters that the selected action requires. It can be seen that, since Γ is finite and, for fixed A, only finitely many H exist, prunings have always finite branching, even if $\Upsilon_{\mathcal{K}}$ does not. The following lemma says that, despite their structural differences, if $\mathcal{K}$ is generic, then $\Upsilon_{\mathcal{K}}$ is preservation-bisimilar to $\Theta_{\mathcal{K}}$.

Lemma 1. *Given a generic DLDS $\mathcal{K}$ and its transition system $\Upsilon_{\mathcal{K}}$, for every pruning $\Theta_{\mathcal{K}}$ of $\Upsilon_{\mathcal{K}}$, we have $\Theta_{\mathcal{K}} \sim \Upsilon_{\mathcal{K}}$.*

Proof (sketch). By genericity, for an ABox A and an action (π, τ), if two parameter assignments $m_1, m_2 : \pi(A) \to \Delta$ respect the same equality commitment H on $\textsc{adom}(A) \cup \pi(A) \cup C$, the applications of (π, τ) with m_1 and m_2, result in logically equivalent (modulo renaming) ABoxes, i.e., $A_1 = \tau_1(A, m_1) \cong_T^h A_2 = \tau_2(A, m_2)$. Further, still by genericity, we have that $\textsc{adom}(A) \cap \textsc{adom}(A_1) = \textsc{adom}(A) \cap \textsc{adom}(A_2)$, that is, A_1 and A_2 preserve the same values w.r.t. A. This can be taken as the basic step for constructing a persistence-preserving bisimulation between $\Upsilon_{\mathcal{K}}$ and $\Theta_{\mathcal{K}}$, starting from the observation that $A_0 \cong_T A_0$. □

Prunings are in general infinite-state. An important question is whether there exists some that are finite-state and effectively computable. This, by Lemma 1, Th. 1, and

the fact that verification is decidable for finite-state transition systems, would yield decidability of verification for $\mathcal{K}$. While this is not the case in general (we already stated undecidability), we can prove that this holds on state-bounded, generic DLDS.

Given a b-bounded, generic DLDS, consider a finite set D of individual constants s.t. $D \subset \Delta$ and $C \subseteq D$. Let $\Upsilon_{\mathcal{K}}^{D}$ be the fragment of transition system of $\mathcal{K}$ built using only individuals from D in action parameter assignments. Such a $\Upsilon_{\mathcal{K}}^{D}$ is finite and effectively computable. We have the following result.

Lemma 2. *If $|D| \geq b+n+|C|$, then $\Upsilon_{\mathcal{K}}^{D}$ is a pruning of $\Upsilon_{\mathcal{K}}$.*

Proof (sketch). The construction of $\Upsilon_{\mathcal{K}}^{D}$ essentially follows the inductive structure of the definition of pruning. In particular, since D is finite, only finitely many parameter assignments are considered at each step. It can be seen that, because $D \geq n$, for every action application and corresponding equality commitment H, we can find a parameter assignment that respects H. Further, since $D \geq b + n$ we have enough individuals to construct, at each step, the successor state, containing at most b individuals, that results from the action application. Finally, genericity ensures that the particular choice of D, as long as containing C, does not affect, except for individual renaming, the behavior of the effect function τ. □

Observe that this construction uses the bound b to build the finite set D, before constructing the pruning. An alternative strategy that does not need to know b in advance, consists in maximizing the reuse of previously introduced individuals. In this approach new individuals are added, as new states are generated, only if needed, in particular avoiding the generation of a new state whenever one logically equivalent (modulo renaming) has already been generated. A construction on this line, although in a different setting, is proposed in [5].

Together, Lemma 1 and 2 imply the following decidability result:

Theorem 2. *Model checking of μDL_p over a state-bounded, generic DLDS $\mathcal{K}$ is decidable, and can be reduced to model checking of propositional μ-calculus over a finite-state transition system, whose number of state is at most exponential in the size of $\mathcal{K}$.*

Proof (sketch). By Lemma 1, we know that $\Upsilon_{\mathcal{K}}^{D}$ is bisimilar to $\Upsilon_{\mathcal{K}}$. By Lemma 2, we know that $\Upsilon_{\mathcal{K}}^{D}$ contains at most an exponential number of states in the size of the specification $\mathcal{K}$, which includes the initial ABox, thus the (finite) set of distinguished constants C. Hence, by Theorem 1, we can check formulae over the finite transition system $\Upsilon_{\mathcal{K}}^{D}$ instead of $\Upsilon_{\mathcal{K}}$. Since the number of objects present in $\Upsilon_{\mathcal{K}}^{D}$ is finite, μDL_p formulae can be propositionalized and checked through standard algorithms [20]. □

5 Adversarial Synthesis

We turn now to the problem of adversarial synthesis, i.e., we consider a setting in which two agents act in turn as adversaries. The first agent, called *environment*, acts autonomously, whereas we control the second agent, called *system*. The joint behavior of the two agents gives rise to a so-called *two-player game structure* (2GS) [17,31,29], which can be seen as the arena of a game. On top of the 2GS we can formulate, using variants of μ-calculus, what the system should obtain in spite of the adversarial moves

of the environment. This specification can be considered the goal of the game for the system. The synthesis problem amounts to synthesizing a *strategy*, i.e., a suitable refined behavior for the system that guarantees to the system the fulfillment of the specification. Many synthesis problems can be rephrased using 2GS. An example is *conditional planning in nondeterministic fully observable domains*, where the system is the action executor, and the environment is the domain that nondeterministically chooses the (fully observable) effect of the action among those possible [22]. Another example is *behavior composition*, which aims at synthesizing an orchestrator that realizes a given target behavior by delegating the execution of actions to available behaviors. Here the system is the orchestrator, and the environment is formed by the target and the available behaviors. The goal of the system is to maintain over time the ability of delegating requested actions [18]. Several other sophisticated forms of synthesis and planning can be captured through 2GS, see, e.g., [17].

We can deal with adversarial synthesis in our framework by building a 2GS in which we encode explicitly in the DL KB the alternation of the moves of the two players. Specifically, we introduce a fresh concept name Turn and two fresh distinguished constants t_e and t_s, whose (mutual exclusive) presence in Turn indicates which of the two players will move next. We denote with $\mathcal{A}_T^e$ the set of ABoxes in $\mathcal{A}_T$ that contain $\mathsf{Turn}(e)$ (and not $\mathsf{Turn}(s)$). Similarly, for $\mathcal{A}_T^s$.

A *DL based 2GS* (*DL2GS*) is a DLDS $\mathcal{K} = (T, A_0, \Gamma)$, where $\mathsf{Turn}(e) \in A_0$ and the set of actions Γ is partitioned into a set Γ_e of *environment actions* and a set Γ_s of *system actions*. The effect function of each environment action is defined only for ABoxes in Γ_e and brings about an ABox in Γ_s. Symmetrically, the effect function of each system action is defined only for ABoxes in Γ_s and brings about an ABox in Γ_e. In this way we achieve the desired alternation of environment and system moves: starting from the initial state, the environment moves arbitrarily and the system suitably responds to the environment move, and this is repeated (possibly forever).

Logics of interest for 2GSs are temporal logics in the style of ATL [2], in which "next Φ" has the meaning of "system can force Φ" to hold in the next state, by suitably responding to every move done by the environment. Here, we introduce a specialization of the logic μDL_p, called μADL_p, defined as follows:

$$\begin{aligned}\Phi ::= {} & Q \mid \neg Q \mid \Phi_1 \wedge \Phi_2 \mid \Phi_1 \vee \Phi_2 \mid Z \mid \mu Z.\Phi \mid \nu Z.\Phi \mid \\ & \exists x.\textsc{live}(x) \wedge \Phi \mid \forall x.\textsc{live}(x) \rightarrow \Phi \mid \\ & \textsc{live}(\boldsymbol{x}) \wedge [\!\langle\!\rightarrow]\!]_s \Phi \mid \textsc{live}(\boldsymbol{x}) \rightarrow [\!\langle\!\rightarrow]\!]_s \Phi \mid \\ & \textsc{live}(\boldsymbol{x}) \wedge [\!\langle\!\rightarrow]\!]_w \Phi \mid \textsc{live}(\boldsymbol{x}) \rightarrow [\!\langle\!\rightarrow]\!]_w \Phi\end{aligned}$$

where $[\!\langle\!\rightarrow]\!]_s \Phi$ stands for $\boxdot(\textsc{live}(\boldsymbol{x}) \wedge \langle\!\rightarrow\!\rangle \Phi)$, and $[\!\langle\!\rightarrow]\!]_w \Phi$ stands for $\boxdot(\textsc{live}(\boldsymbol{x}) \rightarrow \langle\!\rightarrow\!\rangle \Phi)$. Technically, μADL_p is a fragment of μDL_p, where negation normal form is enforced, i.e., negation is pushed inwards so that it appears only in front of atoms (in our case, queries Q).

Intuitively, both $[\!\langle\!\rightarrow]\!]_s \Phi$ and $[\!\langle\!\rightarrow]\!]_w \Phi$ mean that the system can force Φ to hold, i.e., for every (environment) move there is a (system) move leading to Φ. The difference between the two is that $[\!\langle\!\rightarrow]\!]_s \Phi$ precludes the environment from dropping objects existing before its move, while $[\!\langle\!\rightarrow]\!]_w \Phi$ does not.

As an immediate consequence of Theorem 2 we have:

Theorem 3. *Checking μADL_p formulas over state-bounded, generic DL2GS's is decidable.*

We are not only interested in verifying μADL_p formulas, but also in *synthesizing* strategies to actually fulfill them. A *strategy* is a partial function $f : (\mathcal{A}^e_T \times \mathcal{A}^s_T)^* \times \mathcal{A}^e_T \mapsto \Gamma_s \times \mathcal{M}$ where $\mathcal{M}$ is the domain of action parameter assignments, such that for every *history* $\lambda = A^e_0, A^s_0 \cdots A^e_n, A^s_n, A^e_{n+1}$, it is the case that $A^s_i = \tau_{f(\lambda[i])}(A^e_i, m_{f(\lambda[i])})$ where $\lambda[i]$ is prefix of λ of length i, and $m_{f(\lambda[i])} : \pi_{f(\lambda[i])}(A^e_i) \to \Delta$, and symmetrically $A^e_{i+1} = \tau(A^s_i, m)$ for some $m : \pi(A^s_i) \to \Delta$. We say that a strategy f is *winning* if by resolving the existential choice in evaluating the formulas of the form $[\![\langle - \rangle]\!]_s \Phi$ and $[\![\langle - \rangle]\!]_w \Phi$ according to f, the goal formula is satisfied. Notably, model checking algorithms provide a *witness* of the checked property [20,31], which, in our case, consists of a *labeling* produced during the model checking process of the abstract DL2GS's. From *labelled* game states, one can read how the controller is meant to react to the environment in order to fulfill the formulas that *label* the state itself, and from this, define a strategy to fulfill the goal formula. It remains to lift such abstract strategy on the finite abstract DL2GS to the actual DL2GS. The abstract strategy $\bar{f}$ models a family of concrete strategies. Thus, in principle, in order to obtain an actual strategy, it would be sufficient *concretizing* $\bar{f}$ by replacing the abstract individuals and parameter assignments with concrete individuals and assignments that satisfy, step-by-step, the same equality commitments. While theoretically correct, this procedure cannot be realized in practice, as the resulting family of strategies is in general infinite. However, we adopt a lazy approach that allows us to generate and follow a concrete strategy, as the game progresses. Formally we have:

Theorem 4. *There exists an algorithm that, given a state-bounded, generic DL2G $\mathcal{K}$ and a μADL_p formula Φ, realizes a concrete strategy to force Φ.*

Proof (sketch). The algorithm iterates over three steps: *(i)* matching of the current concrete history λ with an abstract history $\bar{\lambda}$ over which $\bar{f}$ is defined; *(ii)* extraction of the action and corresponding abstract parameter assignment; *(iii)* concretization of the obtained parameter assignment. The first step requires building a family of functions, in fact bijections, that transform each pair of consecutive states of the concrete history, into a pair of abstract successor states, so as to satisfy the same equality commitments at the abstract and at the concrete level, and to guarantee that $\bar{f}$ is defined over the obtained abstract history $\bar{\lambda}$. This can be done as both λ and the abstract DL2GS contain, by state boundedness, only finitely many distinct elements, thus only finitely many such functions exist. The existence of $\bar{\lambda}$ is guaranteed by the bisimilarity, induced, in turn, by genericity. By applying $\bar{f}$ to the abstract history $\bar{\lambda}$, we extract the action (τ, π) and abstract parameter assignment $\bar{m}$ to execute next, in the abstract game, i.e., $((\tau, \pi), \bar{m}) = \bar{f}(\bar{\lambda})$. Finally, in order to concretize $\bar{m}$, it is sufficient to reconstruct the equality commitment H enforced by $\bar{m}$ and the bijection over the last pairs of states of λ and $\bar{\lambda}$, and then replacing the abstract values assigned by m with concrete ones, arbitrarily chosen, so as to satisfy H. By genericity, for any such choice, we obtain an action executable at the concrete level, after λ, and that is compatible with (at least) one strategy of the family defined by $\bar{f}$. In this way, at every step, the system is presented a set of choices, namely one per possible concretization of the abstract assignment, that can thus be resolved based on information available at runtime. □

Theorem 4 and its proof give us an effective method for actually synthesizing strategies to force the desired property. Obviously, optimizations for practical efficiency (which are out of the scope of this paper) require further study.

6 Instantiations of the Framework

As a concrete instantiation of the abstract framework introduced in Section 2, we consider a variant of Knowledge and Action Bases (KABs) [6], instantiated on the lightweight DL *DL-Lite* and the expressive DL $\mathcal{ALCQI}$. A *KAB* is a tuple $\mathcal{K} = (T, A_0, Act)$ where T and A_0 form the *knowledge base*, and Act is the *action base*. In practice, $\mathcal{K}$ is a stateful device that stores the information of interest into a KB, formed by a fixed TBox T and an initial ABox A_0, which evolves by executing actions in Act.

An *action* $\alpha \in Act$ modifies the current ABox A by adding or deleting assertions, thus generating a new ABox A'. α consists of a set $\{e_1, \ldots, e_n\}$ of effects, that take place simultaneously. An *effect* e_i has the form $Q_i \rightsquigarrow A'_i$, where

- Q_i is an ECQ, i.e., a domain independent first-order query whose atoms represent certain answers of unions of conjunctive queries (UCQs) [11].
- A'_i is a set of ABox assertions that include as terms: individuals in A_0, free variables of Q_i, and Skolem terms $f(\boldsymbol{x})$ having as arguments such free variables.

Given an action $\alpha \in Act$, we can capture its execution by defining an action $(\pi_\alpha, \tau_\alpha)$ as follows:

- $\pi_\alpha(A)$ returns the ground Skolem terms obtained by executing the queries Q_i of the effects over (T, A), instantiating the facts A'_i using the returned answers, and extracting the (ground) Skolem terms occurring therein.
- $\tau_\alpha(A, m)$ returns the ABox obtained by executing over (T, A) the queries Q_i of the effects, instantiating the facts A'_i using the returned answers, and assigning to the (ground) Skolem terms occurring therein values according to a freely chosen parameter assignment m, as long as such an ABox is consistent with T. If the ABox is not consistent with T, then $\tau_\alpha(A, m)$ is undefined.

In this way, we can define Γ from Act, and hence the DLDS $\mathcal{S} = (T, A_0, \Gamma)$ corresponding to the KAB $\mathcal{K} = (T, A_0, Act)$. Notice that, being the parameter assignment freely chosen, the resulting DLDS is actually generic.

In this context, we use ECQs also as the query language for the local properties in the verification and synthesis formalism. Now observe that if the TBox is expressed in a lightweight DL of the *DL-Lite* family, answering ECQ queries is PSPACE-complete in combined complexity (and in AC^0 in data complexity, i.e., the complexity measured in the size of the ABox only). The same complexity bounds hold for the construction of $\pi_\alpha(A)$ and $\tau_\alpha(A, m)$, and hence, under the assumption of state-boundedness, the abstract transition system generated by $\mathcal{S}$ can be constructed in EXPTIME. It follows that both verification and synthesis can be done in EXPTIME.

Instead, if the TBox is expressed in an expressive DL such as $\mathcal{ALCQI}$, the cost that dominates the construction of the abstract transition system is the 2EXPTIME cost of answering over (T, A) the UCQs that are the atoms of the EQL queries in the actions. Finally, if instead of using as atoms UCQs, we use atomic concepts and roles (i.e., we

do instance checking), the cost of query evaluation drops to EXPTIME, and so does the cost of building the abstract transition system.

These results can be extended to other DLs as long as they do not include nominals, given that the presence of nominals blurs the distinction between the extensional and the intensional knowledge, and hence requires more careful handling.

7 Conclusion

This work complements and generalizes two previous papers focussing on forms of verification on DL-based dynamics. One is [6] from which we took the formalism for the instantiations. The crucial difference is that in their framework they use Skolem terms to denote new values, which as a consequence remain unknown during the construction of the transition system, while we substitute these Skolem terms with actual values. Decidability of *weakly acyclic systems* is shown. The other one is [13], where a sort of light-weight DL-based dynamic was proposed. There, a semantic layer in *DL-Lite* is built on top of a data-aware process. The *DL-Lite* ontology plus mapping is our knowledge component, while the dynamic component (the actions) are induced by the process working directly on the data-layer. Exploiting *DL-Lite* first-order rewritability properties of conjunctive queries, the verification can be done directly on the data-aware process. Decidability of checking properties in μ-calculus *without quantification across* is shown for state-bounded data-aware process. In both, synthesis is not considered.

Acknowledgments. This research has been partially supported by the EU under the ICT Collaborative Project ACSI (Artifact-Centric Service Interoperation), grant agreement n. FP7-257593, and under the large-scale integrating project (IP) Optique (Scalable End-user Access to Big Data), grant agreement n. FP7-318338.

References

1. Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases. Addison Wesley Publ. Co. (1995)
2. Alur, R., Henzinger, T.A., Kupferman, O.: Alternating-time temporal logic. J. of the ACM 49(5), 672–713 (2002)
3. Artale, A., Franconi, E.: Temporal description logics. In: Gabbay, D., Fisher, M., Vila, L. (eds.) Handbook of Temporal Reasoning in Artificial Intelligence. Foundations of Artificial Intelligence, Elsevier (2005)
4. Baader, F., Ghilardi, S., Lutz, C.: LTL over description logic axioms. ACM Trans. on Computational Logic 13(3), 21:1–21:32 (2012)
5. 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/SIGART Symp. on Principles of Database Systems (PODS 2013) (2013), to appear
6. 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 (2013), to appear

7. Baier, C., Katoen, J.P., Guldstrand Larsen, K.: Principles of Model Checking. The MIT Press (2008)
8. Belardinelli, F., Lomuscio, A., Patrizi, F.: An abstraction technique for the verification of artifact-centric systems. In: Proc. of the 13th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR 2012). pp. 319–328 (2012)
9. Berardi, D., Calvanese, D., De Giacomo, G.: Reasoning on UML class diagrams. Artificial Intelligence 168(1–2), 70–118 (2005)
10. Bhattacharya, K., Gerede, C., Hull, R., Liu, R., Su, J.: Towards formal analysis of artifact-centric business process models. In: Proc. of the 5th Int. Conference on Business Process Management (BPM 2007). Lecture Notes in Computer Science, vol. 4714, pp. 288–234. Springer (2007)
11. 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 2007). pp. 274–279 (2007)
12. 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)
13. Calvanese, D., De Giacomo, G., Lembo, D., Montali, M., Santoso, A.: Ontology-based governance of data-aware processes. In: Proc. of the 6th Int. Conf. on Web Reasoning and Rule Systems (RR 2012). Lecture Notes in Computer Science, vol. 7497, pp. 25–41. Springer (2012)
14. Calvanese, D., De Giacomo, G., Lenzerini, M.: Conjunctive query containment and answering under description logics constraints. ACM Trans. on Computational Logic 9(3), 22.1–22.31 (2008)
15. Clarke, E.M., Grumberg, O., Peled, D.A.: Model checking. The MIT Press, Cambridge, MA, USA (1999)
16. De Giacomo, G., De Masellis, R., Rosati, R.: Verification of conjunctive artifact-centric services. Int. J. of Cooperative Information Systems 21(2), 111–139 (2012)
17. De Giacomo, G., Felli, P., Patrizi, F., Sardiña, S.: Two-player game structures for generalized planning and agent composition. In: Proc. of the 24th AAAI Conf. on Artificial Intelligence (AAAI 2010). pp. 297–302 (2010)
18. De Giacomo, G., Patrizi, F., Sardiña, S.: Automatic behavior composition synthesis. Artificial Intelligence 196, 106–142 (2013)
19. 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 2009). pp. 252–267 (2009)
20. Emerson, E.A.: Model checking and the Mu-calculus. In: Immerman, N., Kolaitis, P. (eds.) Proc. of the DIMACS Symposium on Descriptive Complexity and Finite Model. pp. 185–214. American Mathematical Society Press (1997)
21. Gabbay, D., Kurusz, A., Wolter, F., Zakharyaschev, M.: Many-dimensional Modal Logics: Theory and Applications. Elsevier Science Publishers (2003)
22. Ghallab, M., Nau, D.S., Traverso, P.: Automated planning – Theory and Practice. Elsevier (2004)
23. Gu, Y., Soutchanski, M.: A description logic based situation calculus. Ann. of Mathematics and Artificial Intelligence 58(1-2), 3–83 (2010)
24. Jamroga, W.: Concepts, agents, and coalitions in alternating time. In: Proc. of the 20th Eur. Conf. on Artificial Intelligence (ECAI 2012). pp. 438–443 (2012)
25. Levesque, H.J.: Foundations of a functional approach to knowledge representation. Artificial Intelligence 23, 155–212 (1984)
26. Levesque, H.J., Lakemeyer, G.: The Logic of Knowledge Bases. The MIT Press (2001)

27. Lin, F., Reiter, R.: State constraints revisited. J. of Logic Programming 4(5), 655–678 (1994)
28. Martin, D.L., Burstein, M.H., McDermott, D.V., McIlraith, S.A., Paolucci, M., Sycara, K.P., McGuinness, D.L., Sirin, E., Srinivasan, N.: Bringing semantics to web services with OWL-S. In: Proc. of the 16th Int. World Wide Web Conf. (WWW 2007). pp. 243–277 (2007)
29. Mazala, R.: Infinite games. In: Grädel, E., Thomas, W., Wilke, T. (eds.) Automata, Logics, and Infinite Games, Lecture Notes in Computer Science, vol. 2500, pp. 23–42. Springer (2002)
30. Park, D.M.R.: Finiteness is Mu-ineffable. Theoretical Computer Science 3(2), 173–181 (1976)
31. Piterman, N., Pnueli, A., Sa'ar, Y.: Synthesis of reactive(1) designs. In: Proc. of the 7th Int. Conf. on Verification, Model Checking, and Abstract Interpretation (VMCAI 2006). Lecture Notes in Computer Science, vol. 3855, pp. 364–380. Springer (2006)
32. Reiter, R.: Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems. The MIT Press (2001)
33. Schild, K.: Combining terminological logics with tense logic. In: Proc. of the 6th Portuguese Conf. on Artificial Intelligence (EPIA'93). Lecture Notes in Computer Science, vol. 727, pp. 105–120. Springer (1993)
34. Stirling, C.: Modal and Temporal Properties of Processes. Springer (2001)
35. Wolter, F., Zakharyaschev, M.: Temporalizing description logic. In: Gabbay, D., de Rijke, M. (eds.) Frontiers of Combining Systems, pp. 379–402. Studies Press/Wiley (1999)