Description Logic Based Dynamic Systems: Modeling, Verification, and Synthesis*†

Diego Calvanese,
Marco Montali, Fabio Patrizi
KRDB Research Centre
Free Univ. of Bozen-Bolzano
Bozen-Bolzano, Italy
lastname@inf.unibz.it

Giuseppe De Giacomo,
DIAG
Sapienza Univ. of Rome
Rome, Italy
degiacomo@dis.uniroma1.it

Abstract

In this paper, we overview the recently introduced general framework of Description Logic Based Dynamic Systems, which leverages Levesque's functional approach to model systems that evolve the extensional part of a description logic knowledge base by means of actions. This framework is parametric w.r.t. the adopted description logic and the progression mechanism. In this setting, we discuss verification and adversarial synthesis for specifications expressed in a variant of first-order μ-calculus, with a controlled form of quantification across successive states and present key decidability results under the natural assumption of state-boundedness.

1 Introduction

To attack the inherent complexity of organizations, static (data-related) and dynamic (process-related) aspects are traditionally modeled and managed independently from each other. This *divide et impera* approach has led to the development of successful theories and technologies, such as databases, ontologies and information integration to account for static aspects, and business process management, service-oriented computing, formal verification and model checking for dynamic ones.

On the other hand, in the last decade it has been extensively argued that this separation hampers the possibility of understanding the organization as a whole, and of taking corresponding strategic decisions [Karel *et al.*, 2009; Reichert, 2012]. E.g., to define resource assignment policies, a company might require that if an employee A has reviewed the job of B, then A cannot work in the same department as B. In this scenario, any simplification wrt to the static or dynamic component (the assignments of employees to departments and their evolution over time) leads to an incomplete model and prevents the possibility of fulfilling the policy requirements.

The combination of these two aspects has led to flourishing lines of research on formal foundations [Bhattacharya *et al.*, 2007; Calvanese *et al.*, 2013a], modeling paradigms [Martin *et al.*, 2007; Hull, 2008], and integrated software platforms [Künzle *et al.*, 2011] for *data-aware (business) processes*.

In this light, there has been an increasing interest in integrating semantic mechanisms like description logics (DLs), to describe static knowledge, with mechanisms, like transition systems (TS), to describe dynamics. Combining such aspects into a single logical formalism is notoriously difficult, as it yields a semantics based on two-dimensional structures (DL domain + dynamics/time) which leads to undecidability [Wolter and Zakharyaschev, 1999; Gabbay *et al.*, 2003].

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 [Baader *et al.*, 2012; De Giacomo *et al.*, 2012; Calvanese *et al.*, 2012; Bagheri Hariri *et al.*, 2013b]. Virtually all such work is implicitly or explicitly based on Levesque's functional approach [Levesque, 1984], 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 TS, whose states are DL KBs, and transitions are labeled by the action that causes the transition. The key issue in this context is that such TSs are *infinite* in general, and hence some form of faithful abstraction is needed. Note that, if the number of states in this TS is finite, then verifying dynamic properties over such systems amounts to a form a finite-state model checking [Emerson, 1996].

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

*This paper was invited for submission to the Best Papers From Sister Conferences Track, based on a paper that appeared in the 7th International Conference on Web Reasoning and Rule Systems (RR-2013).

†This research has been partially supported by the Provincia Autonoma di Bolzano – Alto Adige, under the project VeriSynCoPateD (*Verification and Synthesis from Components of Processes that Manipulate Data*), and by the EU, under the large-scale integrating project (IP) Optique (*Scalable End-user Access to Big Data*), grant agreement n. FP7-318338.

of synthesis problems, including conditional planning with full-observability [Ghallab *et al.*, 2004], behaviour composition [De Giacomo *et al.*, 2013], and several other sophisticated forms of synthesis and planning [De Giacomo *et al.*, 2010].

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 objects but their total number at each time point cannot exceed a certain bound. Notice that along an infinite run (and hence in the overall TS), the total numer of objects 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 [Alur *et al.*, 2002]. We show again decidability under the "bounded-state" assumption (this time for the game structure).

This paper recasts the results presented in [Calvanese *et al.*, 2013b] for a more general AI audience. We refer to that article for additional details.

2 Description Logic based Dynamic Systems

Description Logic based Dynamic Systems (DLDSs) [Calvanese *et al.*, 2013b] support a representation of the domain of interest in terms of a full-fledged DL KB, and describe how the extensional component of the KB evolves as actions are executed. The coupling of static and dynamic aspects is inspired by Levesque's functional approach [Levesque, 1984], which sees a KB as a device supporting two key operations: *(i)* ASK, which retrieves tuples of objects as answers to query over the KB; *(ii)* TELL, which produces a new KB resulting from an action application on the current KB. Notably, DLDSs introduce no requirement on the underlying DL, nor on the action specification formalism; thus, they can be concretized under different modeling choices, without affecting the obtained technical results, see, Sections 4 and 5. We next overview the components and the working assumptions for DLDSs.

Static Component

The static component of a DLDS consists of a DL KB, with objects coming from a countably infinite universe Δ acting as *standard names* [Levesque and Lakemeyer, 2001]. A (DL) *KB* is a pair (T, A), where T, the *TBox*, represents the intensional knowledge about the domain of interest, in terms of *concepts* (unary relations) and *roles* (binary relations), and A, the *ABox*, represents the extensional knowledge about the objects. The TBox is a finite set of universal assertions (not allowed to mention nominals). The ABox is a finite set of (ground) facts of the form $N(d)$ or $P(d, d')$, with N and P a concept and a role name, respectively, and d, d' coming from Δ. $\textsc{adom}(A)$ denotes the set of objects explicitly mentioned in A and $\mathcal{A}_T$ denotes the set of all possible ABoxes over concepts and roles from T, and objects from Δ. We adopt the standard FO semantics for DLs, and assume that TBoxes are always *satisfiable*,

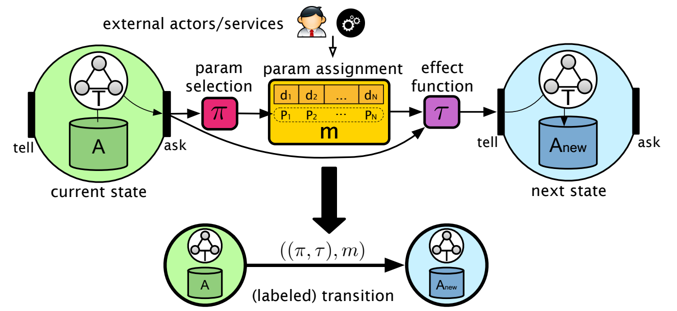

Figure 1: Action execution in a DLDS

i.e., admit at least one model. For the specification of the static component, we do not commit to any particular DL but assume, as typical in DL, that the standard reasoning services, i.e., *checking KB satisfiability* and *logical implication*, are decidable.

Dynamic Component

The dynamic component of a DLDS consists of a set of parametric actions that manipulate the ABox (the intensional knowledge is fixed). Actions are mathematically characterized in terms of their input/output behavior. An action execution consists of the following three steps, illustrated in Figure 1:

1. A query over the KB is issued to determine the formal input parameters accepted by the action in the current state (action parameters are not fixed but depend on the current state, i.e., the ABox).
2. The obtained formal parameters are bound to objects from Δ by the external environment –this makes it possible to inject fresh data into the DLDS.
3. The action generates a new ABox, which depends on the current ABox and the actual parameter values.

We stress that the parameter binding mechanism is more general than that of procedures/functions of programming languages, where the number of parameters is fixed by their signature. This data-driven mechanism is inspired by web-based systems, where input forms are dynamically constructed and customized depending on data acquired previously.

Example 1 In a conference submission system, when the corresponding author registers a paper, inserting the author names, the system creates a form to enter affiliations and e-mail addresses. The number of input fields of this form depends on the inserted data, i.e., the number of registered authors. The insertion of such data can be modeled in a DLDS action as follows: *(i)* the action queries the current KB, extracts the author names, and produces two input parameters, *affiliation* and *e-mail*, per author; *(ii)* the corresponding author binds the parameters to actual values; *(iii)* the action is executed, by adding to the current ABox, the affiliation data for each author. ■

Formal Definition

A *DLDS* is a tuple $\mathcal{K} = (T, A_0, \Gamma)$, where (T, A_0) is a KB and Γ is a finite set of *actions*. Constants in $\textsc{adom}(A_0)$ are *distinguished*, and are denoted by C, i.e., $C = \textsc{adom}(A_0)$. Let P be a countably infinite set of *parameter names*. Each action in Γ has the form (π, τ), where

- $\pi : \mathcal{A}_T \rightarrow 2^P$ 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);

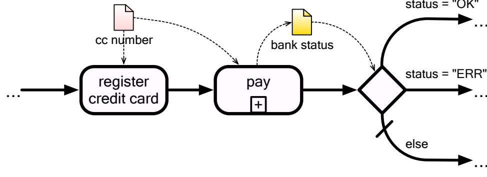

Figure 2: A BPMN fragment with control and generic data

- $\tau : \mathcal{A}_T \times \Delta^P \mapsto \mathcal{A}_T$, 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 $\textsc{adom}(A) \cup \textsc{im}(m)$ ($\textsc{im}(m)$ denotes m's *image*).

Note that only finitely many new objects can be added to A', depending on A, being $\pi(A)$ (and thus $\textsc{im}(m)$) finite.

Generic DLDSs

Observe that the definition of actions is completely general, as no assumptions are made on how they operate. In practice, though, it is natural to assume that actions are specified using some data-centric query/update languages. Virtually all such languages enjoy the property of *genericity* [Abiteboul *et al.*, 1995], i.e., they are invariant under renaming of objects. Essentially, genericity yields that objects are distinguishable based only on properties that can be inferred from the KB. More specifically, in a generic DLDS we can conceptually partition Δ into two sets: (i) a *finite* subset containing special *control data*, typically distinguished constants, used to drive the dynamics of the DLDS, and which constitute the "control state"; (ii) a countably infinite subset containing data that do not play a role *per se* in the DLDS dynamics.

Example 2 Consider the fragment of Business Process Model and Notation (BPMN) process shown in Figure 2, where a credit card registration is followed by a payment, in turn followed by three different branches, chosen depending on the payment outcome. For this process, the credit card number is *generic*, in that its *value* does not affect directly the process execution, specifically which branch is taken after payment. On the other hand, the bank status is a control data, in that its value determines the branch that is followed next. ■

To define genericity, we introduce a notion of logical equivalence between ABoxes weaker than the standard one, that does not depend on the actual values mentioned in the ABoxes. Informally, two ABoxes A_1, A_2 are said to be *logically equivalent modulo renaming* wrt a TBox T, if one can rename the objects mentioned in A_1 and A_2, preserving the distinguished constants, in a way such that the assertions of one ABox are entailed by the other in conjunction with the TBox T. This notion between ABoxes generalizes that of isomorphism between FO interpretations, in that object renaming allows one to obtain the model defined by one ABox, from the other.

We define a *generic* DLDS $\mathcal{K} = (T, A_0, \Gamma)$ as one such that, given two ABoxes A_1, A_2 logically equivalent modulo renaming, the execution of the same action on both ABoxes, using the same parameters modulo the renaming between A_1 and A_2, possibly extended to the new injected objects, produces two ABoxes logically equivalent modulo renaming.

Example 3 Consider a DLDS modeling the BPMN diagram in Figure 2, and focus on the formalization of the credit card registration action. This action always generates a single parameter, used to inject the credit card number into the system, and has the effect of inserting a new fact in the KB, expressing that the customer who executed the action is now associated with the inserted card number. Suppose now that the DLDS is instantiated twice by different users, Alice (A) and Bob (B), and that, before executing the registration action, the two current KBs of each process instance are logically equivalent modulo renaming of A as B (and viceversa). Then, genericity guarantees that the KBs resulting from the execution of the registration action in each process instance are equivalent modulo renaming of A as B, and of A's credit card number into B's. ■

Execution Semantics

The execution semantics of a DLDS $\mathcal{K}$ is characterized by a *transition system* (TS) $\Upsilon_\mathcal{K}$ that accounts for all possible action executions, with all possible parameter instantiations. To account for the evolution of the extensional part of a DL KB, $\Upsilon_\mathcal{K}$ incorporates Δ and the KB TBox T, and annotates each state with the corresponding ABox. Transitions in $\Upsilon_\mathcal{K}$ then correspond to the result of action executions on ABoxes.

The *generated* TS of a DLDS $\mathcal{K} = (T, A_0, \Gamma)$, is defined as $\Upsilon_\mathcal{K} = (\Delta, T, \Sigma, s_0, abox, \Rightarrow)$, where: *(i)* $abox$ is the identity function (states correspond to ABoxes); *(ii)* $s_0 = A_0$; *(iii)* $\Rightarrow \subseteq \Sigma \times \Sigma$ is a transition relation; *(iv)* 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 $A \Rightarrow A'$. Figure 1 depicts the link between the execution of instantiated actions of $\mathcal{K}$ and transitions of $\Upsilon_\mathcal{K}$.

3 Specification Logic μDL_p

This work concerns verification and synthesis of dynamic properties (over DLDSs), which we specify using a FO variant of the μ-calculus [Stirling, 2001; Park, 1976], called μDL_p. In μDL_p, local properties are queries over DL KBs, expressed in any language for which query entailment for DL KBs is decidable [Calvanese *et al.*, 2007; 2008].

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 as specified by v. Notice that the set of v such that $(T, A) \models Qv$ are the so-called *certain answers*. A query of interest is $\textsc{live}(x)$, whose answers correspond exactly to the objects of the active domain of the ABox the query is applied to. $\textsc{live}(x)$ can be expressed as a union of atomic queries expressing that x is member of a concept of T, or participates in a binary relation of T. We abbreviate $\bigwedge_{i \in \{1,\ldots,n\}} \textsc{live}(x_i)$ as $\textsc{live}(x_1, \ldots, x_n)$.

By leveraging on [Bagheri Hariri *et al.*, 2013a], μDL_p employs $\textsc{live}$ queries to control the effect of FO quantification across states. In particular, μDL_p forces quantification to track only the evolution of objects that persist in the active domain, while the evaluation of a formula with objects that disappear from the active domain immediately leads to false or true. Formally, a μDL_p formula Φ is built as follows:

$$\Phi ::= Q \mid \neg\Phi \mid \Phi_1 \wedge \Phi_2 \mid \exists x.\textsc{live}(x) \wedge \Phi \mid \textsc{live}(\vec{x}) \wedge \langle - \rangle \Phi \mid \textsc{live}(\vec{x}) \wedge [-] \Phi \mid Z \mid \mu Z.\Phi,$$

where Q is a (possibly open) query as described above, in which the only constants that may appear are those in C,

and Z is a second order predicate variable (of arity 0). We use standard abbreviations to define $\vee$ and $\rightarrow$, including: $\text{LIVE}(\vec{x}) \rightarrow \langle - \rangle \Phi = \neg(\text{LIVE}(\vec{x}) \wedge [-]\neg\Phi)$ and $\text{LIVE}(\vec{x}) \rightarrow [-]\Phi = \neg(\text{LIVE}(\vec{x}) \wedge \langle - \rangle \neg\Phi)$.

The semantics of μDL_p formulae over a DLDS $\mathcal{K}$ is defined over the (possibly infinite) TS generated by $\mathcal{K}$, i.e., $\Upsilon_{\mathcal{K}}$, and defines the set of $\Upsilon_{\mathcal{K}}$ states that satisfy each formula. For the detailed semantics, we refer the reader to [Calvanese *et al.*, 2013b]. Here, we discuss only some examples.

Example 4 Consider Example 3. Assume that the TBox contains: *(i)* the concepts $Customer$, $Open$, $Closed$, respectively modeling customers and pending and closed orders; and *(ii)* the roles $Owns$ and $Paid$, where $Owns(c, o)$ (resp., $Paid(c, o)$) means that customer c owns (resp., paid for) order o. The μDL_p formula:

$$\nu Z.(\forall c, o.Owns(c, o) \rightarrow \mu Y. (Paid(c, o) \vee (\text{LIVE}(c, o) \wedge \langle - \rangle Y))) \wedge [-]Z$$

expresses that it must always be the case that, whenever customer c owns o, there exists a run of the DLDS in which c and o persist in the active domain, until o is paid by c. Formula

$$\nu Z.(\forall o.Open(c, o) \rightarrow \mu Y. (Closed(o) \vee (\text{LIVE}(o) \rightarrow \langle - \rangle Y))) \wedge [-]Z$$

instead expresses that it must always be the case that, for every open order, either the order disappears from the active domain, or becomes eventually closed. The two properties show the two different forms of FO quantification over time supported by μDL_p. ■

A fundamental property of μDL_p is that, analogously to standard μ-calculus, it enjoys invariance under *bisimulation*. This is not the standard notion of bisimulation, but one taking into account the added richness of the TSs used here –where states are labeled by ABoxes rather than simple propositions– and preserving the same co-inductive structure.

Intuitively, two states s_1 and s_2 of Υ_1 and Υ_2, respectively, are *persistence-preserving bisimilar* under a renaming h, if:

1. s_1 and s_2 are logically equivalent modulo the object renaming h;
2. for each possible successor s'_1 of s_1, there exists a corresponding successor s'_2 of s_2 such that s'_1 and s'_2 are persistence-preserving bisimilar under a renaming h' that preserves the renamings of h on $\text{ADOM}(abox(s_1))$;
3. for each possible successor s'_2 of s_2, there exists a corresponding successor s'_1 of s_1 such that s'_1 and s'_2 are persistence-preserving bisimilar under a renaming h' that preserves the renamings of h on $\text{ADOM}(abox(s_1))$;

We say that Υ_1 is persistence-preserving bisimilar to Υ_2, written $\Upsilon_1 \sim \Upsilon_2$, if the respective initial states are so, under some renaming h. We have the following result, analogous in spirit to that in [Bagheri Hariri *et al.*, 2013a]:

Theorem 3.1 *Consider two TSs 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 $\Upsilon_2 \models \Phi$.*

This result is particularly useful when holding between an infinite and a finite TS. In this case, indeed, one can check whether Φ holds by operating on the finite TS instead of the infinite one. We exploit this fundamental implication in the next sections.

4 Verification

Given a DLDS $\mathcal{K}$ and a closed μDL_p formula Φ, we call *model checking* (MC) the problem of verifying whether Φ holds in the initial state of $\Upsilon_{\mathcal{K}}$. This is the first problem we address. Observe that MC is interesting only if Δ is infinite; indeed, for finite Δ, only finitely many ABoxes –i.e., possible states in the generated TS– exist and, by quantifier elimination, one can rewrite any μDL_p formula into a propositional one, thus reducing the problem to model checking of propositional μ-calculus (against finite-state TS), which is decidable [Emerson, 1996]. On the other hand, for infinite Δ, it is easy to show, by reduction from the halting problem, that the problem is in general undecidable, even under the assumption of genericity (see, e.g., [Bagheri Hariri *et al.*, 2013b]). Thus, the question arises whether decidable (and interesting) classes of problem instances exist. Our work presents a positive answer for a particular class of generic DLDSs, which we call *state-bounded*.

A *state-bounded* DLDS $\mathcal{K}$ is one for which there exists a finite bound b s.t., for each state s of $\Upsilon_{\mathcal{K}}$, $|\text{ADOM}(abox(s))| < b$. When this requirement holds, $\mathcal{K}$ is said to be *b-bounded*. Intuitively, in a b-bounded DLDS, the labeling of any state of $\Upsilon_{\mathcal{K}}$ mentions at most b distinct constants. Notice that state-bounded DLDSs may contain infinitely many states, and that a DLDS $\mathcal{K}$ can be state-unbounded even if, for every state s of $\Upsilon_{\mathcal{K}}$, $|\text{ADOM}(abox(s))|$ 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.[1]

Decidability of model checking for μDL_p formulas against state-bounded, generic DLDSs can be proven by reduction to model checking against DLDSs with finite Δ. In this case, as discussed above, the verification task reduces to model checking of propositional μ-calculus over finite-state TSs. The crux of the proof is the construction of a finite-state TS $\Upsilon_{\mathcal{K}}^D$ s.t. $\Upsilon_{\mathcal{K}}^D \sim \Upsilon_{\mathcal{K}}$, where $\Upsilon_{\mathcal{K}}$ is the TS generated by $\mathcal{K}$. This is done through an abstraction technique inspired by that of [Bagheri Hariri *et al.*, 2013a; Belardinelli *et al.*, 2014]; here, however, we deal with DL knowledge bases, instead of relational databases.

Essentially, $\Upsilon_{\mathcal{K}}^D$ is a finite fragment of $\Upsilon_{\mathcal{K}}$, inductively obtained by "executing" $\mathcal{K}$ over a *finite* set of objects $D \subset \Delta$, until no new ABoxes are generated. More precisely, the initial state is expanded by executing all possible actions over D, thus producing new states, over which the process can be iterated. The finiteness of D guarantees that, at every step, only finitely many actions are executed, and that eventually no new states can be generated. As a consequence, the obtained TS is finite.

To guarantee that $\Upsilon_{\mathcal{K}}^D$ is bisimilar to $\Upsilon_{\mathcal{K}}$, the size of D and the objects it contains must be carefully chosen so that no μDL_p formula is able to distinguish $\Upsilon_{\mathcal{K}}^D$ from $\Upsilon_{\mathcal{K}}$. To this end, D is required to contain:

- at least b objects other than those in C, to cover the case when the facts entailed by an ABox (and the TBox) mention up to b objects different from those in C;
- at least n additional objects, distinct from those above and those in C, accounting for the new objects possibly

[1] n 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.

introduced by the last action executed –it can be proven that, in a bounded DLDS, the new objects introduced by a transition can only be objects bound to action parameters, which are at most n;
- the objects in C, accounting for the distinguished constants possibly occurring in μDL_p formulas.

Under these assumptions, the outlined construction produces a *faithful* finite abstraction of $\Upsilon_{\mathcal{K}}$, i.e., a finite TS indistinguishable from $\Upsilon_{\mathcal{K}}^{D}$, to μDL_p formulas:

Lemma 4.1 *If* $|D| \geq b+n+|C|$, *then* $\Upsilon_{\mathcal{K}}^{D} \sim \Upsilon_{\mathcal{K}}$.

Notice that, as a consequence of genericity, the particular choice of the objects in D –except, obviously, those coming from C– does not play any role in guaranteeing the equivalence of $\Upsilon_{\mathcal{K}}^{D}$ and $\Upsilon_{\mathcal{K}}$, wrt verification of μDL_p formulas.

We observe that the outlined construction uses b to build D, before constructing $\Upsilon_{\mathcal{K}}^{D}$. An alternative strategy that does not need to know b in advance, consists in maximizing the reuse of the previously introduced objects. In this approach new objects are added as new states are generated, only if needed, avoiding the generation of a new state whenever one logically equivalent (modulo renaming) has already been generated. A construction along this line, is proposed, in a different setting, in [Bagheri Hariri *et al.*, 2013a].

Lemma 4.1 implies the following decidability result:

Theorem 4.2 *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 TS whose number of states is at most exponential in the size of $\mathcal{K}$.*

Proof (sketch). By Lemma 4.1, $\Upsilon_{\mathcal{K}}^{D}$ is bisimilar to $\Upsilon_{\mathcal{K}}$. Further, it can be shown that $\Upsilon_{\mathcal{K}}^{D}$ contains at most an exponential number of states in the size of the specification $\mathcal{K}$. Hence, by Theorem 3.1, one can check any μDL_p formula Φ against the finite TS $\Upsilon_{\mathcal{K}}^{D}$ instead of $\Upsilon_{\mathcal{K}}$. Since the number of objects occurring in $\Upsilon_{\mathcal{K}}^{D}$ is finite, Φ can be propositionalized and checked through standard algorithms [Emerson, 1996]. □

5 Adversarial Synthesis

Another interesting problem addressed in our work is that of *adversarial synthesis*. Consider a setting where two agents act in turn as adversaries. One agent, called *environment*, acts autonomously, while the other, called *system*, can be controlled. When the agents interact by alternating their moves (the environment moves first), their joint behavior defines a so-called *two-player game structure* (2GS) [De Giacomo *et al.*, 2010; Bloem *et al.*, 2012; Mazala, 2002], which is essentially the *arena* of a game, i.e., the set of rules defining how players can move. On top of the 2GS one can express, using some variant of μ-calculus, the *goal* of the game, which captures a desired property that the system should enforce, in spite of how the environment behaves. Given a 2GS and a goal specification, the adversarial synthesis problem requires to synthesize a *strategy* for the system, i.e., a suitable refined behavior, guaranteeing that, no matter how the environment plays, the game evolution fulfills the goal.

Many problems can be phrased as adversarial synthesis. These include *conditional planning in nondeterministic fully observable domains* [Ghallab *et al.*, 2004], *behavior composition* [De Giacomo *et al.*, 2013], and advanced forms of synthesis and planning, see, e.g., [De Giacomo *et al.*, 2010].

To deal with adversarial synthesis, we build a 2GS that encodes explicitly the alternation of player moves. This is done using a fresh concept name Turn and two fresh distinguished constants t_e, t_s, whose (mutually exclusive) presence in Turn indicates which player the turn is next. We denote with $\mathcal{A}_T^e$ (resp. $\mathcal{A}_T^s$) the set of ABoxes, i.e., states, where the environment (system) is to move next.

A *DL-based 2GS* (*DL2GS*) is a DLDS $\mathcal{K} = (T, A_0, \Gamma)$, where $A_0 \in \mathcal{A}_T^e$, i.e., the environment moves first, 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 and the system responds, iterating on these steps, possibly forever.

Logics of interest for 2GSs are temporal logics in the style of ATL [Alur *et al.*, 2002], where "next Φ" expresses that "the system can force Φ" to hold in the next state, by suitably responding to any move available to the environment. We use the following specialization of μDL_p, called μADL_p:

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

where $[\langle - \rangle]_s \Phi$ stands for $[-](\textsc{live}(\vec{x}) \wedge \langle - \rangle \Phi)$, and $[\langle - \rangle]_w \Phi$ stands for $[-](\textsc{live}(\vec{x}) \rightarrow \langle - \rangle \Phi)$.

Technically, μADL_p is a fragment of μDL_p, where negation normal form is enforced, i.e., negation is allowed to appear only in front of atoms (in our case, queries Q). Intuitively, both $[\langle - \rangle]_s \Phi$ and $[\langle - \rangle]_w \Phi$ express that the system can force Φ to hold, but $[\langle - \rangle]_s$ precludes the environment from dropping objects existing before its move, while $[\langle - \rangle]_w$ does not.

As an immediate consequence of Theorem 4.2, one can check whether the system has a strategy that enforces a μADL_p goal over a DL2GS:

Theorem 5.1 *Checking μADL_p formulas over state-bounded, generic DL2GS's is decidable.*

Notice that this result does not provide indications on whether or how a strategy can be built. We next focus on this issue, discussing how one can actually fulfill a μADL_p goal, by *synthesizing* and executing a strategy.

Intuitively, a *strategy* is a function f that, given a *history* of a DL2GS, i.e., a sequence of ABoxes stemming from the alternation of environment and system actions and terminating in an ABox A^s where the system is to move, returns an action for the system that is executable in A^s. A strategy f is said to be *winning* (for the system) 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.

Model checking algorithms provide a *witness* of the checked property [Emerson, 1996; Bloem *et al.*, 2012], which, in our

case, consists of a *labeling* produced during the model checking process of the abstract DL2GS constructed as outlined in the previous section –which is, ultimately, a $\mathcal{K}$. From the labeling of each state one can obtain a set of possible replies to the environment move, that allow one to fulfill the formulas that *label* the state itself; from these replies, a strategy to fulfill the goal formula can be constructed.

The approach discussed above is effective in producing a strategy for the *abstract* DL2GS, while we are interested in one that can be used in the original (concrete) structure. This gap can be filled by *lifting* the abstract strategy to the original DL2GS.

In fact, the abstract strategy $\bar{f}$ models a family of concrete strategies. Thus, in principle, in order to obtain a strategy executable on the concrete system, one could simply *concretize* $\bar{f}$ by appropriately replacing the abstract objects and parameter assignments with concrete objects and assignments that satisfy, step-by-step, the same mutual equalities, both at the abstract and at the concrete level. While theoretically correct, though, this procedure cannot be realized in practice, as the resulting family of strategies is in general infinite. Nonetheless, one can adopt a *lazy* approach where the concrete strategy is generated from the abstract one, as the game progresses.

Theorem 5.2 *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.*

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 an abstract history $\bar{\lambda}$ that is state-wise bisimilar to the concrete one λ. This can be done as the histories and the abstract DL2GS contain, by state-boundedness, only finitely many distinct elements, thus there exist only finitely many candidate abstract histories. The existence of $\bar{\lambda}$ is guaranteed by bisimilarity, in turn guaranteed by genericity. The second step consists in extracting the action (τ, π) and the abstract parameter assignment $\bar{m}$ by applying $\bar{f}$ $\bar{\lambda}$. The last step requires to concretize $\bar{m}$. To this end, it is sufficient to reconstruct the mutual equalities on the parameter values 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 such equalities. 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, one is presented a set of choices, one per possible concretization of the abstract assignment, that can thus be resolved lazily, at runtime.

6 Instantiating the Framework

To instantiate the framework of generic DLDSs, one needs to fix: *(i)* the DL used to describe its static component, *(ii)* the query language used to ASK information in the states of the DLDS, and *(iii)* a concrete action specification formalism, used to substantiate the TELL operation, guaranteeing at the same time that the concrete framework so obtained enjoys genericity. This is the case, for example, of Knowledge and Action Bases (KABs) [Bagheri Hariri *et al.*, 2013b], which have been indeed embedded into generic DLDSs in [Calvanese *et al.*, 2013b]. Instantiating such components allows us to actually implement DLDSs. It also allows us to study the complexity of verification and synthesis when the system is state-bounded. This is characterized by the cost of constructing the abstract TS, which in turn depends on *(i)* the complexity of query answering, *(ii)* the complexity of updating the ABox when an action is applied, and *(iii)* the overall complexity of verification and synthesis in terms of the TS size.

The complexity of query answering obviously depends on the chosen DL and query language. For example, KABs employ a lightweight DL of the *DL-Lite* family, and rely on *EQL-Lite(UCQ)* queries, i.e., domain independent first-order queries whose atoms compute certain answers of unions of conjunctive queries [Calvanese *et al.*, 2007]. In this setting, query answering 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 complexity of update varies widely based on the chosen update operators. In the case of KABs, where updates are performed by querying the current state and asserting and retracting a number of facts that is polynomially related to the answers, the update is polynomial. Consequently, the abstract transition system generated by the DLDS can be constructed in EXPTIME, yielding an EXPTIME upper bound for both verification and synthesis.

If we adopt the same KAB framework but the TBox is expressed using a more expressive DL such as $\mathcal{ALCQI}$, the complexity that dominates the construction of the abstract transition system is the 2EXPTIME cost of answering the UCQs that are the atoms of the EQL queries employed by the DLDS actions. Complexity drops back to EXPTIME if instead of UCQs, only atomic concepts and roles are used.

7 Conclusion

This work complements and generalizes two previous papers on forms of verification on DL-based dynamics. One is [Bagheri Hariri *et al.*, 2013b] from which we took the formalism for the instantiations. In that work, Skolem terms are used to denote new objects, which, as a consequence, remain unknown during the construction of the transition system, while here we substitute Skolem terms with actual objects. The other one is [Calvanese *et al.*, 2012], 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) is 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 state* is shown for state-bounded data-aware process. In both, synthesis is not considered.

References

[Abiteboul *et al.*, 1995] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison Wesley Publ. Co., 1995.

[Alur *et al.*, 2002] Rajeev Alur, Thomas A. Henzinger, and Orna Kupferman. Alternating-time temporal logic. *JACM*, 49(5):672–713, 2002.

[Baader *et al.*, 2012] Franz Baader, Silvio Ghilardi, and Carsten Lutz. LTL over description logic axioms. *ACM TOCL*, 13(3):21:1–21:32, 2012.

[Bagheri Hariri *et al.*, 2013a] Babak Bagheri Hariri, Diego Calvanese, Giuseppe De Giacomo, Alin Deutsch, and Marco Montali. Verification of relational data-centric dynamic systems with external services. In *Proc. of PODS*, 2013.

[Bagheri Hariri *et al.*, 2013b] Babak Bagheri Hariri, Diego Calvanese, Marco Montali, Giuseppe De Giacomo, Riccardo De Masellis, and Paolo Felli. Description logic Knowledge and Action Bases. *JAIR*, 46:651–686, 2013.

[Belardinelli *et al.*, 2014] Francesco Belardinelli, Alessio Lomuscio, and Fabio Patrizi. Verification of agent-based artifact systems. *JAIR*, 51:333–376, 2014.

[Bhattacharya *et al.*, 2007] K. Bhattacharya, C. Gerede, R. Hull, R. Liu, and J. Su. Towards formal analysis of artifact-centric business process models. In *Proc. of BPM*, 2007.

[Bloem *et al.*, 2012] Roderick Bloem, Barbara Jobstmann, Nir Piterman, Amir Pnueli, and Yaniv Sa'ar. Synthesis of reactive(1) designs. *JCSS*, 78(3):911–938, 2012.

[Calvanese *et al.*, 2007] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati. EQL-Lite: Effective first-order query processing in description logics. In *Proc. of IJCAI*, 2007.

[Calvanese *et al.*, 2008] Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. Conjunctive query containment and answering under description logics constraints. *ACM TOCL*, 9(3):22.1–22.31, 2008.

[Calvanese *et al.*, 2012] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Marco Montali, and Ario Santoso. Ontology-based governance of data-aware processes. In *Proc. of RR*, 2012.

[Calvanese *et al.*, 2013a] Diego Calvanese, Giuseppe De Giacomo, and Marco Montali. Foundations of data aware process analysis: A database theory perspective. In *Proc. of PODS*, 2013.

[Calvanese *et al.*, 2013b] Diego Calvanese, Giuseppe De Giacomo, Marco Montali, and Fabio Patrizi. Verification and synthesis in description logic based dynamic systems. In *Proc. of RR*, 2013.

[De Giacomo *et al.*, 2010] Giuseppe De Giacomo, Paolo Felli, Fabio Patrizi, and Sebastian Sardiña. Two-player game structures for generalized planning and agent composition. In *Proc. of AAAI*, 2010.

[De Giacomo *et al.*, 2012] Giuseppe De Giacomo, Riccardo De Masellis, and Riccardo Rosati. Verification of conjunctive artifact-centric services. *Int. J. of Cooperative Information Systems*, 21(2):111–139, 2012.

[De Giacomo *et al.*, 2013] Giuseppe De Giacomo, Fabio Patrizi, and Sebastian Sardiña. Automatic behavior composition synthesis. *AIJ*, 196:106–142, 2013.

[Emerson, 1996] E. Allen Emerson. Model checking and the Mu-calculus. In *Proc. of the DIMACS Symposium on Descriptive Complexity and Finite Models*, 1996.

[Gabbay *et al.*, 2003] Dov Gabbay, Agnes Kurusz, Frank Wolter, and Michael Zakharyaschev. *Many-dimensional Modal Logics: Theory and Applications*. Elsevier, 2003.

[Ghallab *et al.*, 2004] Malik Ghallab, Dana S. Nau, and Paolo Traverso. *Automated planning – Theory and Practice*. Elsevier, 2004.

[Hull, 2008] Richard Hull. Artifact-centric business process models: Brief survey of research results and challenges. In *Proc. of ODBASE*, 2008.

[Karel *et al.*, 2009] Rob Karel, Clay Richardson, and Connie Moore. Warning: Don't assume your business processes use master data – Synchronize your business process and master data strategies. Report, Forrester, September 2009.

[Künzle *et al.*, 2011] Vera Künzle, Barbara Weber, and Manfred Reichert. Object-aware business processes: Fundamental requirements and their support in existing approaches. *Int. J. of Information System Modeling and Design*, 2(2):19–46, 2011.

[Levesque and Lakemeyer, 2001] Hector J. Levesque and Gerhard Lakemeyer. *The Logic of Knowledge Bases*. The MIT Press, 2001.

[Levesque, 1984] Hector J. Levesque. Foundations of a functional approach to knowledge representation. *AIJ*, 23:155–212, 1984.

[Martin *et al.*, 2007] David L. Martin, Mark H. Burstein, Drew V. McDermott, Sheila A. McIlraith, Massimo Paolucci, Katia P. Sycara, Deborah L. McGuinness, Evren Sirin, and Naveen Srinivasan. Bringing semantics to web services with OWL-S. In *Proc. of WWW*, 2007.

[Mazala, 2002] René Mazala. Infinite games. In *Automata, Logics, and Infinite Games*, volume 2500 of *LNCS*. Springer, 2002.

[Park, 1976] David Michael Ritchie Park. Finiteness is Mu-ineffable. *TCS*, 3(2):173–181, 1976.

[Reichert, 2012] Manfred Reichert. Process and data: Two sides of the same coin? In *Proc. of OTM*, 2012.

[Stirling, 2001] Colin Stirling. *Modal and Temporal Properties of Processes*. Springer, 2001.

[Wolter and Zakharyaschev, 1999] Frank Wolter and Michael Zakharyaschev. Temporalizing description logic. In *Frontiers of Combining Systems*. Studies Press/Wiley, 1999.