Managing Change in Graph-structured Data Using Description Logics

SHQIPONJA AHMETAJ, TU Wien
DIEGO CALVANESE, Free University of Bozen-Bolzano
MAGDALENA ORTIZ and MANTAS ŠIMKUS, TU Wien

In this paper, we consider the setting of graph-structured data (GSD) that evolves as a result of operations carried out by users or applications. We study different reasoning problems, which range from deciding whether a given sequence of actions preserves the satisfaction of a given set of integrity constraints, for every possible initial data instance, to deciding the (non-)existence of a sequence of actions that would take the data to an (un)desirable state, starting either from a specific data instance or from an incomplete description of it. For describing states of the data instances and expressing integrity constraints on them, we use Description Logics (DLs) closely related to the two-variable fragment of first-order logic with counting quantifiers. The updates are defined as *actions* in a simple yet flexible language, as finite sequences of conditional insertions and deletions, which allow one to use complex DL formulas to select the (pairs of) nodes for which (node or arc) labels are added or deleted. We formalize the above data management problems as a static verification problem and several planning problems and show that, due to the adequate choice of formalisms for describing actions and states of the data, most of these data management problems can be effectively reduced to the (un)satisfiability of suitable formulas in decidable logical formalisms. Leveraging this, we provide algorithms and tight complexity bounds for the formalized problems, both for expressive DLs and for a variant of the popular DL-Lite, advocated for data management in recent years.

CCS Concepts: •**Theory of computation** → **Logic and verification; Description logics;** *Logic and databases; Pre- and post-conditions;* •**Information systems** → *Graph-based database models;*

Additional Key Words and Phrases: Integrity Constraints, Graph-structured Data, Static Analysis, Planning

ACM Reference Format:
Shqiponja Ahmetaj, Diego Calvanese, Magdalena Ortiz, Mantas Šimkus, 2017. Managing Change in Graph-structured Data Using Description Logics. *ACM Trans. Comput. Logic* 18, 4, Article 27 (October 2017), 35 pages.
DOI: http://doi.acm.org/10.1145/3143803

1. INTRODUCTION

The complex structure and increasing size of information that has to be managed in today's applications calls for flexible mechanisms for storing such information, making it easily and efficiently accessible, and facilitating its change and evolution over time. While technologies for data storage and access have not ceased to evolve and mature on the solid theoretical basis of the principles of data management, these technologies often disregard the fundamental fact that data is intrinsically dynamic and evolves constantly, mainly as the results of updates carried out by users and applications. Tools for *automated reasoning about the evolution of data, and the preservation of its integrity over time* remain in their infancy. The urgent need for such tools has become increasingly clear, and the development of tools for reasoning about evolving data has in fact been recognized as one of the central research challenges that must be

This research has been partially supported by FWF projects T515, P25518, P30360, and W1255, by WWTF project ICT12-015, by EU IP Project Optique FP7-318338, and by the Wolfgang Pauli Institute.
Author's addresses: Shqiponja Ahmetaj, TU Wien, Austria, email: ahmetaj@dbai.tuwien.ac.at; Diego Calvanese, Free University of Bozen-Bolzano, Italy, email: calvanese@inf.unibz.it; Magdalena Ortiz, TU Wien, Austria, email: ortiz@kr.tuwien.ac.at; Mantas Šimkus, TU Wien, Austria, email: simkus@dbai.tuwien.ac.at.

faced by the Foundations of Data Management community in the next years [Arenas et al. 2016]. This has become the main goal of significant research efforts. For example, in the area of data-centric dynamic processes, many database-aware verification techniques have been developed [Calvanese et al. 2013a; Deutsch et al. 2014].

In this paper, we also aim at developing tools that allow one to reason about evolving data, and in particular, to ensure the correctness and integrity preservation of databases that are modified and updated. Our focus is on the paradigm of *graph structured data* (GSD) [Sakr and Pardede 2011], which has roots in work done in the early 1990s (see, e.g., [Consens and Mendelzon 1990]), and has gained popularity recently as an alternative to traditional relational databases that provides more flexibility and thus can overcome the limitations of an a priori imposed rigid structure on the data. Indeed, differently from relational data, GSD do not require a schema to be fixed a priori. This flexibility makes them well suited for many emerging application areas such as managing Web data, information integration, persistent storage in object-oriented software development, or management of scientific data. Concrete examples of models for GSD are RDFS [Brickley and Guha 2004], object-oriented data models [Hull and King 1987; Abiteboul and Kanellakis 1989], and XML [Bray et al. 1998].

It is widely recognized that GSD are often dynamic, and problems like evaluating queries over evolving GSD have been studied [Muñoz et al. 2016]. However, reasoning about the effects that changes on GSD have on the integrity of data and on the properties the data satisfy, remains largely unexplored. As we will see here, focusing on GSD allows us to leverage results from powerful languages for knowledge representation, which can describe complex properties of GSD instances, and still have decidable satisfiability problems. In this way, we can go beyond the task of simply *model checking a given initial data instance* that is common in dynamic data-centric systems, and instead consider more challenging problems, like testing whether *for every possible data instance*, some given property is guaranteed to hold after applying some given updates.

The languages we leverage in this paper are *Description Logics* (DLs) [Baader et al. 2003], a well-established family of formalisms for knowledge representation that are particularly well-suited for describing properties of GSD. Indeed, in GSD, information is represented by means of a node and edge labeled graph, in which the labels convey semantic information. The representation structures underlying most DLs are paradigmatic examples of GSD: in DLs, a domain of interest is modeled by means of unary relations (a.k.a. *concepts*) and binary relations (a.k.a. *roles*), and hence the first-order interpretations of a DL knowledge base (KB) can be viewed as node and edge labeled graphs. DLs have been advocated as a proper tool for data management [Lenzerini 2011], and are very natural for describing complex knowledge about domains represented as GSD. A DL KB comprises an assertional component, called *ABox*, which is often viewed as a possibly incomplete instance of GSD, and a logical theory, called terminology or *TBox*, which can be used to infer implicit information from the assertions in the ABox. An alternative possibility is to view the *finite* structures over which DLs are interpreted as (complete) GSD, and the KB as a description of constraints and properties of the data. Taking this view, DLs have been applied, for example, for the static analysis of traditional data models, such as UML class diagrams [Berardi et al. 2005; Balaban and Maraee 2013] and Entity Relationship schemata [Calvanese et al. 1999; Artale et al. 2007]. Problems such as the consistency of a diagram are reduced to KB (or TBox) satisfiability in a suitable DL, and DL reasoning services become tools for data management.

In this paper, we follow the latter view, but aim at using DLs not only for static reasoning about data models, but also for reasoning about the evolution and change over time of GSD that happens as the result of executing actions. The development of automated tools to support such tasks is becoming a pressing problem, given the large amounts and complexity of GSD currently available. Having tools to understand the properties and effects of actions is important and provides added value for many purposes, including application development, integrity preservation, security, and optimization. Questions of interest are, e.g.,: *(i)* Will the execution of a given action *preserve* the integrity constraints, for every initial data instance? *(ii)* Is there a sequence of actions that leads a given data instance into a state where some property (either desired or not) holds? *(iii)* Does a given sequence of actions lead every possible initial data instance into a state where some property necessarily holds? The first question is analogous to a classic problem in relational databases: verifying *consistency* of database transactions. The second and third questions are classic questions in AI (called *planning* and *projection*, respectively). In this paper we address these and other related questions, develop tools to answer them, and characterize the computational complexity of the underlying problems. Our results are quite general and allow for analyzing data evolution in several practically relevant settings, including RDF data under constraints expressed in RDFS or OWL. Via the standard reification technique [Calvanese et al. 1999; Berardi et al. 2005], they also apply to the more traditional setting of relational data under schemas expressed in conceptual models (e.g., ER schemas, or UML class diagrams), or to object-oriented data.

Our contributions are organized as follows:

— In Section 2, we propose a very expressive DL that is suitable for reasoning about evolving GSD. In particular, we aim at a DL that is good for the following purposes: *(i)* modeling rich constraints on GSD, which express sophisticated domain knowledge, *(ii)* specifying conditions on data states that should be reached or avoided, *(iii)* specifying actions to evolve GSD over time, and *(iv)* expressing *weakest pre-conditions* that capture the effect of action sequences on data states. While standard DLs could be used for *(i)* and *(ii)*, their use for *(iii)* and *(iv)* is to our knowledge novel and calls for some less frequent constructs, which we introduce in the logic $\mathcal{ALCHOIQ}br$. This DL is an extension of the well known $\mathcal{ALCHOIQ}$ and, as we will see, is closely related to the two-variable fragment of first-order logic with counting quantifiers.

— Next, in Section 3, we introduce a simple yet powerful language in which actions are finite sequences of (possibly conditional) insertions and deletions performed on concepts and roles, using complex concepts and roles in $\mathcal{ALCHOIQ}br$ as queries. We define the syntax and semantics of this language, and give some examples illustrating its suitability for describing changes on GSD.

— Our main technical tool is developed in Section 4. From a DL KB expressing a set of constraints and a sequence of actions, we obtain a new KB that is satisfied by a GSD instance exactly when the result of executing the actions on it leads to a model of the constraints. It can be seen as a computation of the *weakest pre-conditions* for the constraints and the actions, and allows one to express the effects of actions within the syntax of DLs. The technique for obtaining the new KB is similar in spirit to *regression* in reasoning about actions [Levesque et al. 1997]. This is the core stepping stone of our results, as it allows us to reduce reasoning about evolving GSD to reasoning about traditional, static, DL KBs.

— In Section 5 we address the *static verification problem*, that is, the problem of verifying whether for every possible state satisfying a given set of constraints (i.e., a given KB), the constraints are still satisfied in the state resulting from the execution of a given (complex) action. Using the technique of Section 4 we are able to reduce static verification to satisfiability of DL KBs, showing that the former problem is decidable, and obtaining tight complexity bounds for it, using two different DLs in the constraints and actions. Specifically, we provide a tight coNEXPTIME bound for the considered expressive DL, and a tight coNP bound for a variation of *DL-Lite* [Calvanese et al. 2007].

— In Section 6 we then study different variants of *planning*. We define a plan as a sequence of actions that leads a given structure into a state where some property (either desired or not) holds. Then we study problems such as deciding the existence of a plan, both for the case where the initial structure is fully known, and where only a partial description of it is available, and deciding whether a given sequence of actions is always a plan for some goal. Since the existence of a plan (of unbounded length) is undecidable in general, even for lightweight DLs and restricted actions, we also study plans of bounded length. We provide tight complexity bounds for different variants of the problem, both for lightweight and for expressive DLs.

A detailed discussion of related work is given in Section 7, and conclusions in Section 8.

This paper extends the results by Ahmetaj et al. [2014] (and a preliminary informal publication [Calvanese et al. 2013b]) with proofs, extended examples, and complete definitions that were omitted from the conference version. We also obtain some new complexity results. In particular, in Section 2.3 we discuss some more expressive languages to which our upper bounds apply, and in Section 2.5 we introduce a fragment of $\mathcal{ALCHOIQ}br$ for which we obtain better upper bounds for the considered reasoning problems (namely, ExpTime vs. (co)NExpTime for static verification and some variants of planning). A new discussion of the detailed relationship between the main DL we consider, and other better known DLs, as well as the two-variable fragment of first-order logic with counting quantifiers, was added as an appendix to this version.

2. AN EXPRESSIVE DL FOR REASONING ABOUT EVOLVING GSD

The use of DLs in this paper is manifold. On the one hand, DLs act as constraint languages for data instances, and describe properties of data states that are to be ensured or avoided as data evolves. On the other hand, we employ them as the basis of a simple action language to update data instances, and as tools for reasoning about the evolution of data by capturing weakest pre-conditions on action sequences and data states. The former use matches closely the typical scenario where DLs are used to model rich domain knowledge and describe properties of data instances, and standard DLs are suitable for this purpose. The latter uses are less standard, and call for some expressive features not always supported by common DLs.

2.1. $\mathcal{ALCHOIQ}$ as a Constraint Language for GSD

To express constraints on GSD we choose the expressive DL $\mathcal{ALCHOIQ}$. It is an extension of the basic DL $\mathcal{ALC}$ that supports role hierarchies, inverses, and number restrictions, all of which are useful for expressing constraints on data instances. It also supports *nominals*, which additionally to allowing one to refer to specific individuals in constraints, also play an important role in updating GSD and reasoning about the effects of updates, as we will see below.

$\mathcal{ALCHOIQ}$ Syntax and Semantics. We recall the standard definition of $\mathcal{ALCHOIQ}$, summarized in the left column of Table I. Countably infinite sets $\mathsf{N_R}$ of *role names*, $\mathsf{N_C}$ of *concept names*, and $\mathsf{N_I}$ of *individual names* are used to build complex *concepts* and *roles*. Intuitively, concepts describe *classes* of objects, and *roles* binary relations between objects. These expressions are then used in *inclusions* and *assertions*. The former express general dependencies to be satisfied by data instances, while the latter assert the membership of specific (pairs of) individuals in concepts and roles. A *knowl-*

Table I. Syntax of $\mathcal{ALCHOIQ}$ and $\mathcal{ALCHOIQ}br$.

$\mathcal{ALCHOIQ}$	$\mathcal{ALCHOIQ}br$	
Roles $r_{(i)}$ and concepts $C_{(i)}$:		
$r \longrightarrow p \mid p^-$	$r \longrightarrow p \mid r^- \mid \{(t_1,t_2)\} \mid r_1 \cup r_2 \mid r_1 \setminus r_2 \mid r_1 \cap r_2 \mid r{\upharpoonright_C} \mid r{\downharpoonright_C}$	
$C \longrightarrow A \mid \top \mid \bot \mid \{o\} \mid C_1 \sqcap C_2 \mid C_1 \sqcup C_2 \mid \neg C \mid \exists r.C \mid \forall r.C \mid \leqslant n\, r.C \mid \geqslant n\, r.C$	$C \longrightarrow A \mid \top \mid \bot \mid \{t\} \mid C_1 \sqcap C_2 \mid C_1 \sqcup C_2 \mid \neg C \mid \exists r.C \mid \forall r.C \mid \leqslant n\, r.C \mid \geqslant n\, r.C$	
where $p \in \mathsf{N_R}$, $A \in \mathsf{N_C}$, $o \in \mathsf{N_I}$, $\{t, t_1, t_2\} \subseteq \mathsf{N_I} \cup \mathsf{N_V}$, and $n \geq 0$ is an integer.		
Assertions and inclusions:		
$C_1 \sqsubseteq C_2$	$C_1 \sqsubseteq C_2$	concept inclusion
$r_1 \sqsubseteq r_2$	$r_1 \sqsubseteq r_2$	role inclusion
$o : C$	$t : C$	concept assertion
$(o_1, o_2) : r$	$(t_1, t_2) : r$	role assertion
where C_1, C_2 are concepts, r_1, r_2 are roles, $\{o, o_1, o_2\} \subseteq \mathsf{N_I}$, and $\{t, t_1, t_2\} \subseteq \mathsf{N_I} \cup \mathsf{N_V}$. Concepts, roles, inclusions, and assertions with no variables are called *ordinary*.		
KBs:	Formulas: (KBs are formulas with no variables)	
$\mathcal{K} \longrightarrow \alpha \mid \mathcal{K}_1 \wedge \mathcal{K}_2$	$\mathcal{K} \longrightarrow \alpha \mid \mathcal{K}_1 \wedge \mathcal{K}_2 \mid \mathcal{K}_1 \vee \mathcal{K}_2 \mid \dot{\neg}\mathcal{K}_1$	
where α denotes a (concept or role) inclusion or a (concept or role) assertion.		

edge base (KB) is a conjunction of inclusions and assertions.[1] In the following, we use $C_1 \equiv C_2$ as a shortcut for the conjunction $(C_1 \sqsubseteq C_2) \wedge (C_2 \sqsubseteq C_1)$ of two concept inclusions; similarly for role inclusions.

Semantics. The semantics of DL KBs is usually defined in terms of *interpretations*, defined as relational structures over a (possibly infinite) non-empty domain and a signature consisting of unary predicates (the concept names), binary predicates (the role names), and constants (the individuals). Such an interpretation is called *finite* if its domain is finite. In turn, GSD instances are often defined as relational databases where all relations are unary or binary, that is, the same kind of relational structures as DL interpretations. Although infinite GSD instances are sometimes of interest, here we focus on the more common setting of *finite* ones, which are in direct correspondence with finite DL interpretations. We define next the semantics of $\mathcal{ALCHOIQ}$ using the standard notion of an interpretation, which in what follows we call a *(GSD) instance*.

Definition 2.1. An *instance* is a pair $\mathcal{I} = \langle \Delta^{\mathcal{I}}, \cdot^{\mathcal{I}} \rangle$ where $\Delta^{\mathcal{I}} \neq \emptyset$ is the *domain*, $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$ for each $A \in \mathsf{N_C}$, $p^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ for each $p \in \mathsf{N_R}$, and $o^{\mathcal{I}} \in \Delta^{\mathcal{I}}$ for each $o \in \mathsf{N_I}$. The function $\cdot^{\mathcal{I}}$ is extended to all $\mathcal{ALCHOIQ}$ concepts and roles as usual, see Table II.

Consider an instance $\mathcal{I}$. For an inclusion $\alpha_1 \sqsubseteq \alpha_2$, $\mathcal{I}$ *satisfies* $\alpha_1 \sqsubseteq \alpha_2$, written $\mathcal{I} \models \alpha_1 \sqsubseteq \alpha_2$, if $\alpha_1^{\mathcal{I}} \subseteq \alpha_2^{\mathcal{I}}$. For an assertion β of the form $o : C$ (resp., $(o_1, o_2) : r$), $\mathcal{I}$ *satisfies* β, in symbols $\mathcal{I} \models \beta$, if $o^{\mathcal{I}} \in C^{\mathcal{I}}$ (resp., $(o_1^{\mathcal{I}}, o_2^{\mathcal{I}}) \in r^{\mathcal{I}}$). The notion of satisfaction extends naturally to KBs: $\mathcal{I} \models \mathcal{K}_1 \wedge \mathcal{K}_2$ if $\mathcal{I} \models \mathcal{K}_1$ and $\mathcal{I} \models \mathcal{K}_2$. If $\mathcal{I} \models \mathcal{K}$, then $\mathcal{I}$ is a *model* of $\mathcal{K}$. $\lhd$

[1]Here we depart from the standard conventions and define a KB as a single conjunction, rather than as a pair containing a set of inclusions, called TBox, and a set of assertions, the ABox. The two definitions are equivalent.

Table II. Semantics of $\mathcal{ALCHOIQ}$ and $\mathcal{ALCHOIQ}br$.

Concept constructors in $\mathcal{ALCHOIQ}$:		
$\top^{\mathcal{I}}$	$=$	$\Delta^{\mathcal{I}}$
$\bot^{\mathcal{I}}$	$=$	$\emptyset$
$(\neg C_1)^{\mathcal{I}}$	$=$	$\Delta^{\mathcal{I}} \setminus C_1^{\mathcal{I}}$
$(C_1 \sqcap C_2)^{\mathcal{I}}$	$=$	$C_1^{\mathcal{I}} \cap C_2^{\mathcal{I}}$
$(C_1 \sqcup C_2)^{\mathcal{I}}$	$=$	$C_1^{\mathcal{I}} \cup C_2^{\mathcal{I}}$
$(\exists r.C)^{\mathcal{I}}$	$=$	$\{e_1 \mid \text{ for some } e_2 \in \Delta^{\mathcal{I}}, (e_1, e_2) \in r^{\mathcal{I}} \text{ and } e_2 \in C^{\mathcal{I}}\}$
$(\forall r.C)^{\mathcal{I}}$	$=$	$\{e_1 \mid \text{ for all } e_2 \in \Delta^{\mathcal{I}}, (e_1, e_2) \in r^{\mathcal{I}} \text{ implies } e_2 \in C^{\mathcal{I}}\}$
$(\leqslant n\, r.C)^{\mathcal{I}}$	$=$	$\{e_1 \mid \lvert\{e_2 \mid (e_1, e_2) \in r^{\mathcal{I}} \text{ and } e_2 \in C^{\mathcal{I}}\}\rvert \leq n\}$
$(\geqslant n\, r.C)^{\mathcal{I}}$	$=$	$\{e_1 \mid \lvert\{e_2 \mid (e_1, e_2) \in r^{\mathcal{I}} \text{ and } e_2 \in C^{\mathcal{I}}\}\rvert \geq n\}$
Role constructors in $\mathcal{ALCHOIQ}$:		
$(r^-)^{\mathcal{I}}$	$=$	$\{(e_1, e_2) \mid (e_2, e_1) \in r^{\mathcal{I}}\}$
Additional role constructors in $\mathcal{ALCHOIQ}br$:		
$\{(o_1, o_2)\}^{\mathcal{I}}$	$=$	$\{(o_1^{\mathcal{I}}, o_2^{\mathcal{I}})\}$
$(r_1 \cup r_2)^{\mathcal{I}}$	$=$	$r_1^{\mathcal{I}} \cup r_2^{\mathcal{I}}$
$(r_1 \setminus r_2)^{\mathcal{I}}$	$=$	$r_1^{\mathcal{I}} \setminus r_2^{\mathcal{I}}$
$(r_1 \cap r_2)^{\mathcal{I}}$	$=$	$r_1^{\mathcal{I}} \cap r_2^{\mathcal{I}}$
$(r\restriction_C)^{\mathcal{I}}$	$=$	$\{(e_1, e_2) \in r^{\mathcal{I}} \mid e_1 \in C^{\mathcal{I}}\}$
$(r\vert_C)^{\mathcal{I}}$	$=$	$\{(e_1, e_2) \in r^{\mathcal{I}} \mid e_2 \in C^{\mathcal{I}}\}$

C, C_1, C_2 denote concepts, r, r_1, r_2 roles, and $\{o_1, o_2\} \subseteq \mathsf{N_I}$.

Next, we provide an example of useful constraints expressible in $\mathcal{ALCHOIQ}$.

Example 2.2. Suppose that some institution wants to ensure the following constraints on its database; (1) projects and departments are disjoint; projects are active projects or finished projects; (2) hasMember, hasHead, and hasLeader are respectively the inverses of isMemberOf, isHeadOf, and isLeaderOf (this allows us to speak naturally about the relations in both directions); (3) the head of a department is also a member of it; similarly, the leader of a project also works for that project; (4) the domain of worksFor are employees, and its range are projects; (5) similarly, the domain of hasMember are departments, and its range are employees; (6) a project employee is an employee who works for an active project; all employees must be project employees or members of a department; (7) departments have exactly one department head; and (8) all active projects must have a project leader who is a member of some department.
The constraints are captured by the KB below:

$$(\mathsf{Prj} \sqcap \mathsf{Dept} \sqsubseteq \bot) \land (\mathsf{Prj} \equiv \mathsf{ActivePrj} \sqcup \mathsf{FinishedPrj}) \land \tag{1}$$

$$(\mathsf{hasMember} \equiv \mathsf{isMemberOf}^-) \land (\mathsf{hasHead} \equiv \mathsf{isHeadOf}^-) \land (\mathsf{hasLeader} \equiv \mathsf{isLeaderOf}^-) \land \tag{2}$$

$$(\mathsf{hasHead} \sqsubseteq \mathsf{hasMember}) \land (\mathsf{isLeaderOf} \sqsubseteq \mathsf{worksFor}) \land \tag{3}$$

$$(\exists \mathsf{worksFor}.\top \sqsubseteq \mathsf{Empl}) \land (\top \sqsubseteq \forall \mathsf{worksFor}.\mathsf{Prj}) \land \tag{4}$$

$$(\exists \mathsf{hasMember}.\top \sqsubseteq \mathsf{Dept}) \land (\top \sqsubseteq \forall \mathsf{hasMember}.\mathsf{Empl}) \land \tag{5}$$

$$(\mathsf{PrjEmpl} \equiv \exists \mathsf{worksFor}.\mathsf{ActivePrj}) \land (\mathsf{Empl} \sqsubseteq \mathsf{PrjEmpl} \sqcup \exists \mathsf{isMemberOf}.\mathsf{Dept}) \land \tag{6}$$

$$(\mathsf{Dept} \sqsubseteq \exists \mathsf{hasHead}.\top) \land (\mathsf{Dept} \sqsubseteq \leqslant 1\ \mathsf{hasHead}.\top) \land \tag{7}$$

$$(\mathsf{ActivePrj} \sqsubseteq \exists \mathsf{hasLeader}.(\exists \mathsf{isMemberOf}.\mathsf{Dept})) \tag{8}$$

2.2. Extending $\mathcal{ALCHOIQ}$ for Reasoning About Evolving GSD

To make $\mathcal{ALCHOIQ}$ better suitable for reasoning about evolving GSD, we extend it with a construct $\{(o_1, o_2)\}$ for a singleton role, with union and difference of roles, and with role constructs that restrict the domain and the range of a role to some concept, sometimes called *domain restriction* and *range restriction* respectively. We also allow for Boolean KBs, and we allow for *variables* to occur in the place of individuals.

Syntax. Additionally to countably infinite sets of $\mathsf{N_R}$, $\mathsf{N_C}$, and $\mathsf{N_I}$, we consider a countably infinite set $\mathsf{N_V}$ of *variables*; all these sets are pairwise disjoint. The syntax of $\mathcal{ALCHOIQ}br$ is summarized in the right column of Table I.

Note that variables can occur in all places where individuals are usually allowed. Concepts, roles, inclusions, and assertions with no variables are called *ordinary*. Note that knowledge bases and formulas are defined as *Boolean* combinations of inclusions and assertions, rather than just their (positive) conjunctions, as usual in the DL literature. We use the traditional DL term *knowledge base* for the case when no variables occur in the expression, and the term *formula* for its generalization allowing also for variables.

Semantics. We now give semantics to $\mathcal{ALCHOIQ}br$ KBs (that is, where no variables occur). Formulas (with variables) will only take meaning later, when we use them in the action language defined in Section 3.

Definition 2.3. Consider an instance $\mathcal{I}$. The function $\cdot^{\mathcal{I}}$ is extended to the additional constructs in $\mathcal{ALCHOIQ}br$ as given in Table II. The semantics of $\mathcal{ALCHOIQ}$ presented in Definition 2.1 carries over to $\mathcal{ALCHOIQ}br$ ordinary inclusions and assertions and is extended to $\mathcal{ALCHOIQ}br$ KBs as follows:

$$\begin{array}{lll} \mathcal{I} \models \mathcal{K}_1 \wedge \mathcal{K}_2 & \text{if} & \mathcal{I} \models \mathcal{K}_1 \text{ and } \mathcal{I} \models \mathcal{K}_2; \\ \mathcal{I} \models \mathcal{K}_1 \vee \mathcal{K}_2 & \text{if} & \mathcal{I} \models \mathcal{K}_1 \text{ or } \mathcal{I} \models \mathcal{K}_2; \\ \mathcal{I} \models \dot{\neg}\mathcal{K} & \text{if} & \mathcal{I} \not\models \mathcal{K}. \end{array} \qquad \lhd$$

Finite Satisfiability Problem. We are interested in the problem of effectively managing GSD satisfying the constraints given by a DL KB $\mathcal{K}$, that is, finite GSD instances (interpretations) that are models of $\mathcal{K}$. Hence, the following problem is relevant.

Definition 2.4. The *finite satisfiability* (resp., *unsatisfiability*) *problem* is to decide, given a KB $\mathcal{K}$, if there exists (resp., doesn't exist) a model $\mathcal{I}$ of $\mathcal{K}$ with $\Delta^{\mathcal{I}}$ finite. $\lhd$

We note that $\mathcal{ALCHOIQ}br$ does not enjoy the *finite model property*. That is, it is possible to write a set of *infinity axioms* that are satisfiable but do not have any finite model. This applies already to weaker DLs that also support inverse roles and number restrictions, see e.g., [Calvanese 1996; Lutz et al. 2005].

In the next sections, we will tackle a range of problems related to evolving GSD by reducing them to finite satisfiability in $\mathcal{ALCHOIQ}br$. Please keep in mind that modelhood is only defined for KBs (that is, with no variables), and that in the finite satisfiability problem the input is such a KB.

$\mathcal{ALCHOIQ}br$ and Other DLs. Although some features of $\mathcal{ALCHOIQ}br$ may not seem common, it can be seen as a rather standard DL with some 'syntactic sugar'. In fact, KBs in $\mathcal{ALCHOIQ}br$ extend KBs in $\mathcal{ALCHOIQ}$ only with role union and role difference, two of the so called *safe Boolean role constructs*. We show in the appendix how the remaining constructs can be simulated. However, we will see in the following sections that adding them explicitly to the syntax will be useful for describing actions and reasoning about their effects.

2.3. $\mathcal{ALCHOIQ}br$ and $\mathcal{C}^2$

We note that we have defined $\mathcal{ALCHOIQ}br$ by adding to a well-known and sufficiently expressive DL the additional features necessary for reasoning about evolving GSD. But we could just as well use a more expressive DL that is still captured by $\mathcal{C}^2$, the two variable fragment of first-order predicate logic extended with counting quantifiers. For example, in the appendix we define $\mathcal{ALBOQ}^{id}$, which adds number restrictions to the DL $\mathcal{ALBO}^{id}$, which is known to have the same expressiveness as the two-variable fragment of first-order logic with equality.

The crucial property of $\mathcal{ALCHOIQ}br$ is that there exists a polynomial translation from KBs in $\mathcal{ALCHOIQ}br$ to sentences in $\mathcal{C}^2$ that preserves (finite) models. This is what we use to establish Theorem 2.6 below, on which we rely for the upper bounds that follow. We thus obtain the same upper bounds for other logics that share this property. For example, we can use $\mathcal{C}^2$ itself as a constraint language, or the extension of $\mathcal{ALCHOIQ}br$ with the combination of the constructs expressible in $\mathcal{ALBOQ}^{id}$ that we show in the appendix (see Table IV).

Remark 2.5. All the upper bounds given in this article, hold under the assumption that the numbers in number restrictions may be coded in binary, and apply to the following cases:

— When $\mathcal{ALCHOIQ}br$ KBs, concepts, and roles are respectively replaced by KBs, concepts, and roles in $\mathcal{ALBOQ}^{id}$.
— When $\mathcal{ALCHOIQ}br$ KBs are replaced by $\mathcal{C}^2$ sentences, while concepts and roles are respectively replaced by $\mathcal{C}^2$ formulas with one and two free variables.

2.4. Complexity of Reasoning in $\mathcal{ALCHOIQ}br$

The features of $\mathcal{ALCHOIQ}br$ that we have added to $\mathcal{ALCHOIQ}$ have no impact on the worst-case complexity of the finite satisfiability problem, which is the decision problem we rely on in this paper. Indeed, deciding the existence of a finite model for a KB in our extended logic has the same computational complexity as for plain $\mathcal{ALCHOIQ}$, as established by the following result.

THEOREM 2.6. *Finite satisfiability of $\mathcal{ALCHOIQ}br$ KBs is* NEXPTIME-*complete.*

PROOF (SKETCH). A NEXPTIME lower bound for finite satisfiability in $\mathcal{ALCHOIQ}br$ follows from the work of Tobies [2000], who showed NEXPTIME-hardness for unrestricted (not necessarily finite) satisfiability of KBs in the DL $\mathcal{ALCOIQ}$, the fragment of $\mathcal{ALCHOIQ}$ that disallows role inclusions. An inspection of the proof shows that the KB used in the hardness reduction is such that if it is satisfiable, it admits a finite model. Hence, the result holds for finite satisfiability as well. A matching upper bound follows from the translation into $\mathcal{C}^2$ (see the Appendix), and the NEXPTIME upper bound for finite satisfiability of $\mathcal{C}^2$ formulas established by Pratt-Hartmann [2005]. □

2.5. $\mathcal{ALCHOI}br$, an EXPTIME Fragment

$\mathcal{ALCHOIQ}$, the standard DL we have chosen as a basis in this paper, is known to have a NEXPTIME-hard satisfiability problem. A natural question that arises is whether we could obtain better upper bounds by considering suitable fragments. For instance, it is well-known that for the DLs $\mathcal{ALCHOI}$, $\mathcal{ALCHIQ}$, and $\mathcal{ALCHOQ}$, which are obtained from $\mathcal{ALCHOIQ}$ by disallowing respectively number restrictions ($\mathcal{Q}$), nominals ($\mathcal{O}$), and inverse roles ($\mathcal{I}$), KB satisfiability is feasible in EXPTIME. Can we obtain better upper bounds if we consider the extensions of these logics with the constructs from Section 2.2?

In the first case, the answer is positive. Let $\mathcal{ALCHOI}br$ be the DL that is defined just like $\mathcal{ALCHOIQ}br$, except that concepts of the forms $\leqslant n\, r.C$ and $\geqslant n\, r.C$ are not allowed in the syntax. $\mathcal{ALCHOQ}br$ and $\mathcal{ALCHIQ}br$ are defined analogously: the former disallows roles of the form r^-, and the latter concepts of the form $\{t\}$ and roles of the form $\{(t,t')\}$. We get the following result.

PROPOSITION 2.7. *Finite satisfiability of $\mathcal{ALCHOI}br$ KBs is* EXPTIME*-complete.*

PROOF. EXPTIME hardness holds already for KB satisfiability in $\mathcal{ALC}$ [Schild 1991]. The EXPTIME membership follows from the translation of $\mathcal{ALCHOI}br$ to $\mathcal{GF}^2$ with an unlimited number of constants, where $\mathcal{GF}^2$ is the guarded fragment of first order logic with two variables. Note that the translation in Table III of an $\mathcal{ALCHOI}br$ KB yields a $\mathcal{GF}^2$ sentence with constants. Indeed, deciding (unrestricted) satisfiability in this fragment was shown to be in EXPTIME by ten Cate and Franceschet [2005]. Since $\mathcal{GF}^2$ with constants is contained in the fragment of first order logic with two variables $\mathcal{L}^2$ (with equality), which was shown to have the finite model property [Mortimer 1975], it follows that finite satisfiability for $\mathcal{ALCHOI}br$ KBs collapses to unrestricted satisfiability. □

For $\mathcal{ALCHOQ}br$, EXPTIME-membership of (unrestricted) satisfiability can also be inferred from the literature, since it is a sublogic of $\mathcal{ZOQ}$ [Calvanese et al. 2009]. To our knowledge such a result for finite satisfiability has not been shown, although we conjecture that $\mathcal{ALCHOQ}br$ has the finite model property. If this conjecture is correct, all the results for static verification and planning we establish below for $\mathcal{ALCHOI}br$ would also hold for $\mathcal{ALCHOQ}br$.

An analogous result to Proposition 2.7 holds for $\mathcal{ALCHIQ}br$, even though it does not enjoy the finite model property [Calvanese 1996]. Indeed, finite satisfiability in $\mathcal{GC}^2$, the *guarded* fragment of $\mathcal{C}^2$, is decidable in EXPTIME [Pratt-Hartmann 2005], and the standard translation in Table III (see the Appendix) yields a $\mathcal{GC}^2$ sentence when the input is an $\mathcal{ALCHIQ}br$ KB (see also [Kazakov 2004]). However, this fragment of lower complexity is of limited use to us, since it does not support all the constructs we argued for in Section 2.2. In particular, it does not have nominals, which our techniques rely on. Hence the results we give for $\mathcal{ALCHOI}br$ do not directly extend to $\mathcal{ALCHIQ}br$.

3. UPDATING GRAPH STRUCTURED DATA

We now define an action language for manipulating GSD instances. The basic actions allow one to insert or delete all individuals from the extension of a concept, and all pairs of individuals from the extension of a role. The candidates for additions and deletions are instances of complex $\mathcal{ALCHOIQ}br$ concepts and roles. Since our DL supports nominals $\{o\}$ and singleton roles $\{(o,o')\}$, our actions allow us also to add/remove a single individual to/from a concept, or a pair of individuals to/from a role. We allow also for action composition and conditional actions. Note that the action language introduced here is a slight generalization of the one by Calvanese et al. [2013b].

Definition 3.1 (*Action language*). *Basic actions* β are defined by the grammar:

$$\beta \longrightarrow (A \oplus C) \mid (A \ominus C) \mid (p \oplus r) \mid (p \ominus r),$$

where A is a concept name, C is an arbitrary concept, p is a role name, and r is an arbitrary role. Then *(complex) actions* are defined by the following grammar:

$$\alpha \longrightarrow \varepsilon \mid \beta \cdot \alpha \mid (\mathcal{K}\,?\,\alpha[\![\alpha]\!]) \cdot \alpha$$

where β is a basic action, $\mathcal{K}$ is an arbitrary $\mathcal{ALCHOIQ}br$-formula, and ε denotes the *empty action*.

A *substitution* is a function σ from $\mathsf{N_V}$ to $\mathsf{N_I}$. For a formula, an action, or an action sequence γ, we use $\sigma(\gamma)$ to denote the result of replacing in γ every occurrence of a variable x by the individual $\sigma(x)$. An action α' is *ground* if it has no variables, and α' is a *ground instance* of an action α if $\alpha' = \sigma(\alpha)$ for some substitution σ. $\triangleleft$

Intuitively, an application of an action $(A \oplus C)$ on an instance $\mathcal{I}$ stands for the addition of the content of $C^{\mathcal{I}}$ to $A^{\mathcal{I}}$. Similarly, $(A \ominus C)$ stands for the removal of $C^{\mathcal{I}}$ from $A^{\mathcal{I}}$. The two operations can also be performed on extensions of roles. Composition stands for successive action execution, and a conditional action $\mathcal{K}\,?\,\alpha_1[\![\alpha_2]\!]$ expresses that α_1 is executed if the instance is a model of $\mathcal{K}$, and α_2 is executed otherwise. If $\alpha_2 = \varepsilon$ then we have an action with a simple *pre-condition* as in classical planning languages, and we write it as $\mathcal{K}\,?\,\alpha_1$, omitting α_2.

Formally, the semantics of actions is defined as follows:

Definition 3.2. Consider an instance $\mathcal{I}$ and let E be a concept or role name. If E is a concept, let $W \subseteq \Delta^{\mathcal{I}}$, and if E is a role, let $W \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$. Then, $\mathcal{I} \oplus_E W$ (resp., $\mathcal{I} \ominus_E W$) denotes the instance $\mathcal{I}'$ such that $\Delta^{\mathcal{I}'} = \Delta^{\mathcal{I}}$, and
- $E^{\mathcal{I}'} = E^{\mathcal{I}} \cup W$ (resp., $E^{\mathcal{I}'} = E^{\mathcal{I}} \setminus W$), and
- $E_1^{\mathcal{I}'} = E_1^{\mathcal{I}}$, for all symbols $E_1 \neq E$.

Given a ground action α, we define a mapping S_α from instances to instances as follows:

$$S_\varepsilon(\mathcal{I}) = \mathcal{I} \qquad \begin{aligned} S_{(A\oplus C)\cdot\alpha}(\mathcal{I}) &= S_\alpha(\mathcal{I} \oplus_A C^{\mathcal{I}}) \\ S_{(A\ominus C)\cdot\alpha}(\mathcal{I}) &= S_\alpha(\mathcal{I} \ominus_A C^{\mathcal{I}}) \end{aligned} \qquad \begin{aligned} S_{(p\oplus r)\cdot\alpha}(\mathcal{I}) &= S_\alpha(\mathcal{I} \oplus_p r^{\mathcal{I}}) \\ S_{(p\ominus r)\cdot\alpha}(\mathcal{I}) &= S_\alpha(\mathcal{I} \ominus_p r^{\mathcal{I}}) \end{aligned}$$

$$S_{(\mathcal{K}\,?\,\alpha_1[\![\alpha_2]\!])\cdot\alpha}(\mathcal{I}) = \begin{cases} S_{\alpha_1\cdot\alpha}(\mathcal{I}), & \text{if } \mathcal{I} \models \mathcal{K}, \\ S_{\alpha_2\cdot\alpha}(\mathcal{I}), & \text{if } \mathcal{I} \not\models \mathcal{K}. \end{cases}$$ $\triangleleft$

Example 3.3. The following instance $\mathcal{I}_1$ represents (part of) the project database of some research institute. There are two active projects, and there are three employees that work in the active projects.

$$\begin{aligned} \mathsf{Prj}^{\mathcal{I}_1} &= \{p_1, p_2\}, & \mathsf{ActivePrj}^{\mathcal{I}_1} &= \{p_1, p_2\}, \\ \mathsf{Empl}^{\mathcal{I}_1} &= \{e_1, e_3, e_7\}, & \mathsf{FinishedPrj}^{\mathcal{I}_1} &= \{\}, \\ \mathsf{worksFor}^{\mathcal{I}_1} &= \{(e_1, p_1), (e_3, p_1), (e_7, p_1), (e_7, p_2)\}. \end{aligned}$$

We consider individuals $\mathsf{p_i}$ with $\mathsf{p_i}^{\mathcal{I}} = p_i$ for projects, and analogously individuals $\mathsf{e_i}$ for employees. The following action α_1 captures the termination of project p_1, which is removed from the active projects and added to the finished ones. The employees working only for this project are removed.

$$\alpha_1 = (\mathsf{ActivePrj} \ominus \{\mathsf{p_1}\}) \cdot (\mathsf{FinishedPrj} \oplus \{\mathsf{p_1}\}) \cdot (\mathsf{Empl} \ominus \forall \mathsf{worksFor}.\{\mathsf{p_1}\})$$

The instance $S_{\alpha_1}(\mathcal{I}_1)$ that reflects the status of the database after action α_1 is as follows:

$$\begin{aligned} \mathsf{Prj}^{\mathcal{I}_1} &= \{p_1, p_2\}, & \mathsf{ActivePrj}^{\mathcal{I}_1} &= \{p_2\}, \\ \mathsf{Empl}^{\mathcal{I}_1} &= \{e_7\}, & \mathsf{FinishedPrj}^{\mathcal{I}_1} &= \{p_1\}, \\ \mathsf{worksFor}^{\mathcal{I}_1} &= \{(e_1, p_1), (e_3, p_1), (e_7, p_1), (e_7, p_2)\}. \end{aligned}$$

Despite the apparent simplicity of the action language, it can express interesting forms of update on the data. For instance, the presence of complex concepts in basic actions allows us to do updates such as 'delete all employees that work only for project p_1', as in the example above. If we did not have such complex formulas in actions, we would need to resort to more complex constructors in the action language, such as *while* loops, as considered, e.g., by Brenas et al. [2014]. However, this would quickly lead to the undecidability of all reasoning problems considered in this paper.

Note that we have not defined the semantics of actions with variables, i.e., of non-ground actions. In our approach, all variables of an action are seen as parameters whose values are given before execution by a substitution with actual individuals, i.e., by grounding.

Example 3.4. The following action α_2 with variables x, y, z transfers the employee x from project y to project z:

$$\begin{aligned}\alpha_2 = \ & (x : \mathsf{Empl} \wedge y : \mathsf{Prj} \wedge z : \mathsf{Prj} \wedge (x,y) : \mathsf{worksFor})\,?\\ & ((\mathsf{worksFor} \ominus \{(x,y)\}) \cdot (\mathsf{worksFor} \oplus \{(x,z)\}))\end{aligned}$$

Under the substitution σ with $\sigma(x) = \mathsf{e}_1$, $\sigma(y) = \mathsf{p}_1$, and $\sigma(z) = \mathsf{p}_2$, the action α_2 first checks whether e_1 is an (instance of) employee, p_1 and p_2 are projects, and e_1 works for p_1. If yes, it removes the $\mathsf{worksFor}$ link between e_1 and p_1, and creates a $\mathsf{worksFor}$ link between e_1 and p_2. If any of the checks fails, it does nothing.

Note that the execution of actions on an initial instance is allowed to modify the extensions of concept and role names, yet the domain remains fixed. In many scenarios, we would like actions to have the ability to introduce "fresh" objects, e.g., to add new nodes to a graph database. Intuitively, introduction of new objects can be modeled in our setting by separating the domain of an instance $\mathcal{I}$ into the *active domain*, and the *inactive domain*. The active domain consists of all objects that occur in extensions of concept and role names, while the inactive domain contains the remaining elements, which can be seen as a supply of fresh objects that can be introduced into the active domain by executing actions. Since in this paper we are interested only in finite sequences of actions, a sufficient supply of fresh objects can always be ensured by taking a sufficiently large yet still finite inactive domain in the initial instance. We remark also that deletion of objects can naturally be modeled in this setting by actions that move objects from the active domain to the inactive domain.

Example 3.5. Consider instance $\mathcal{I}_1$ from Example 3.3, and let ADom be a shorthand for the concept $\mathsf{Empl} \sqcup \mathsf{Prj} \sqcup \mathsf{ActivePrj} \sqcup \mathsf{FinishedPrj} \sqcup \exists \mathsf{worksFor}.\top \sqcup \exists \mathsf{worksFor}^{-}.\top$. Intuitively, ADom collects the active domain of the instance $\mathcal{I}_1$.

The following action α_3 adds a new employee x to work for project p_2.

$$\alpha_3 = (x : \neg\mathsf{ADom} \wedge \mathsf{p}_2 : \mathsf{Prj})?((\mathsf{Empl} \oplus \{x\}) \cdot (\mathsf{worksFor} \oplus \{(x, \mathsf{p}_2)\}))$$

Under the substitution σ with $\sigma(x) = \mathsf{e}_8$, the action α_3 first checks whether e_8 is a fresh object (i.e., is not in the active domain) and p_2 is a project. If yes, it adds e_8 to the employees and creates a $\mathsf{worksFor}$ link between e_8 and p_2. If any of the checks fails, it does nothing.

4. CAPTURING ACTION EFFECTS

In this section we present our core technical tool: a transformation $\mathsf{TR}_\alpha(\mathcal{K})$ that rewrites $\mathcal{K}$ by incorporating the possible effects of an action α. Intuitively, the models of $\mathsf{TR}_\alpha(\mathcal{K})$ are exactly the instances $\mathcal{I}$ such that applying α on $\mathcal{I}$ leads to a model of $\mathcal{K}$. That is, $\mathsf{TR}_\alpha(\mathcal{K})$ can be seen as the *weakest pre-condition* of $\mathcal{K}$ for α. Having $\mathsf{TR}_\alpha(\mathcal{K})$ as an $\mathcal{ALCHOIQ}br$ KB allows us to effectively reduce reasoning about changes in any database that satisfies a given $\mathcal{K}$, to reasoning about a single KB. We will use this transformation to solve a wide range of data management problems by reducing them to standard DL reasoning services, such as finite (un)satisfiability. This transformation can be seen as a form of *regression* [Levesque et al. 1997], which incorporates the effects of a sequence of actions 'backwards', from the last one to the first one.

Definition 4.1. Given a KB $\mathcal{K}$, we use $\mathcal{K}_{E\leftarrow E'}$ to denote the KB that is obtained from $\mathcal{K}$ by replacing every name E by the (possibly more complex) expression E'. Given a KB $\mathcal{K}$ and an action α, we define $\mathsf{TR}_\alpha(\mathcal{K})$ as follows:

$$\begin{aligned}
\mathsf{TR}_\varepsilon(\mathcal{K}) &= \mathcal{K}\\
\mathsf{TR}_{(A\oplus C)\cdot\alpha}(\mathcal{K}) &= (\mathsf{TR}_\alpha(\mathcal{K}))_{A\leftarrow A\sqcup C}\\
\mathsf{TR}_{(A\ominus C)\cdot\alpha}(\mathcal{K}) &= (\mathsf{TR}_\alpha(\mathcal{K}))_{A\leftarrow A\sqcap\neg C}\\
\mathsf{TR}_{(p\oplus r)\cdot\alpha}(\mathcal{K}) &= (\mathsf{TR}_\alpha(\mathcal{K}))_{p\leftarrow p\cup r}\\
\mathsf{TR}_{(p\ominus r)\cdot\alpha}(\mathcal{K}) &= (\mathsf{TR}_\alpha(\mathcal{K}))_{p\leftarrow p\setminus r}\\
\mathsf{TR}_{(\mathcal{K}_1\,?\,\alpha_1[\![\alpha_2]\!])\cdot\alpha}(\mathcal{K}) &= (\dot\neg\mathcal{K}_1\vee\mathsf{TR}_{\alpha_1\cdot\alpha}(\mathcal{K}))\wedge(\mathcal{K}_1\vee\mathsf{TR}_{\alpha_2\cdot\alpha}(\mathcal{K})).
\end{aligned} \quad \triangleleft$$

Note that, in the presence of conditional axioms, the size of $\mathsf{TR}_\alpha(\mathcal{K})$ might be exponential in the size of α. We now show that this transformation correctly captures the effects of complex actions.

THEOREM 4.2. *Consider a ground action α and a KB $\mathcal{K}$. For every instance $\mathcal{I}$, we have that $S_\alpha(\mathcal{I})\models\mathcal{K}$ iff $\mathcal{I}\models\mathsf{TR}_\alpha(\mathcal{K})$.*

PROOF. We prove the claim by induction on $\ell(\alpha)$, defined as follows: $\ell(\varepsilon)=0$, $\ell(\beta\cdot\alpha)=1+\ell(\alpha)$, and $\ell(\mathcal{K}\,?\,\alpha_1[\![\alpha_2]\!]\cdot\alpha_3)=1+\ell(\alpha_1)+\ell(\alpha_2)+\ell(\alpha_3)$. In the base case where $\ell(\alpha)=0$ and $\alpha=\varepsilon$, we have $S_\alpha(\mathcal{I})=\mathcal{I}$ and $\mathsf{TR}_\alpha(\mathcal{K})=\mathcal{K}$ by definition, and thus the claim holds.

Assume $\alpha=(A\oplus C)\cdot\alpha'$. Let $\mathcal{I}'=\mathcal{I}\oplus_A C^{\mathcal{I}}$, that is, $\mathcal{I}'$ coincides with $\mathcal{I}$ except that $A^{\mathcal{I}'}=A^{\mathcal{I}}\cup C^{\mathcal{I}}$. For every KB $\mathcal{K}'$, $\mathcal{I}'\models\mathcal{K}'$ iff $\mathcal{I}\models\mathcal{K}'_{A\leftarrow A\sqcup C}$ (This can be proved by a straightforward induction on the structure of the expressions in $\mathcal{K}'$). In particular, $\mathcal{I}'\models\mathsf{TR}_{\alpha'}(\mathcal{K})$ iff $\mathcal{I}\models(\mathsf{TR}_{\alpha'}(\mathcal{K}))_{A\leftarrow A\sqcup C}$. Since $(\mathsf{TR}_{\alpha'}(\mathcal{K}))_{A\leftarrow A\sqcup C}=\mathsf{TR}_\alpha(\mathcal{K})$, we get $\mathcal{I}'\models\mathsf{TR}_{\alpha'}(\mathcal{K})$ iff $\mathcal{I}\models\mathsf{TR}_\alpha(\mathcal{K})$. By the induction hypothesis, $\mathcal{I}'\models\mathsf{TR}_{\alpha'}(\mathcal{K})$ iff $S_{\alpha'}(\mathcal{I}')\models\mathcal{K}$, thus $\mathcal{I}\models\mathsf{TR}_\alpha(\mathcal{K})$ iff $S_{\alpha'}(\mathcal{I}')\models\mathcal{K}$. Since $S_{\alpha'}(\mathcal{I}')=S_{\alpha'}(S_{(A\oplus C)}(\mathcal{I}))=S_\alpha(\mathcal{I})$ by definition, we obtain $\mathcal{I}\models\mathsf{TR}_\alpha(\mathcal{K})$ iff $S_\alpha(\mathcal{I})\models\mathcal{K}$ as desired.

For the cases $\alpha=(A\ominus C)\cdot\alpha'$, $\alpha=(p\oplus r)\cdot\alpha'$, and $\alpha=(p\ominus r)\cdot\alpha'$, the argument is analogous.

Finally, we consider $\alpha=(\mathcal{K}_1\,?\,\alpha_1[\![\alpha_2]\!])\cdot\alpha'$, and assume an arbitrary $\mathcal{I}$. We consider the case where $\mathcal{I}\models\mathcal{K}_1$; the case where $\mathcal{I}\not\models\mathcal{K}_1$ is analogous. By definition $S_\alpha(\mathcal{I})=S_{\alpha_1\cdot\alpha'}(\mathcal{I})$. By the induction hypothesis we know that $S_{\alpha_1\cdot\alpha'}(\mathcal{I})\models\mathcal{K}$ iff $\mathcal{I}\models\mathsf{TR}_{\alpha_1\cdot\alpha'}(\mathcal{K})$, so $S_\alpha(\mathcal{I})\models\mathcal{K}$ iff $\mathcal{I}\models\mathsf{TR}_{\alpha_1\cdot\alpha'}(\mathcal{K})$. Since $\mathcal{I}\models\mathcal{K}_1$ and $\mathsf{TR}_{(\mathcal{K}_1\,?\,\alpha_1[\![\alpha_2]\!])\cdot\alpha'}(\mathcal{K})=(\dot\neg\mathcal{K}_1\vee\mathsf{TR}_{\alpha_1\cdot\alpha'}(\mathcal{K}))\wedge(\mathcal{K}_1\vee\mathsf{TR}_{\alpha_2\cdot\alpha'}(\mathcal{K}))$, it follows that $S_\alpha(\mathcal{I})\models\mathcal{K}$ iff $\mathcal{I}\models\mathsf{TR}_{(\mathcal{K}_1\,?\,\alpha_1[\![\alpha_2]\!])\cdot\alpha'}(\mathcal{K})$. □

This theorem will be important for solving the reasoning problems we study below.

Example 4.3. The KB $\mathcal{K}_1$ below expresses the following constraints on the project database of our running example: all projects are active or finished, the domain of worksFor are the employees, and its range the projects.

$$\mathcal{K}_1=(\mathsf{Prj}\sqsubseteq\mathsf{ActivePrj}\sqcup\mathsf{FinishedPrj})\wedge(\exists\mathsf{worksFor}.\top\sqsubseteq\mathsf{Empl})\wedge(\exists\mathsf{worksFor}^-.\top\sqsubseteq\mathsf{Prj})$$

By applying our transformation to $\mathcal{K}_1$ and the action α_1 from Example 3.3, i.e.,

$$\alpha_1=(\mathsf{ActivePrj}\ominus\{\mathsf{p_1}\})\cdot(\mathsf{FinishedPrj}\oplus\{\mathsf{p_1}\})\cdot(\mathsf{Empl}\ominus\forall\mathsf{worksFor}.\{\mathsf{p_1}\})$$

we obtain the following KB $\mathsf{TR}_{\alpha_1}(\mathcal{K}_1)$:

$$\begin{gathered}
(\mathsf{Prj}\sqsubseteq(\mathsf{ActivePrj}\sqcap\neg\{\mathsf{p_1}\})\sqcup(\mathsf{FinishedPrj}\sqcup\{\mathsf{p_1}\}))\wedge\\
(\exists\mathsf{worksFor}.\top\sqsubseteq\mathsf{Empl}\sqcap\exists\mathsf{worksFor}.\neg\{\mathsf{p_1}\})\wedge\\
(\exists\mathsf{worksFor}^-.\top\sqsubseteq\mathsf{Prj})
\end{gathered}$$

5. STATIC VERIFICATION

In this section, we consider the scenario where DL KBs are used to impose integrity constraints on GSD. One of the most basic reasoning problems for action analysis in this setting is the *static verification problem*, which consists in checking whether the execution of an action always preserves the satisfaction of integrity constraints given by a KB.

Definition 5.1 (*Static verification*). Let $\mathcal{K}$ be a KB. We call an action α *$\mathcal{K}$-preserving* if for every ground instance α' of α and every finite instance $\mathcal{I}$, we have that $\mathcal{I} \models \mathcal{K}$ implies $S_{\alpha'}(\mathcal{I}) \models \mathcal{K}$. The *static verification problem* is defined as follows:

(SV) Given an action α and a KB $\mathcal{K}$, is α $\mathcal{K}$-preserving? $\lhd$

Using the transformation $\mathsf{TR}_\alpha(\mathcal{K})$ above, we can reduce static verification to finite (un)satisfiability of $\mathcal{ALCHOIQ}br$ KBs: An action α is not $\mathcal{K}$-preserving iff some finite model of $\mathcal{K}$ does not satisfy $\mathsf{TR}_{\alpha^*}(\mathcal{K})$, where α^* is a 'canonical' grounding of α obtained by replacing each variable with a fresh individual. Formally:

THEOREM 5.2. *Assume a (complex) action α and a KB $\mathcal{K}$. Then the following are equivalent:*
(i) The action α is not $\mathcal{K}$-preserving.
(ii) $\mathcal{K} \wedge \dot\neg\mathsf{TR}_{\alpha^*}(\mathcal{K})$ is finitely satisfiable, where α^* is obtained from α by replacing each variable with a fresh individual name not occurring in α and $\mathcal{K}$.

PROOF. *(i)* implies *(ii)*. Assume there exist a ground instance α' of α and a finite instance $\mathcal{I}$ such that $\mathcal{I} \models \mathcal{K}$ and $S_{\alpha'}(\mathcal{I}) \not\models \mathcal{K}$. Then by Theorem 4.2, $\mathcal{I} \not\models \mathsf{TR}_{\alpha'}(\mathcal{K})$. Thus $\mathcal{I} \models \dot\neg\mathsf{TR}_{\alpha'}(\mathcal{K})$. Suppose $o_1 \to x_1, \ldots, o_n \to x_n$ is the substitution that transforms α into α'. Suppose also $o'_1 \to x_1, \ldots, o'_n \to x_n$ is the substitution that transforms α into α^*. Take the instance $\mathcal{I}^*$ that coincides with $\mathcal{I}$ except for ${o'_i}^{\mathcal{I}^*} = o_i^{\mathcal{I}}$. Then $\mathcal{I}^* \models \mathcal{K} \wedge \dot\neg\mathsf{TR}_{\alpha^*}(\mathcal{K})$.
(ii) implies *(i)*. Assume $\mathcal{K} \wedge \dot\neg\mathsf{TR}_{\alpha^*}(\mathcal{K})$ is finitely satisfiable, i.e., there is an instance $\mathcal{I}$ such that $\mathcal{I} \models \mathcal{K}$ and $\mathcal{I} \not\models \mathsf{TR}_{\alpha^*}(\mathcal{K})$. Then by Theorem 4.2, $S_{\alpha^*}(\mathcal{I}) \not\models \mathcal{K}$. $\square$

Example 5.3. The action α_1 from Example 3.3 is not $\mathcal{K}_1$-preserving: $\mathcal{I}_1 \models \mathcal{K}_1$, but $S_{\alpha_1}(\mathcal{I}_1) \not\models \mathcal{K}_1$ since the concept inclusion $\exists\mathsf{worksFor}.\top \sqsubseteq \mathsf{Empl}$ is violated. This is reflected in the fact that $\mathcal{I}_1 \not\models \mathsf{TR}_{\alpha_1}(\mathcal{K}_1)$, as can be readily checked. Indeed, $(\exists\mathsf{worksFor}.\top)^{\mathcal{I}_1} = \{e_1, e_3, e_7\}$ while $(\mathsf{Empl} \sqcap \exists\mathsf{worksFor}.\{\neg\mathsf{p}_1\})^{\mathcal{I}_1} = \{e_7\}$, violating the second conjunct of $\mathsf{TR}_{\alpha_1}(\mathcal{K}_1)$. Intuitively, values removed from Empl should also be removed from worksFor, as in the following $\mathcal{K}_1$-preserving action:

$$\alpha'_1 = (\mathsf{ActivePrj} \ominus \{\mathsf{p}_1\}) \cdot (\mathsf{FinishedPrj} \oplus \{\mathsf{p}_1\}) \cdot \\ (\mathsf{Empl} \ominus \forall\mathsf{worksFor}.\{\mathsf{p}_1\}) \cdot (\mathsf{worksFor} \ominus \mathsf{worksFor}|_{\{\mathsf{p}_1\}})$$

By applying α'_1 to $\mathcal{K}_1$, we obtain the following transformed KB $\mathsf{TR}_{\alpha'_1}(\mathcal{K}_1)$:

$$(\mathsf{Prj} \sqsubseteq (\mathsf{ActivePrj} \sqcap \neg\{\mathsf{p}_1\}) \sqcup (\mathsf{FinishedPrj} \sqcup \{\mathsf{p}_1\})) \wedge \\ (\exists(\mathsf{worksFor} \setminus (\mathsf{worksFor}|_{\{\mathsf{p}_1\}})).\top \sqsubseteq \mathsf{Empl} \sqcap \exists(\mathsf{worksFor} \setminus (\mathsf{worksFor}|_{\{\mathsf{p}_1\}})).\neg\{\mathsf{p}_1\}) \wedge \\ (\exists(\mathsf{worksFor} \setminus (\mathsf{worksFor}|_{\{\mathsf{p}_1\}}))^-.\top \sqsubseteq \mathsf{Prj})$$

The above theorem provides an algorithm for static verification, which we can also use to obtain tight bounds on the computational complexity of the problem. Indeed, even though $\mathcal{K} \wedge \dot\neg\mathsf{TR}_{\alpha^*}(\mathcal{K})$ may be of size exponential in α, we can avoid to generate it all at once. More precisely, we use a non-deterministic polynomial time many-one reduction that builds only $\mathcal{K} \wedge \dot\neg\mathsf{TR}^c_{\alpha^*}(\mathcal{K})$ for a fragment $\dot\neg\mathsf{TR}^c_{\alpha^*}(\mathcal{K})$ of $\dot\neg\mathsf{TR}_{\alpha^*}(\mathcal{K})$ that corresponds to one fixed way of choosing one of α_1 or α_2 for each conditional action $\mathcal{K}' \,?\, \alpha_1 [\![\alpha_2]\!]$ in α (intuitively, we can view $\dot\neg\mathsf{TR}^c_{\alpha^*}(\mathcal{K})$ as one conjunct of the DNF of

$\dot{\neg}\mathsf{TR}_{\alpha^*}(\mathcal{K})$, where axioms and assertions are treated as propositions). Such a $\dot{\neg}\mathsf{TR}^c_{\alpha^*}(\mathcal{K})$ has polynomial size, and it can be built non-deterministically in polynomial time. We can then show that $\mathcal{K} \wedge \dot{\neg}\mathsf{TR}_{\alpha^*}(\mathcal{K})$ is finitely satisfiable iff there is some choice $\mathsf{TR}^c_{\alpha^*}(\mathcal{K})$ such that $\mathcal{K} \wedge \dot{\neg}\mathsf{TR}^c_{\alpha^*}(\mathcal{K})$ is finitely satisfiable. By Theorem 2.6, the latter test can be done in non-deterministic exponential time, hence from Theorem 5.2 we obtain a coNEXPTIME upper bound, which turns out to be tight. When we rule out number restrictions from the input KB, the complexity drops to EXPTIME. The details are provided in the proof of the following theorem.

THEOREM 5.4. *The problem (SV) is* coNEXPTIME*-complete when the input KB is expressed in $\mathcal{ALCHOIQ}br$, and* EXPTIME*-complete when the input KB is expressed in $\mathcal{ALCHOI}br$.*

PROOF. For the hardness, we note that for both $\mathcal{ALCHOIQ}br$ and $\mathcal{ALCHOI}br$, finite unsatisfiability of KBs can be reduced in polynomial time to static verification in the presence of KBs in the same logic. Indeed, a KB $\mathcal{K}$ is finitely satisfiable iff $(A' \oplus \{o\})$ is not $(\mathcal{K} \wedge (A \sqsubseteq \neg A') \wedge (o : A))$-preserving, where A and A' are fresh concept names and o is a fresh individual. Hence hardness for $\mathcal{ALCHOIQ}br$ follows from Theorem 2.6, and for $\mathcal{ALCHOI}br$ from Proposition 2.7.

Obtaining a matching upper bound is slightly more involved. Theorem 5.2 establishes that verifying whether an action α is *not* $\mathcal{K}$-preserving, where $\mathcal{K}$ is an $\mathcal{ALCHOIQ}br$ KBs, reduces to finite satisfiability of a KB $\mathcal{K} \wedge \dot{\neg}\mathsf{TR}_{\alpha^*}(\mathcal{K})$ in $\mathcal{ALCHOIQ}br$, but unfortunately, this reduction is exponential in general. Hence we use an alternative reduction that allows us to build a *set of KBs* of exponential size, but where each of them is of polynomial size, and can be obtained in polynomial time. Crucially, $\mathcal{K} \wedge \dot{\neg}\mathsf{TR}_{\alpha^*}(\mathcal{K})$ is finitely satisfiable iff this set contains some $\mathcal{K}'$ such that $\mathcal{K} \wedge \mathcal{K}'$ is finitely satisfiable. Hence, the complement of the (SV) problem reduces to deciding whether there is some $\mathcal{K}'$ in the set for which $\mathcal{K} \wedge \mathcal{K}'$ is satisfiable. The desired upper bounds can now be easily inferred. In the case of $\mathcal{ALCHOIQ}br$, we non-deterministically build the right $\mathcal{K}'$ in polynomial time, and then test for finite satisfiability of $\mathcal{K} \wedge \mathcal{K}'$. Since the latter is an $\mathcal{ALCHOIQ}br$ KB whose size is bounded polynomially by the input $\mathcal{K}$ and α, deciding its finite satisfiability is in NEXPTIME, see Theorem 2.6. For $\mathcal{ALCHOI}br$, we can build in exponential time all the KBs $\mathcal{K}'$ in the set, which are exponentially many, but each of them is polynomial in the input $\mathcal{K}$ and α. Then we iterate over the full set and, for each $\mathcal{K}'$, we decide in EXPTIME whether $\mathcal{K} \wedge \mathcal{K}'$ is finitely satisfiable (see Proposition 2.7). The upper bound then follows from the fact that doing exponentially many EXPTIME tests is feasible in EXPTIME.

Now, to obtain the described set of KBs, it is convenient to first define a minor variation $\overline{\mathsf{TR}}_\alpha(\mathcal{K})$ of the transformation $\mathsf{TR}_\alpha(\mathcal{K})$, which generates an already negated KB.

$$\begin{aligned}
\overline{\mathsf{TR}}_\varepsilon(\mathcal{K}) &= \dot{\neg}\mathcal{K} \\
\overline{\mathsf{TR}}_{(A\oplus C)\cdot\alpha}(\mathcal{K}) &= (\overline{\mathsf{TR}}_\alpha(\mathcal{K}))_{A \leftarrow A \sqcup C} \\
\overline{\mathsf{TR}}_{(A\ominus C)\cdot\alpha}(\mathcal{K}) &= (\overline{\mathsf{TR}}_\alpha(\mathcal{K}))_{A \leftarrow A \sqcap \neg C} \\
\overline{\mathsf{TR}}_{(p\oplus r)\cdot\alpha}(\mathcal{K}) &= (\overline{\mathsf{TR}}_\alpha(\mathcal{K}))_{p \leftarrow p \cup r} \\
\overline{\mathsf{TR}}_{(p\ominus r)\cdot\alpha}(\mathcal{K}) &= (\overline{\mathsf{TR}}_\alpha(\mathcal{K}))_{p \leftarrow p \setminus r} \\
\overline{\mathsf{TR}}_{(\mathcal{K}_1\,?\,\alpha_1[\![\alpha_2]\!])\cdot\alpha}(\mathcal{K}) &= \big(\mathcal{K}_1 \wedge \overline{\mathsf{TR}}_{\alpha_1\cdot\alpha}(\mathcal{K})\big) \vee \big(\dot{\neg}\mathcal{K}_1 \wedge \overline{\mathsf{TR}}_{\alpha_2\cdot\alpha}(\mathcal{K})\big)
\end{aligned}$$

It can be shown by a straightforward induction on $\ell(\alpha)$ (as defined in the proof of Theorem 4.2) that $\overline{\mathsf{TR}}_\alpha(\mathcal{K})$ is logically equivalent to $\dot{\neg}\mathsf{TR}_\alpha(\mathcal{K})$ for every $\mathcal{K}$ and every α. Hence, by Theorem 4.2, $\mathcal{K} \wedge \overline{\mathsf{TR}}_{\alpha^*}(\mathcal{K})$ is finitely satisfiable iff $\mathcal{K} \wedge \dot{\neg}\mathsf{TR}_{\alpha^*}(\mathcal{K})$ is finitely satisfiable iff α is not $\mathcal{K}$-preserving.

Now, we define a translation analogous to $\overline{\mathsf{TR}}_\alpha(\mathcal{K})$, but in the last case, for the conditional axioms, we generate two separate KBs $\mathcal{K}_1 \wedge \overline{\mathsf{TR}}_{\alpha_1\cdot\alpha}(\mathcal{K})$ and $\dot\neg\mathcal{K}_1 \wedge \overline{\mathsf{TR}}_{\alpha_2\cdot\alpha}(\mathcal{K})$, rather than considering the disjunction of both. We denote by $\overline{\mathbf{TR}}_\alpha(\mathcal{K})$ the set of all the KBs obtained this way, that is:

$$\begin{aligned}
\overline{\mathbf{TR}}_\varepsilon(\mathcal{K}) &= \{\dot\neg\mathcal{K}\}\\
\overline{\mathbf{TR}}_{(A\oplus C)\cdot\alpha}(\mathcal{K}) &= \{\mathcal{K}'_{A\leftarrow A\sqcup C} \mid \mathcal{K}' \in \overline{\mathbf{TR}}_\alpha(\mathcal{K})\}\\
\overline{\mathbf{TR}}_{(A\ominus C)\cdot\alpha}(\mathcal{K}) &= \{\mathcal{K}'_{A\leftarrow A\sqcap\neg C} \mid \mathcal{K}' \in \overline{\mathbf{TR}}_\alpha(\mathcal{K})\}\\
\overline{\mathbf{TR}}_{(p\oplus r)\cdot\alpha}(\mathcal{K}) &= \{\mathcal{K}'_{p\leftarrow p\cup r} \mid \mathcal{K}' \in \overline{\mathbf{TR}}_\alpha(\mathcal{K})\}\\
\overline{\mathbf{TR}}_{(p\ominus r)\cdot\alpha}(\mathcal{K}) &= \{\mathcal{K}'_{p\leftarrow p\setminus r} \mid \mathcal{K}' \in \overline{\mathbf{TR}}_\alpha(\mathcal{K})\}\\
\overline{\mathbf{TR}}_{(\mathcal{K}_1\,?\,\alpha_1[\![\alpha_2]\!])\cdot\alpha}(\mathcal{K}) &= \{\mathcal{K}_1 \wedge \mathcal{K}' \mid \mathcal{K}' \in \overline{\mathbf{TR}}_{\alpha_1\cdot\alpha}(\mathcal{K})\} \cup \{\dot\neg\mathcal{K}_1 \wedge \mathcal{K}' \mid \mathcal{K}' \in \overline{\mathbf{TR}}_{\alpha_2\cdot\alpha}(\mathcal{K})\}
\end{aligned}$$

It is easy to see that $|\overline{\mathbf{TR}}_\alpha(\mathcal{K})|$ may be exponential in α and $\mathcal{K}$, but each $\mathcal{K}' \in \overline{\mathbf{TR}}_\alpha(\mathcal{K})$ is of polynomial size and can be built in polynomial time. It is only left to show that $\mathcal{K} \wedge \overline{\mathsf{TR}}_\alpha(\mathcal{K})$ is finitely satisfiable iff there is some $\mathcal{K}' \in \overline{\mathbf{TR}}_\alpha(\mathcal{K})$ such that $\mathcal{K} \wedge \mathcal{K}'$ is finitely satisfiable. This is a consequence of the fact that, for every instance $\mathcal{I}$, we have that $\mathcal{I} \models \overline{\mathsf{TR}}_\alpha(\mathcal{K})$ iff there is some $\mathcal{K}' \in \overline{\mathbf{TR}}_\alpha(\mathcal{K})$ such that $\mathcal{I} \models \mathcal{K}'$.

We show this by induction on $\ell(\alpha)$. The base case is straightforward: if $\alpha = \varepsilon$, then $\overline{\mathbf{TR}}_\alpha(\mathcal{K}) = \{\overline{\mathsf{TR}}_\alpha(\mathcal{K})\}$. For the inductive step, we first consider $\alpha' = (A \oplus C)\cdot\alpha$. First we assume that $\mathcal{I} \models \overline{\mathsf{TR}}_{\alpha'}(\mathcal{K})$. That is, $\mathcal{I} \models (\overline{\mathsf{TR}}_\alpha(\mathcal{K}))_{A\leftarrow A\sqcup C}$, hence $S_{(A\oplus C)}(\mathcal{I}) \models \overline{\mathsf{TR}}_\alpha(\mathcal{K})$. We can apply the induction hypothesis to $S_{(A\oplus C)}(\mathcal{I})$ and $\overline{\mathsf{TR}}_\alpha(\mathcal{K})$ to infer that there exists $\mathcal{K}' \in \overline{\mathbf{TR}}_\alpha(\mathcal{K})$ such that $S_{(A\oplus C)}(\mathcal{I}) \models \mathcal{K}'$, hence $\mathcal{I} \models \mathcal{K}'_{A\leftarrow A\sqcup C}$. Since $\overline{\mathbf{TR}}_{\alpha'}(\mathcal{K}) = \{\mathcal{K}'_{A\leftarrow A\sqcup C} \mid \mathcal{K}' \in \overline{\mathbf{TR}}_\alpha(\mathcal{K})\}$, we have that $\mathcal{K}'' = \mathcal{K}'_{A\leftarrow A\sqcup C}$ is such that $\mathcal{K}'' \in \overline{\mathbf{TR}}_{\alpha'}(\mathcal{K})$ and $\mathcal{I} \models \mathcal{K}''$, as desired. For the converse, if $\mathcal{I} \models \mathcal{K}''$ for some $\mathcal{K}'' \in \overline{\mathbf{TR}}_{\alpha'}(\mathcal{K})$, by definition we have that there is some $\mathcal{K}' \in \overline{\mathbf{TR}}_\alpha(\mathcal{K})$ such that $\mathcal{K}'' = \mathcal{K}'_{A\leftarrow A\sqcup C}$, and since $\mathcal{I} \models \mathcal{K}'_{A\leftarrow A\sqcup C}$ we have that $S_{(A\oplus C)}(\mathcal{I}) \models \mathcal{K}'$. Using the induction hypothesis on $S_{(A\oplus C)}(\mathcal{I})$ and $\mathcal{K}'$, we get $S_{(A\oplus C)}(\mathcal{I}) \models \overline{\mathsf{TR}}_\alpha(\mathcal{K})$, hence $\mathcal{I} \models (\overline{\mathsf{TR}}_\alpha(\mathcal{K}))_{A\leftarrow A\sqcup C}$, that is, $\mathcal{I} \models \overline{\mathsf{TR}}_{\alpha'}(\mathcal{K})$, as desired. The cases of $\alpha' = (A\ominus C)\cdot\alpha$, $\alpha' = (p\oplus r)\cdot\alpha$, and $\alpha' = (p\ominus r)\cdot\alpha$ are analogous.

Finally, consider $\alpha' = (\mathcal{K}_1\,?\,\alpha_1[\![\alpha_2]\!])\cdot\alpha$. We first show that if $\mathcal{I} \models \overline{\mathsf{TR}}_{\alpha'}(\mathcal{K})$, then there is some $\mathcal{K}' \in \overline{\mathbf{TR}}_{\alpha'}(\mathcal{K})$ such that $\mathcal{I} \models \mathcal{K}'$. By definition, $\overline{\mathsf{TR}}_{\alpha'}(\mathcal{K}) = \big(\mathcal{K}_1 \wedge \overline{\mathsf{TR}}_{\alpha_1\cdot\alpha}(\mathcal{K})\big) \vee \big(\dot\neg\mathcal{K}_1 \wedge \overline{\mathsf{TR}}_{\alpha_2\cdot\alpha}(\mathcal{K})\big)$. So, if $\mathcal{I} \models \overline{\mathsf{TR}}_{\alpha'}(\mathcal{K})$, then one of $\mathcal{I} \models \mathcal{K}_1 \wedge \overline{\mathsf{TR}}_{\alpha_1\cdot\alpha}(\mathcal{K})$ or $\mathcal{I} \models \dot\neg\mathcal{K}_1 \wedge \overline{\mathsf{TR}}_{\alpha_2\cdot\alpha}(\mathcal{K})$ holds. In the former case, we can use the induction hypothesis to conclude that there exists some $\mathcal{K}' \in \overline{\mathbf{TR}}_{\alpha_1\cdot\alpha}(\mathcal{K})$ such that $\mathcal{I} \models \mathcal{K}_1 \wedge \mathcal{K}'$. Since $\mathcal{K}_1 \wedge \mathcal{K}' \in \overline{\mathbf{TR}}_{\alpha'}(\mathcal{K})$ by definition, the claim follows. The latter case is analogous. For the converse, we assume that there exists some $\mathcal{K}' \in \overline{\mathbf{TR}}_{\alpha'}(\mathcal{K})$ such that $\mathcal{I} \models \mathcal{K}'$. By definition, this $\mathcal{K}'$ must be of the form $\mathcal{K}_1 \wedge \mathcal{K}''$ with $\mathcal{K}'' \in \overline{\mathbf{TR}}_{\alpha_1\cdot\alpha}(\mathcal{K})$, or of the form $\dot\neg\mathcal{K}_1 \wedge \mathcal{K}''$ with $\mathcal{K}'' \in \overline{\mathbf{TR}}_{\alpha_2\cdot\alpha}(\mathcal{K})$. In the former case, it follows from the induction hypothesis that $\mathcal{I} \models \mathcal{K}_1 \wedge \overline{\mathsf{TR}}_{\alpha_1\cdot\alpha}(\mathcal{K})$, and hence $\mathcal{I} \models \big(\mathcal{K}_1 \wedge \overline{\mathsf{TR}}_{\alpha_1\cdot\alpha}(\mathcal{K})\big) \vee \big(\dot\neg\mathcal{K}_1 \wedge \overline{\mathsf{TR}}_{\alpha_2\cdot\alpha}(\mathcal{K})\big)$ and the claim follows. The second case, where $\mathcal{K}'$ is of the form $\dot\neg\mathcal{K}_1 \wedge \mathcal{K}''$, is analogous to the first one. □

We note that in our definition of the (SV) problem, in addition to the action to be verified, one has as input only one KB $\mathcal{K}$ expressing constraints. We can also consider other interesting variations of the problem where, for example, we have a pair of KBs $\mathcal{K}_{pre}$ and $\mathcal{K}_{post}$ instead of (or in addition to) $\mathcal{K}$ and we want to decide whether executing the action on any model of $\mathcal{K}_{pre}$ (and $\mathcal{K}$) leads to a model of $\mathcal{K}_{post}$ (and $\mathcal{K}$). The reasoning techniques and upper bounds presented above also apply to these generalized settings.

5.1. Lowering the Complexity

The goal of this section is to identify a setting for which the computational complexity of static verification is lower. The natural way to achieve this is to consider as constraint language a DL with better computational properties, such as the logics of the *DL-Lite* family [Calvanese et al. 2007].

Unfortunately, we cannot achieve tractability, since static verification is coNP-hard even in a very restricted setting, as shown next.

THEOREM 5.5. *The static verification problem is* coNP*-hard already for KBs of the form* $(A_1 \sqsubseteq \neg A'_1) \wedge \cdots \wedge (A_n \sqsubseteq \neg A'_n)$*, where each* A_i*,* A'_i *is a concept name, and for ground sequences of basic actions of the forms* $(A \oplus C)$ *and* $(A \ominus C)$*, where* C *is a concept name or a nominal.*

PROOF. We employ the 3-Coloring problem for graphs. Consider a graph $G = (V, E)$ with $V = \{1, \ldots, n\}$. We construct in polynomial time a KB $\mathcal{K}$ and an action α such that G is 3-colorable iff α is not $\mathcal{K}$-preserving. For every $v \in V$, we use 3 concept names A_v^0, A_v^1, A_v^2 for the 3 possible colors of the vertex v. In addition, we employ a concept name D. Let $\mathcal{K}$ be the following KB:

$$\mathcal{K} \;=\; (D \sqsubseteq \neg D) \;\wedge \bigwedge_{(v,v')\in E \;\wedge\; 0\leq c\leq 2} (A_v^c \sqsubseteq \neg A_{v'}^c).$$

It remains to define the action α. For this we additionally use a nominal $\{o\}$ and fresh concept names $B_1, \ldots, B_n$. We let $\alpha := \alpha_1 \alpha_2^1 \cdots \alpha_2^n \alpha_3$, where

(i) $\alpha_1 = (D \oplus \{o\}) \cdot (B_1 \oplus \{o\}) \cdots (B_n \oplus \{o\})$,
(ii) $\alpha_2^i = (B_i \ominus A_i^0) \cdot (B_i \ominus A_i^1) \cdot (B_i \ominus A_i^2)$, for all $i \in \{1, \ldots, n\}$, and
(iii) $\alpha_3 = (D \ominus B_1) \cdots (D \ominus B_n)$.

Assume that $\mathcal{I}$ is a model of $\mathcal{K}$ such that $S_\alpha(\mathcal{I}) \not\models \mathcal{K}$. We argue that then G is 3-colorable. Indeed, since α does not modify the extensions of concepts A_v^c, $S_\alpha(\mathcal{I}) \not\models \mathcal{K}$ may only hold if $o^{\mathcal{I}}$ is in the extension of D after applying α on $\mathcal{I}$. Due to α_3, this requires $o^{\mathcal{I}}$ not to be in the extensions of any of $B_1, \ldots, B_n$ after applying $\alpha_1\alpha_2^1 \cdots \alpha_2^n$ on $\mathcal{I}$. Due to α_1 and the various α_2^i, it must be the case that $o^{\mathcal{I}}$ is in the extension in $\mathcal{I}$ of one of A_i^0, A_i^1, or A_i^2, for each $i \in \{1, \ldots, n\}$. For every $v \in V$, let $col(v) \in \{0, 1, 2\}$ be a value such that $o^{\mathcal{I}}$ is in the extension of $A_i^{col(v)}$ in $\mathcal{I}$. Since $\mathcal{I}$ satisfies the disjointness axioms in $\mathcal{K}$, the function col is a proper 3-coloring of G.

Suppose that G is 3-colorable and a proper coloring of G is given by a function $col : V \to \{0, 1, 2\}$. Take any instance $\mathcal{I}$ with $\Delta^{\mathcal{I}} = \{e\}$ and such that *(i)* $\{o\}^{\mathcal{I}} = e$, *(ii)* $D^{\mathcal{I}} = \emptyset$, and *(iii)* $e \in (A_v^c)^{\mathcal{I}}$ iff $col(v) = c$. Since col is a proper coloring of G, $\mathcal{I}$ is a model of $\mathcal{K}$. As easily seen, $S_\alpha(\mathcal{I}) \not\models \mathcal{K}$. □

We next present a rich variant of ***DL-Lite***$_{\mathcal{R}}$, which we call ***DL-Lite***$_{\mathcal{R}}^{+}$, for which the static verification problem is in coNP. It supports (restricted) Boolean combinations of inclusions and assertions, and allows for complex concepts and roles in assertions. As shown below, this allows us to express the effects of actions inside ***DL-Lite***$_{\mathcal{R}}^{+}$ KBs.

Definition 5.6. The logic ***DL-Lite***$_{\mathcal{R}}^{+}$ is defined as follows:

– Concept inclusions have the form $C_1 \sqsubseteq C_2$ or $C_1 \sqsubseteq \neg C_2$, with $\{C_1, C_2\} \subseteq \mathsf{N_C} \cup \{\exists p.\top \mid p \in \mathsf{N_R}\} \cup \{\exists p^-.\top \mid p \in \mathsf{N_R}\}$.
– Role inclusions have the form $r_1 \sqsubseteq r_2$ with $\{r_1, r_2\} \subseteq \mathsf{N_R} \cup \{p^- \mid p \in \mathsf{N_R}\}$.

– Role assertions are defined as for $\mathcal{ALCHOIQ}br$, but in concept assertions $o : C$, we require $C \in \mathbf{B}^+$, where $\mathbf{B}^+$ is the smallest set of concepts such that:
  – $\mathsf{N_C} \subseteq \mathbf{B}^+$,
  – $\{o'\} \in \mathbf{B}^+$, for all $o' \in \mathsf{N_I}$,
  – $\exists r.\top \in \mathbf{B}^+$, for all roles r,
  – $\{B_1 \sqcap B_2, B_1 \sqcup B_2, \neg B_1\} \subseteq \mathbf{B}^+$, for all $B_1, B_2 \in \mathbf{B}^+$.
– Formulae and KBs are defined as for $\mathcal{ALCHOIQ}br$, but the operator $\dot{\neg}$ may occur only in front of assertions.

A *DL-Lite*$_\mathcal{R}$ KB $\mathcal{K}$ is a *DL-Lite*$^+_\mathcal{R}$ KB that satisfies the following restrictions:

– $\mathcal{K}$ is a *conjunction* of inclusions and assertions, and
– all assertions in $\mathcal{K}$ are *basic assertions* of the forms $o : A$ with $A \in \mathsf{N_C}$, and $(o, o') : p$ with $p \in \mathsf{N_R}$. $\triangleleft$

We next characterize the complexity of finite satisfiability in *DL-Lite*$^+_\mathcal{R}$.

THEOREM 5.7. *Finite satisfiability of DL-Lite*$^+_\mathcal{R}$ *KBs is* NP-*complete.*

PROOF. NP-hardness is immediate (e.g., by a reduction from propositional satisfiability). For membership in NP, we simply show that for any satisfiable *DL-Lite*$^+_\mathcal{R}$ KB $\mathcal{K}$ there is a model $\mathcal{I}$ of $\mathcal{K}$ such that *(i)* $Q^\mathcal{I} = \emptyset$ for all concept and role names Q that do not appear in $\mathcal{K}$, and *(ii)* $|\Delta^\mathcal{I}| \leq m + m \cdot k + 2 \cdot k$, where m is the number of individuals and k the number of (possibly complex) roles that appear in $\mathcal{K}$. A non-deterministic algorithm running in polynomial time in the size of $\mathcal{K}$ is then apparent: simply guess an interpretation $\mathcal{I}$ with $|\Delta^\mathcal{I}| \leq m + m \cdot k + 2 \cdot k$, and then check that $\mathcal{I}$ is a model of $\mathcal{K}$ (model checking requires only polynomial time because *DL-Lite*$^+_\mathcal{R}$ KBs correspond to first-order formulas with a bounded—namely at most 2—number of variables [Vardi 1995]).

Consider a model $\mathcal{I} = \langle \Delta^\mathcal{I}, \cdot^\mathcal{I} \rangle$ of $\mathcal{K}$. Let $\mathcal{T}$ be the set of inclusions $\alpha_1 \sqsubseteq \alpha_2$ appearing in $\mathcal{K}$ such that $\mathcal{I} \models \alpha_1 \sqsubseteq \alpha_2$. Similarly, let $\mathcal{R}$ be the set of inclusions β appearing in $\mathcal{K}$ of the form $r_1 \sqsubseteq r_2$ or $r_1 \sqsubseteq \neg r_2$, with $\{r_1, r_2\} \subseteq \mathsf{N_R} \cup \{p^- \mid p \in \mathsf{N_R}\}$, such that $\mathcal{I} \models \beta$. Let $\mathcal{R}^*$ be the closure of $\mathcal{R}$ under the following rules:
(a) if $\alpha_1 \sqsubseteq \alpha_2 \in \mathcal{R}^*$, then $\alpha_1 \sqsubseteq \alpha_1 \in \mathcal{R}^*$ and $\alpha_2 \sqsubseteq \alpha_2 \in \mathcal{R}^*$,
(b) if $\alpha_1 \sqsubseteq \alpha_2 \in \mathcal{R}^*$ and $\alpha_2 \sqsubseteq \alpha_3 \in \mathcal{R}^*$, then $\alpha_1 \sqsubseteq \alpha_3 \in \mathcal{R}^*$, and
(c) if $\alpha_1 \sqsubseteq \alpha_2 \in \mathcal{R}^*$, then $\alpha_1^- \sqsubseteq \alpha_2^- \in \mathcal{R}^*$.

W.l.o.g., we can assume for all concept and role names Q, that if Q does not appear in $\mathcal{K}$, then $Q^\mathcal{I} = \emptyset$. Let $\Delta = \{o^\mathcal{I} \mid \text{individual } o \text{ appears in } \mathcal{K}\}$. Let Δ' be a $\subseteq$-minimal set such that $\Delta \subseteq \Delta'$, and such that the following rule is obeyed:

(*) if $e \in \Delta$, and $e \in (\exists r.\top)^\mathcal{I}$ for some (possibly complex) role r that occurs in $\mathcal{K}$, then there exists $e' \in \Delta'$ such that $(e, e') \in r^\mathcal{I}$.

Note that $|\Delta'| \leq |\Delta| + |\Delta| \cdot k$. In particular, $|\Delta'|$ is polynomial in the size of $\mathcal{K}$. Let $N = \{p \mid p \in \mathsf{N_R} \text{ occurs in } \mathcal{K}\} \cup \{p^- \mid p \in \mathsf{N_R} \text{ occurs in } \mathcal{K}\}$. For every $r \in N$ such that $(\exists r^-.\top)^\mathcal{I} \setminus \Delta \neq \emptyset$, choose an arbitrary element $s_r \in (\exists r^-.\top)^\mathcal{I} \setminus \Delta$. For a role $r \in N$ for which s_r is defined, we let $sink(r)$ denote s_r.

Construct an instance $\mathcal{J} = \langle \Delta^\mathcal{J}, \cdot^\mathcal{J} \rangle$ as follows:

(c1) $\Delta^\mathcal{J} = \Delta' \cup \{sink(r) \mid r \in N \text{ and } sink(r) \text{ is defined}\}$;
(c2) $o^\mathcal{J} = o^\mathcal{I}$ for all individuals o that appear in $\mathcal{K}$;
(c3) $A^\mathcal{J} = A^\mathcal{I} \cap \Delta^\mathcal{J}$, for all concept names A;
(c4) for all role names p, and each $\{e_1, e_2\} \subseteq \Delta^\mathcal{J}$, we have that $(e_1, e_2) \in p^\mathcal{J}$, if one of the following holds:
  (a) $(e_1, e_2) \in p^\mathcal{I}$,

(b) $e_2 = sink(r)$ for some $r \in N$ such that $r \sqsubseteq p \in \mathcal{R}^*$, and $e_1 \in (\exists r.\top)^{\mathcal{I}}$,
(c) $e_1 = sink(r)$ for some $r \in N$ such that $r \sqsubseteq p \in \mathcal{R}^*$, and $e_2 \in (\exists r^-.\top)^{\mathcal{I}}$.

We now show that $\mathcal{J}$ remains a model of $\mathcal{K}$. For this, it suffices to show that: *(I)* for every concept inclusion $C_1 \sqsubseteq C_2$ that appears in $\mathcal{K}$, $\mathcal{I} \models C_1 \sqsubseteq C_2$ implies $\mathcal{J} \models C_1 \sqsubseteq C_2$, *(II)* for every role inclusion $r_1 \sqsubseteq r_2$ that appears in $\mathcal{K}$, $\mathcal{I} \models r_1 \sqsubseteq r_2$ implies $\mathcal{J} \models r_1 \sqsubseteq r_2$, and *(III)* for every assertion β that appears in $\mathcal{K}$, $\mathcal{I} \models \beta$ if and only if $\mathcal{J} \models \beta$. The points *(I)–(III)* are sufficient because in $\mathcal{K}$ the operator $\dot\neg$ is allowed to occur only in front of assertions.

We first prove Claim *(I)*. Assume $C_1 \sqsubseteq C_2$ occurs in $\mathcal{K}$, $\mathcal{I} \models C_1 \sqsubseteq C_2$, and there is $e \in C_1^{\mathcal{J}}$. We argue that $e \in C_2^{\mathcal{J}}$. First note that $C_1 \in \mathsf{N_C} \cup \{\exists p.\top \mid p \in \mathsf{N_R}\} \cup \{\exists p^-.\top \mid p \in \mathsf{N_R}\}$. In case $C_1 \in \mathsf{N_C}$, due to condition (c3), $e \in C_1^{\mathcal{I}}$. In case $C_1 \in \{\exists p.\top \mid p \in \mathsf{N_R}\}$, due to conditions (c4.a) and (c4.b), $e \in C_1^{\mathcal{I}}$. In case $C_1 \in \{\exists p^-.\top \mid p \in \mathsf{N_R}\}$, due to conditions (c4.a) and (c4.c), $e \in C_1^{\mathcal{I}}$. Thus, $e \in C_1^{\mathcal{I}}$ irrespectively of the shape of C_1. Since $\mathcal{I} \models C_1 \sqsubseteq C_2$, we have $e \in C_2^{\mathcal{I}}$. There are 6 cases:

— $C_2 \in \mathsf{N_C}$. Then $e \in C_2^{\mathcal{J}}$ due to condition (c3).
— $C_2 \in \{\exists p.\top \mid p \in \mathsf{N_R}\}$. Then $e \in C_2^{\mathcal{J}}$ due to condition (c4.b).
— $C_2 \in \{\exists p^-.\top \mid p \in \mathsf{N_R}\}$. Then $e \in C_2^{\mathcal{J}}$ due to condition (c4.c).
— $C_2 = \neg A$ for some $A \in \mathsf{N_C}$. Then $e \in C_2^{\mathcal{J}}$ due to condition (c3).
— $C_2 = \neg(\exists p.\top)$ for some $p \in \mathsf{N_R}$. Towards a contradiction, suppose $e \notin (\neg\exists p.\top)^{\mathcal{J}}$, i.e., there is $e' \in \Delta^{\mathcal{J}}$ such that $(e, e') \in p^{\mathcal{J}}$. Then, due to conditions (c4.a) and (c4.b) we obtain that $e \in (\exists p.\top)^{\mathcal{I}}$, i.e., $e \notin C_2^{\mathcal{I}}$.
— $C_2 = \neg(\exists p^-.\top)$ for some $p \in \mathsf{N_R}$. Similar to the point above, due to conditions (c4.a) and (c4.c), we obtain that $e \in C_2$.

Next, we prove Claim *(II)*. Suppose there exist e, e' in $\Delta^{\mathcal{J}}$ such that $(e, e') \in r_1^{\mathcal{J}}$. There are two cases:

— $(e, e') \in r_1^{\mathcal{I}}$. Then $(e, e') \in r_2^{\mathcal{J}}$ due to condition (c4.a) and the fact that $(e, e') \in r_2^{\mathcal{I}}$.
— $(e, e') \notin r_1^{\mathcal{I}}$. Suppose $r_1 \in \mathsf{N_R}$. Then it must be the case that $e' = sink(r)$ for some r such that $(e, e') \in r^{\mathcal{I}}$, e' is anonymous and $r \sqsubseteq r_1 \in \mathcal{R}^*$. Since $r \sqsubseteq r_2 \in \mathcal{R}^*$, due to condition (c4.b) it must be the case that $(e, sink(r)) \in r_2^{\mathcal{J}}$, i.e., $(e, e') \in r_2^{\mathcal{J}}$. Showing $(e, e') \in r_2^{\mathcal{J}}$ in case $r_1 \in \{p^- \mid p \in \mathsf{N_R}\}$ is analogous (the condition (c4.c) must be used instead of (c4.b)).

Finally, we prove Claim *(III)*. First note that due to the construction of $\mathcal{J}$, we have that $\mathcal{J} \models (o, o') : r$ iff $\mathcal{I} \models (o, o') : r$ holds for all role names r and all individuals o, o' that occur in $\mathcal{K}$. Thus Claim *(III)* trivially holds in case β in the claim is a role membership assertion. Suppose β is an assertion $o : C$ with $C \in \mathbf{B}^+$ (recall Definition 5.6). Due to conditions (c2) and (c3), it only remains to make sure that for any invidual o and any (possibly complex) role r that occur in $\mathcal{K}$, we have $o^{\mathcal{I}} \in (\exists r.\top)^{\mathcal{I}}$ iff $o^{\mathcal{J}} \in (\exists r.\top)^{\mathcal{J}}$. The "if" direction follows direction from condition (c4) in the construction of $\mathcal{J}$. The "only if" direction follows from the fact that Δ' satisfies the rule (*). □

To define the restricted setting, we also slightly limit the action language by imposing that the condition $\mathcal{K}$ in actions of the form $\mathcal{K} \,?\, \alpha_1 [\![\alpha_2]\!]$ can be a Boolean combination of assertions only. We observe that most of the examples presented in this paper use the latter type of actions.

Definition 5.8. A (complex) action α is called *simple* if *(i)* no (concept or role) inclusions occur in α, and *(ii)* all concepts of α are from $\mathbf{B}^+$. ◁

DL-Lite$^+_{\mathcal{R}}$ is expressive enough to allow us to reduce static verification for simple actions to finite unsatisfiability, and similarly as above, we can use a non-deterministic polynomial time many-one reduction (from the complement of static verification to finite unsatisfiability) to obtain a coNP upper bound on the complexity of static verification, which is tight.

THEOREM 5.9. *The static verification problem for DL-Lite*$^+_{\mathcal{R}}$ *KBs and simple actions is* coNP-*complete.*

PROOF. The lower bound follows from Theorem 5.5. Alternatively, it can be proved by a reduction from finite unsatisfiability in *DL-Lite*$^+_{\mathcal{R}}$, employing the same reduction as in the proof of Theorem 5.4.

For the upper bound, consider a *DL-Lite*$^+_{\mathcal{R}}$ KB $\mathcal{K}$ and a simple action α. We proceed analogously to the proof of Theorem 5.4. From Theorem 5.2 we know that α is not $\mathcal{K}$-preserving iff $\mathcal{K} \wedge \dot{\neg}\mathsf{TR}_{\alpha^*}(\mathcal{K})$ is finitely satisfiable. Moreover, we have shown that $\mathcal{K} \wedge \dot{\neg}\mathsf{TR}_{\alpha^*}(\mathcal{K})$ is finitely satisfiable iff there exists a $\mathcal{K}' \in \overline{\mathbf{TR}}_{\alpha^*}(\mathcal{K})$ such that $\mathcal{K} \wedge \mathcal{K}'$ is finitely satisfiable, and $\mathcal{K}'$ can be obtained non-deterministically in polynomial time and is of size polynomial in α and $\mathcal{K}$. The KB $\mathcal{K}'$ is not a *DL-Lite*$^+_{\mathcal{R}}$ KB, but it can be transformed into an equisatisfiable *DL-Lite*$^+_{\mathcal{R}}$ KB in linear time. To this end, turn $\mathcal{K}'$ into negation normal form, i.e., push $\dot{\neg}$ inside so that $\dot{\neg}$ occurs in front of inclusions and assertions only. Then, in the resulting KB $\mathcal{K}'$, replace every occurrence of $\dot{\neg}(B_1 \sqsubseteq B_2)$ by $o : B_1 \sqcap \neg B_2$ and every occurrence of $\dot{\neg}(r_1 \sqsubseteq r_2)$ by $(o, o') : r_1 \setminus r_2$, where o, o' are fresh individuals. Clearly, the above transformations preserve satisfiability. Moreover, since in $\mathcal{K}$ the operator $\dot{\neg}$ may occur only in front of assertions, and α is simple, every inclusion in the resulting $\mathcal{K}'$ already appears in $\mathcal{K}$. This implies that $\mathcal{K}'$ is a *DL-Lite*$^+_{\mathcal{R}}$ KB, as desired. □

6. PLANNING

We have focused so far on ensuring that the satisfaction of constraints is preserved when we evolve GSD. But additionally to these constraints, there may be desirable states of the GSD that we want to achieve, or undesirable ones that we want to avoid. For instance, one may want to ensure that a finished project is never made active again, or that an employee that is removed from the table recording active employees is stored as a former employee, and not eliminated from the database. This raises several problems, such as deciding if there exists a sequence of actions to reach a state with certain properties, or whether a given sequence of actions always ensures that a state with certain properties is reached. In this section, we consider these problems and formalize them by means of *automated planning*, which is a major topic in artificial intelligence.

We use DLs to describe states of KBs, which may act as goals or pre-conditions. A *plan* is a sequence of actions taken *from a given finite set*, whose execution leads an agent from the current state to a state that satisfies a given goal.

Definition 6.1. Let $\mathcal{I} = \langle \Delta^{\mathcal{I}}, \cdot^{\mathcal{I}} \rangle$ be a finite instance, Act a finite set of actions, and $\mathcal{K}$ a KB (the *goal* KB). A finite sequence $\langle \alpha_1, \ldots, \alpha_n \rangle$ of ground instances of actions from Act is called a *plan for* $\mathcal{K}$ *from* $\mathcal{I}$ (of *length* n), if there exists a finite instance $\mathcal{I}^* = \langle \Delta^{\mathcal{I}^*}, \cdot^{\mathcal{I}^*} \rangle$ such that $S_{\alpha_1 \cdots \alpha_n}(\mathcal{I}^*) \models \mathcal{K}$, where $\Delta^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}^*}$ and $A^{\mathcal{I}} = A^{\mathcal{I}^*}$ for each $A \in \mathsf{N_C}$, $p^{\mathcal{I}} = p^{\mathcal{I}^*}$ for each $p \in \mathsf{N_R}$, and $o^{\mathcal{I}} = o^{\mathcal{I}^*}$ for each $o \in \mathsf{N_I}$ that occurs in Act and $\mathcal{K}$. ◁

Recall that actions in our setting do not modify the domain of an instance. To support unbounded introduction of values in the data, the definition of planning above allows for the domain to be expanded a-priori with a finite set of fresh domain elements.

We can now define the first planning problems that we study:

(P1) Given a finite set Act of actions, a finite instance $\mathcal{I}$, and a goal KB $\mathcal{K}$, does there exist a plan for $\mathcal{K}$ from $\mathcal{I}$?

(P2) Given a finite set Act of actions and a pair $\mathcal{K}_{pre}$, $\mathcal{K}$ of formulae, does there exist a substitution σ and a plan for $\sigma(\mathcal{K})$ from some finite instance $\mathcal{I}$ with $\mathcal{I} \models \sigma(\mathcal{K}_{pre})$?

(P1) is the classic plan existence problem, formulated in the setting of GSD. Problem (P2) also aims at deciding plan existence, but rather than the full actual state of the data, we have as an input a *pre-condition* KB, and we are interested in deciding the existence of a plan from some of its models. To see the relevance of (P2), consider the complementary problem: a 'no' instance of (P2) means that, from every relevant initial state, (undesired) goals cannot be reached. For instance, $\mathcal{K}_{pre} = \mathcal{K}_{ic} \wedge (x : \mathsf{FinishedPrj})$ and $\mathcal{K} = x : \mathsf{ActivePrj}$ may be used to check whether starting with GSD that satisfy the integrity constraints and contain some finished project p, it is possible to make p an active project again. The following simple example illustrates the plan existence problems (P1) and (P2).

Example 6.2. Recall again the instance $\mathcal{I}_1$, and consider the following *goal* KB, which requires that p_1 is not an active project, and that e_1 is an employee.

$$\mathcal{K} \;=\; \dot{\neg}(\mathsf{p}_1 : \mathsf{ActivePrj}) \wedge (\mathsf{e}_1 : \mathsf{Empl})$$

Assume that Act consists of the action α_2 from Example 3.4 (which we recall for convenience), and of the following α_3:

$$\begin{aligned}\alpha_2 = {} & (x : \mathsf{Empl} \wedge y : \mathsf{Prj} \wedge z : \mathsf{Prj} \wedge (x,y) : \mathsf{worksFor})\,?\\ & \quad ((\mathsf{worksFor} \ominus \{(x,y)\}) \cdot (\mathsf{worksFor} \oplus \{(x,z)\}))\\ \alpha_3 = {} & (\mathsf{ActivePrj} \ominus \{y\}) \cdot (\mathsf{FinishedPrj} \oplus \{y\}) \cdot (\mathsf{Empl} \ominus \forall \mathsf{worksFor}.\{y\}) \cdot \\ & (\mathsf{worksFor} \ominus \mathsf{worksFor}|_{\{y\}})\end{aligned}$$

A plan for $\mathcal{K}$ from $\mathcal{I}_1$ is the sequence of actions $\langle \sigma(\alpha_2), \sigma(\alpha_3)\rangle$, where σ is the substitution from Example 3.4, which maps x to e_1, y to p_1, and z to p_2. To obtain $S_{\sigma(\alpha_2)\cdot\sigma(\alpha_3)}(\mathcal{I}_1)$, we first apply $\sigma(\alpha_2)$ to $\mathcal{I}_1$. The resulting instance $S_{\sigma(\alpha_2)}(\mathcal{I}_1)$ is as follows:

$$\begin{aligned}\mathsf{Prj}^{S_{\sigma(\alpha_2)}(\mathcal{I}_1)} &= \{p_1, p_2\}\\ \mathsf{ActivePrj}^{S_{\sigma(\alpha_2)}(\mathcal{I}_1)} &= \{p_1, p_2\}\\ \mathsf{Empl}^{S_{\sigma(\alpha_2)}(\mathcal{I}_1)} &= \{e_1, e_3, e_7\}\\ \mathsf{FinishedPrj}^{S_{\sigma(\alpha_2)}(\mathcal{I}_1)} &= \{\}\\ \mathsf{worksFor}^{S_{\sigma(\alpha_2)}(\mathcal{I}_1)} &= \{(e_1,p_2),(e_3,p_1),(e_7,p_1),(e_7,p_2)\}\end{aligned}$$

Finally, we apply $\sigma(\alpha_3)$ to $S_{\sigma(\alpha_2)}(\mathcal{I}_1)$, and obtain the instance $S_{\sigma(\alpha_3)}(S_{\sigma(\alpha_2)}(\mathcal{I}_1)) = S_{\sigma(\alpha_2)\cdot\sigma(\alpha_3)}(\mathcal{I}_1)$ below, which reflects the status of the data after applying $\langle \sigma(\alpha_2), \sigma(\alpha_3)\rangle$ to $\mathcal{I}_1$:

$$\begin{aligned}\mathsf{Prj}^{S_{\sigma(\alpha_2)\cdot\sigma(\alpha_3)}(\mathcal{I}_1)} &= \{p_1, p_2\}\\ \mathsf{ActivePrj}^{S_{\sigma(\alpha_2)\cdot\sigma(\alpha_3)}(\mathcal{I}_1)} &= \{p_2\}\\ \mathsf{Empl}^{S_{\sigma(\alpha_2)\cdot\sigma(\alpha_3)}(\mathcal{I}_1)} &= \{e_1, e_7\}\\ \mathsf{FinishedPrj}^{S_{\sigma(\alpha_2)\cdot\sigma(\alpha_3)}(\mathcal{I}_1)} &= \{p_1\}\\ \mathsf{worksFor}^{S_{\sigma(\alpha_2)\cdot\sigma(\alpha_3)}(\mathcal{I}_1)} &= \{(e_1,p_2),(e_7,p_2)\}\end{aligned}$$

Clearly, $S_{\sigma(\alpha_2)\cdot\sigma(\alpha_3)}(\mathcal{I}_1)$ satisfies the goal KB as desired, that is, in $S_{\sigma(\alpha_2)\cdot\sigma(\alpha_3)}(\mathcal{I}_1)$, p_1 is not an active project and e_1 is an employee.

For (P2), assume that additionally the following formula $\mathcal{K}_{pre}$ is given as input:

$$\mathcal{K}_{pre} \;=\; (x : \mathsf{Empl}) \wedge (y : \mathsf{ActivePrj}) \wedge ((x,y) : \mathsf{worksFor})$$

The above σ and $\langle\sigma(\alpha_2),\sigma(\alpha_3)\rangle$ are the desired substitution and plan. Indeed, $\mathcal{I}_1 \models \sigma(\mathcal{K}_{pre})$, and $\langle\sigma(\alpha_2),\sigma(\alpha_3)\rangle$ is a plan for $\sigma(\mathcal{K})$ from $\mathcal{I}_1$. Note that in this case $\sigma(\mathcal{K}) = \mathcal{K}$, but this does not need to happen in general. Indeed, $\langle\sigma(\alpha_2),\sigma(\alpha_3)\rangle$ would also be an answer if we were given as input goal $\mathcal{K}' = \dot{\neg}(y : \mathsf{ActivePrj}) \wedge (x : \mathsf{Empl})$ instead of $\mathcal{K}$.

Unfortunately, the above problems are undecidable in general, which we show by a reduction from the Word problem for Turing machines. We note that this undecidability result, and all lower bounds in this section, hold for the restricted case of actions with pre-conditions of the form $\mathcal{K}\,?\,\alpha_1$, rather than full conditional actions of the form $\mathcal{K}\,?\,\alpha_1[\![\alpha_2]\!]$.

THEOREM 6.3. *The problems (P1) and (P2) are undecidable, already for DL-Lite$_\mathcal{R}$ KBs and simple actions.*

PROOF. The proof is by a reduction from the Word problem. We reduce to (P1) and to (P2) the problem of deciding whether a deterministic Turing machine M accepts a word $w \in \{0,1\}^*$.

For (P1), assume that M is given by a tuple $M = (Q, \delta, q_0, q_a, q_r)$, where Q is a set of states, $\delta : \{0,1,b\} \times Q \to \{0,1,b\} \times Q \times \{+1,-1\}$ is the transition function, b is the blank symbol, $q_0 \in Q$ is the initial state, and $q_a \in Q$ is the accepting state. We can assume w.l.o.g. that before accepting the input the machine returns the read/write head to the initial position.

Intuitively, we define an action that implements the effects of each possible transition from δ. We also have a pair of actions that "extend" the tape with blank symbols as needed. For the reduction we use the role $next$, and concept names Sym_0, Sym_1, Sym_b, and St_q for each $q \in Q$.

The set Act of actions is defined as follows. For every $(\sigma, q) \in \{0,1,b\} \times Q$ with $\delta(\sigma,q) = (\sigma', q', D)$ we have the action

$$\begin{aligned}\alpha_{\sigma,q} \;=\; &((x_1,x_2) : next \wedge x_2 : Sym_\sigma \wedge x_2 : St_q \wedge (x_2,x_3) : next)\,?\\ &\quad((Sym_\sigma \ominus \{x_2\}) \cdot (Sym_{\sigma'} \oplus \{x_2\}) \cdot (St_q \ominus \{x_2\}) \cdot (St_{q'} \oplus \{x_{2+D}\})).\end{aligned}$$

To extend the tape with blank symbols, we have the actions α_r and α_ℓ. In particular,

$$\begin{aligned}\alpha_r \;=\; &(x : (Sym_0 \sqcup Sym_1 \sqcup Sym_b) \wedge y : \neg(Sym_0 \sqcup Sym_1 \sqcup Sym_b))\,?\\ &\quad((next \oplus \{(x,y)\}) \cdot (Sym_b \oplus \{y\})).\end{aligned}$$

The action α_ℓ is obtained from α_r by replacing $(next \oplus \{(x,y)\})$ with $(next \oplus \{(y,x)\})$. Finally, we have an initialization action α_{init}, which stores the initial configuration of M in the database. In particular,

$$\begin{aligned}\alpha_{init} \;=\; &(\textstyle\bigwedge_{1\leq i\neq j\leq m} a_i : \neg\{a_j\} \wedge a_1 : \neg(Sym_0 \sqcup Sym_1 \sqcup Sym_b))\,?\\ &\quad((St_{q_0} \oplus \{a_1\}) \cdot (Sym_{\sigma_1} \oplus \{a_1\}) \cdot (Sym_{\sigma_2} \oplus \{a_2\}) \cdots (Sym_{\sigma_m} \oplus \{a_m\}) \cdot\\ &\quad(next \oplus \{(a_1,a_2)\}) \cdot (next \oplus \{(a_2,a_3)\}) \cdots (next \oplus \{(a_{m-1},a_m)\}))\end{aligned}$$

where $w = \sigma_1 \cdots \sigma_m$. We let $\mathcal{K} = a_1 : St_{q_a}$ and the initial database $\mathcal{I}$ be empty, i.e., no domain element participates in a concept or a role.

It can be easily seen that the reduction is correct. If $\mathcal{K}$ has a plan, then M accepts w. Conversely, if M accepts w, then it accepts w within some number s of steps. One can verify that expanding the domain of $\mathcal{I}$ with s fresh elements is sufficient to find a plan for $\mathcal{K}$ using the actions in Act.

The above reduction also applies to (P2). It suffices to define a pre-condition KB $\mathcal{K}_{pre}$ that describes the above $\mathcal{I}$. Simply let $\mathcal{K}_{pre}$ be the conjunction of $(Sym_0 \sqcup Sym_1 \sqcup Sym_b \sqcup \exists next.\top \sqcup \exists next^-.\top \sqsubseteq \bot)$ and $\bigsqcup_{q\in Q} St_q \sqsubseteq \bot$. $\square$

We observe that the proof above works for *DL-Lite*$_{\mathcal{R}}$ KBs, that is $\mathcal{K}$ and $\mathcal{K}_{pre}$ are *DL-Lite*$_{\mathcal{R}}$ KBs, but we use *DL-Lite*$^{+}_{\mathcal{R}}$ assertions in preconditions of the simple actions.

Intuitively, problem (P1) is undecidable because we cannot know how many fresh objects need to be added to the domain of $\mathcal{I}$, but it becomes decidable if the size of Δ in Definition 6.1 is bounded. It is not difficult to see that problem (P2) remains undecidable even if the domain is assumed to be fixed (as the problem definition quantifies existentially over instances, one can choose instances with sufficiently large domains). However, also (P2) becomes decidable if we place a bound on the length of plans. More precisely, the following problems are decidable.

(P1b) Given a finite set Act of actions, a finite instance $\mathcal{I}$, a goal KB $\mathcal{K}$, and a positive integer ℓ, does there exist a plan for $\mathcal{K}$ from $\mathcal{I}$ where $|\Delta^{\mathcal{I}^*}| \leq \ell$?

(P2b) Given a finite set Act of actions, a pair $\mathcal{K}_{pre}$, $\mathcal{K}$ of formulae, and a positive integer ℓ, does there exist a substitution σ and a plan of length at most ℓ for $\sigma(\mathcal{K})$ from some finite instance $\mathcal{I}$ with $\mathcal{I} \models \sigma(\mathcal{K}_{pre})$?

Example 6.4. Recall Example 6.2. For $\ell \geq 2$, the substitution σ and the plan $\langle \sigma(\alpha_2), \sigma(\alpha_3) \rangle$ are still the desired solutions for the problems (P1b) and (P2b). However there does not exist a plan for $\ell = 1$. Now assume the goal KB is as follows:

$$\mathcal{K}_1 = \dot{\neg}(\mathsf{p}_1 : \mathsf{ActivePrj}) \wedge (\mathsf{e}_7 : \mathsf{Empl})$$

Then, $\langle \sigma'(\alpha_3) \rangle$ would be a plan of length one for $\sigma(\mathcal{K}_1)$ from instance $\mathcal{I}_1$, where σ' is the substitution with $\sigma'(x) = \mathsf{e}_7$ and $\sigma'(y) = \mathsf{p}_1$ and is such that $\mathcal{I}_1 \models \sigma'(\mathcal{K}_{pre})$.

We now study the complexity of these problems, assuming that the input bounds ℓ are coded in unary. The problem (P1b) can be solved in polynomial space, and thus is not harder than deciding the existence of a plan in standard automated planning formalisms such as propositional STRIPS [Bylander 1994], as established by the following result.

THEOREM 6.5. *The problem (P1b) is* PSPACE*-complete for* $\mathcal{ALCHOIQ}br$ *and* $\mathcal{ALCHOI}br$ *KBs.*

PROOF. It suffices to prove the lower bound for $\mathcal{ALCHOI}br$ and the upper bound for $\mathcal{ALCHOIQ}br$. To prove the PSPACE lower bound for $\mathcal{ALCHOI}br$, we can adapt the Turing Machine encoding that we used to show undecidability in Theorem 6.3, but for Turing machines whose space is bounded by a polynomial function p. Intuitively if the domain of an instance in the planning problem is bounded by ℓ then one can easily simulate Turing machines that use space bounded by ℓ. For a Turing machine M and a word w, we define the set Act of actions, the input instance $\mathcal{I}$ and the goal KB $\mathcal{K}$ as in the proof of Theorem 6.3 and we let the integer $\ell = p(|w|)$. Then M accepts w using $p(|w|)$ tape cells if and only if there exists a plan from an instance whose domain is bounded by $p(|w|)$.

For the upper bound we employ a non-deterministic polynomial space procedure that stores in memory a finite instance and non-deterministically applies actions until the goal is satisfied. Since the domain of each candidate instance is fixed and of size linear in the input, each of them can be represented in polynomial space. The number of possible instances is bounded by $s = 2^{r \cdot d^2 + c \cdot d}$, where r and c are respectively the number of roles and concepts appearing in the input set of actions, and d is the cardinality of the domain of the initial instance. Thus the procedure can be terminated after s many steps, without loss of completeness. We note that a counter that counts up to s can be implemented in polynomial space, and that model checking $\mathcal{ALCHOIQ}br$-formulae is feasible in polynomial space. □

We note that we could make the reduction even simpler by removing the actions α_r and α_ℓ that are used to extend the tape if we assume that the machine only uses the m cells where the input word w is written. The Word problem for such machines is also known to be PSPACE-hard.

Now we establish the complexity of (P2b), both in the general setting (i.e., when $\mathcal{K}_{pre}$ and $\mathcal{K}$ are in $\mathcal{ALCHOIQ}br$ or $\mathcal{ALCHOI}br$), and for the restricted case of *DL-Lite*$^+_\mathcal{R}$ KBs and simple actions. For (SV), considering the latter setting allowed us to reduce the complexity from coNEXPTIME and EXPTIME, respectively, to coNP. Here we obtain an analogous result.

THEOREM 6.6. *The problem (P2b) is* NEXPTIME*-complete for $\mathcal{ALCHOIQ}br$ KBs and* EXPTIME*-complete for $\mathcal{ALCHOI}br$ KBs. It is* NP*-complete if $\mathcal{K}_{pre}$ and $\mathcal{K}$ are expressed in DL-Lite$^+_\mathcal{R}$ and all actions in Act are simple.*

PROOF. The lower bounds can be inferred immediately from the complexity of static verification with KBs in $\mathcal{ALCHOIQ}br$ and $\mathcal{ALCHOI}br$ (Theorem 5.4) and *DL-Lite*$^+_\mathcal{R}$ (Theorem 5.5).

For the upper bound for $\mathcal{ALCHOIQ}br$, we first guess a variable substitution σ and a sequence $P = \langle \alpha_1, \ldots, \alpha_m \rangle$ of at most ℓ actions. By Theorem 4.2, it follows that P is a plan as desired iff $\sigma(\mathcal{K}_{pre}) \wedge \mathsf{TR}_{\alpha_1 \cdots \alpha_m}(\sigma(\mathcal{K}))$ is finitely satisfiable. Note that the core difference between this case and the one of Theorem 5.4 (and of Theorem 5.9) is that now the formula $\mathsf{TR}_{\alpha_1 \cdots \alpha_m}(\sigma(\mathcal{K}))$ is not negated and hence, intuitively, we need to decide the existence of an instance that satisfies the negation of *all* formulas in $\overline{\mathbf{TR}}_\alpha(\mathcal{K})$, rather than satisfying just one of them. To be able to check the finite satisfiability of $\sigma(\mathcal{K}_{pre}) \wedge \mathsf{TR}_{\alpha_1 \cdots \alpha_m}(\sigma(\mathcal{K}))$ within the desired bounds, we proceed similarly as above, and consider a procedure that non-deterministically builds a polynomial KB $\mathcal{K}'$ such that $\sigma(\mathcal{K}_{pre}) \wedge \mathcal{K}'$ is finitely satisfiable iff $\sigma(\mathcal{K}_{pre}) \wedge \mathsf{TR}_{\alpha_1 \cdots \alpha_m}(\sigma(\mathcal{K}))$ is so.

More in detail, we define a set $\mathbf{TR}^\wedge_\alpha(\mathcal{K})$ of KBs that is similar to $\overline{\mathbf{TR}}_\alpha(\mathcal{K})$, except that it contains the negation of the formulas in the latter, and uses conjunctions rather than implications for the conditional axioms.

$$\begin{aligned}
\mathbf{TR}^\wedge_\varepsilon(\mathcal{K}) &= \{\mathcal{K}\}\\
\mathbf{TR}^\wedge_{(A \oplus C)\cdot\alpha}(\mathcal{K}) &= \{\mathcal{K}'_{A \leftarrow A \sqcup C} \mid \mathcal{K}' \in \mathbf{TR}^\wedge_\alpha(\mathcal{K})\}\\
\mathbf{TR}^\wedge_{(A \ominus C)\cdot\alpha}(\mathcal{K}) &= \{\mathcal{K}'_{A \leftarrow A \sqcap \neg C} \mid \mathcal{K}' \in \mathbf{TR}^\wedge_\alpha(\mathcal{K})\}\\
\mathbf{TR}^\wedge_{(p \oplus r)\cdot\alpha}(\mathcal{K}) &= \{\mathcal{K}'_{p \leftarrow p \cup r} \mid \mathcal{K}' \in \mathbf{TR}^\wedge_\alpha(\mathcal{K})\}\\
\mathbf{TR}^\wedge_{(p \ominus r)\cdot\alpha}(\mathcal{K}) &= \{\mathcal{K}'_{p \leftarrow p \setminus r} \mid \mathcal{K}' \in \mathbf{TR}^\wedge_\alpha(\mathcal{K})\}\\
\mathbf{TR}^\wedge_{(\mathcal{K}_1 \,?\, \alpha_1 [\![\alpha_2]\!])\cdot\alpha}(\mathcal{K}) &= \{\mathcal{K}_1 \wedge \mathcal{K}' \mid \mathcal{K}' \in \mathbf{TR}^\wedge_{\alpha_1\cdot\alpha}(\mathcal{K})\} \cup \{\dot{\neg}\mathcal{K}_1 \wedge \mathcal{K}' \mid \mathcal{K}' \in \mathbf{TR}^\wedge_{\alpha_2\cdot\alpha}(\mathcal{K})\}
\end{aligned}$$

Similarly as above, $|\mathbf{TR}^\wedge_\alpha(\mathcal{K})|$ may be exponential but each $\mathcal{K}' \in \mathbf{TR}^\wedge_\alpha(\mathcal{K})$ is polynomial and can be built in polynomial time. We show below the following claim:

(‡) For every $\mathcal{I}$ and every $\mathcal{K}$, we have that $\mathcal{I} \models \mathsf{TR}_\alpha(\mathcal{K})$ iff there exists some $\mathcal{K}' \in \mathbf{TR}^\wedge_\alpha(\mathcal{K})$ such that $\mathcal{I} \models \mathcal{K}'$.

With (‡) we can easily show that $\sigma(\mathcal{K}_{pre}) \wedge \mathsf{TR}_{\alpha_1 \cdots \alpha_m}(\sigma(\mathcal{K}))$ is finitely satisfiable iff there exists some $\mathcal{K}' \in \mathbf{TR}^\wedge_{\alpha_1 \cdots \alpha_m}(\sigma(\mathcal{K}))$ such that $\sigma(\mathcal{K}_{pre}) \wedge \mathcal{K}'$ is finitely satisfiable. For the 'only if' direction, assume that $\sigma(\mathcal{K}_{pre}) \wedge \mathsf{TR}_{\alpha_1 \cdots \alpha_m}(\sigma(\mathcal{K}))$ is finitely satisfiable. Then there exists some finite $\mathcal{I}$ such that $\mathcal{I} \models \sigma(\mathcal{K}_{pre})$ and $\mathcal{I} \models \mathsf{TR}_{\alpha_1 \cdots \alpha_m}(\sigma(\mathcal{K}))$. By (‡), for this $\mathcal{I}$ we have that $\mathcal{I} \models \mathsf{TR}_{\alpha_1 \cdots \alpha_m}(\sigma(\mathcal{K}))$ iff there is some $\mathcal{K}' \in \mathbf{TR}^\wedge_{\alpha_1 \cdots \alpha_m}(\sigma(\mathcal{K}))$ such that $\mathcal{I} \models \mathcal{K}'$. We choose this $\mathcal{K}'$. It follows that $\mathcal{I} \models \mathcal{K}'$ and, since $\mathcal{I} \models \sigma(\mathcal{K}_{pre})$, we can conclude that $\sigma(\mathcal{K}_{pre}) \wedge \mathcal{K}'$ is finitely satisfiable. For the other direction, let $\mathcal{K}' \in \mathbf{TR}^\wedge_{\alpha_1 \cdots \alpha_m}(\sigma(\mathcal{K}))$ be such that $\sigma(\mathcal{K}_{pre}) \wedge \mathcal{K}'$ is finitely satisfiable. Then there exists some finite $\mathcal{I}$ such that $\mathcal{I} \models \sigma(\mathcal{K}_{pre})$ and $\mathcal{I} \models \mathcal{K}'$. By (‡), for this $\mathcal{I}$ it follows that $\mathcal{I} \models \mathsf{TR}_{\alpha}(\sigma(\mathcal{K}))$, and since $\mathcal{I} \models \sigma(\mathcal{K}_{pre})$, we have that $\sigma(\mathcal{K}_{pre}) \wedge \mathsf{TR}_{\alpha_1 \cdots \alpha_m}(\sigma(\mathcal{K}))$ is finitely satisfiable. Having shown this, the upper bound follows directly from the complexity of deciding finite satisfiability of $\sigma(\mathcal{K}_{pre}) \wedge \mathcal{K}'$, and the fact that $\mathcal{K}'$ is of polynomial size and can be obtained non-deterministically in polynomial time.

It is only left to show (‡), which we do by induction on $\ell(\alpha)$ (as defined in the proof of Theorem 4.2). The base case is trivial, since for $\alpha = \varepsilon$ we have $\mathbf{TR}^{\wedge}_{\alpha}(\mathcal{K}) = \{\mathcal{K}\}$ and $\mathsf{TR}_{\alpha}(\mathcal{K}) = \mathcal{K}$, so we can set $\mathcal{K}' = \mathcal{K}$ and the claim follows.

For the cases of $\alpha' = (A \oplus C) \cdot \alpha$, $\alpha' = (A \ominus C) \cdot \alpha$, $\alpha' = (p \oplus r) \cdot \alpha$, and $\alpha' = (p \ominus r) \cdot \alpha$, we can proceed exactly as in the proof of Theorem 5.4. The remaining case is when $\alpha' = (\mathcal{K}_1 \,?\, \alpha_1[\![\alpha_2]\!]) \cdot \alpha$. For the 'only if' direction, assume that $\mathcal{I} \models \mathsf{TR}_{\alpha'}(\mathcal{K})$. In this case the choice of $\mathcal{K}'$ depends on $\mathcal{I}$. We distinguish two cases:

— If $\mathcal{I} \models \mathcal{K}_1$, since $\mathsf{TR}_{\alpha'}(\mathcal{K}) = (\dot{\neg}\mathcal{K}_1 \vee \mathsf{TR}_{\alpha_1 \cdot \alpha}(\mathcal{K})) \wedge (\mathcal{K}_1 \vee \mathsf{TR}_{\alpha_2 \cdot \alpha}(\mathcal{K}))$, we have that $\mathcal{I} \models \mathsf{TR}_{\alpha_1 \cdot \alpha}(\mathcal{K})$. By induction hypothesis, there exists $\mathcal{K}'' \in \mathbf{TR}^{\wedge}_{\alpha_1 \cdot \alpha}(\mathcal{K})$ such that $\mathcal{I} \models \mathcal{K}''$. We set $\mathcal{K}' = \mathcal{K}_1 \wedge \mathcal{K}''$, and since we have that $\mathcal{K}' \in \mathbf{TR}^{\wedge}_{\alpha'}(\mathcal{K})$ by definition, the claim follows.

— Otherwise, if $\mathcal{I} \models \dot{\neg}\mathcal{K}_1$, we have that $\mathcal{I} \models \mathsf{TR}_{\alpha_2 \cdot \alpha}(\mathcal{K})$. By induction hypothesis, there exists $\mathcal{K}'' \in \mathbf{TR}^{\wedge}_{\alpha_2 \cdot \alpha}(\mathcal{K})$ such that $\mathcal{I} \models \mathcal{K}''$. We set $\mathcal{K}' = \dot{\neg}\mathcal{K}_1 \wedge \mathcal{K}''$, and since we have again that $\mathcal{K}' \in \mathbf{TR}^{\wedge}_{\alpha'}(\mathcal{K})$ by definition, the claim follows.

For the 'if' direction, assume that there exists some $\mathcal{K}' \in \mathbf{TR}^{\wedge}_{\alpha'}(\mathcal{K})$ such that $\mathcal{I} \models \mathcal{K}'$. We distinguish again two cases:

— If $\mathcal{K}' = \mathcal{K}_1 \wedge \mathcal{K}''$, for some $\mathcal{K}'' \in \mathbf{TR}^{\wedge}_{\alpha_1 \cdot \alpha}(\mathcal{K})$, then $\mathcal{I} \models \mathcal{K}_1$, and by induction hypothesis, $\mathcal{I} \models \mathsf{TR}_{\alpha_1 \cdot \alpha}(\mathcal{K})$. Hence, $\mathcal{I} \models (\dot{\neg}\mathcal{K}_1 \vee \mathsf{TR}_{\alpha_1 \cdot \alpha}(\mathcal{K})) \wedge (\mathcal{K}_1 \vee \mathsf{TR}_{\alpha_2 \cdot \alpha}(\mathcal{K}))$, and since $\mathsf{TR}_{\alpha'}(\mathcal{K}) = (\dot{\neg}\mathcal{K}_1 \vee \mathsf{TR}_{\alpha_1 \cdot \alpha}(\mathcal{K})) \wedge (\mathcal{K}_1 \vee \mathsf{TR}_{\alpha_2 \cdot \alpha}(\mathcal{K}))$, the claim follows.

— Otherwise, if $\mathcal{K}' = \dot{\neg}\mathcal{K}_1 \wedge \mathcal{K}''$, for some $\mathcal{K}'' \in \mathbf{TR}^{\wedge}_{\alpha_2 \cdot \alpha}(\mathcal{K})$, then $\mathcal{I} \models \dot{\neg}\mathcal{K}_1$, and by induction hypothesis, $\mathcal{I} \models \mathsf{TR}_{\alpha_2 \cdot \alpha}(\mathcal{K})$. Hence, $\mathcal{I} \models (\dot{\neg}\mathcal{K}_1 \vee \mathsf{TR}_{\alpha_1 \cdot \alpha}(\mathcal{K})) \wedge (\mathcal{K}_1 \vee \mathsf{TR}_{\alpha_2 \cdot \alpha}(\mathcal{K}))$, and again the claim follows.

This concludes the proof of the upper bound for $\mathcal{ALCHOIQ}br$.

For the upper bound for $\mathcal{ALCHOI}br$, one can build in exponential time all possible sequences of actions of length ℓ, and for each such sequence of actions there are exponentially many substitutions σ (up to renaming of fresh individuals). Then we iterate over the full set of such instantiated sequences of actions, which are exponentially many, and for each of them, reason as above.

For the NP upper bound for *DL-Lite*$^{+}_{\mathcal{R}}$ formulas and when all actions in Act are simple, we can proceed analogously to the proof of Theorem 5.9, and show that the formula $\mathcal{K}'$ can be transformed in linear time into an equisatisfiable *DL-Lite*$^{+}_{\mathcal{R}}$ one. □

Now we consider three problems that are related to ensuring plans that *always* achieve a given goal, no matter what the initial data is. They are variants of so-called *conformant* planning, which deals with planning under various forms of incomplete information. In our case, we assume that we have an incomplete description of the initial state, since we only know it satisfies a given pre-condition, but have no concrete instance.

The first of such problems is to *certify* that a candidate plan is indeed a plan for the goal, for every possible database satisfying the pre-condition.

(C) Given a sequence $P = \langle \alpha_1, \ldots, \alpha_n \rangle$ of actions, and formulae $\mathcal{K}_{pre}$ and $\mathcal{K}$, is $\sigma(P)$ a plan for $\sigma(\mathcal{K})$ from every finite instance $\mathcal{I}$ such that $\mathcal{I} \models \sigma(\mathcal{K}_{pre})$, for every possible substitution σ?

Example 6.7. Consider the sequence $\langle\alpha_3\rangle$ with the single action α_3 from Example 6.2, and the pre-condition $\mathcal{K}_{pre}$ and the goal $\mathcal{K}_2$ given as follows:

$$\begin{aligned}\mathcal{K}_{pre} &= x : \mathsf{Empl} \wedge y : \mathsf{ActivePrj} \wedge x : \geqslant 2\,\mathsf{worksFor}.\top \\ \mathcal{K}_2 &= x : \mathsf{Empl} \wedge \dot{\neg}(y : \mathsf{ActivePrj})\end{aligned}$$

The sequence $\sigma\langle\alpha_3\rangle$ is a plan for $\mathcal{K}_2$ for every substitution σ and for every input instance $\mathcal{I}$ with $\mathcal{I} \models \sigma(\mathcal{K}_{pre})$. Indeed, the pre-condition $\mathcal{K}_{pre}$ ensures that every such input instance $\mathcal{I}$ contains an active project $\sigma(y)$ and an employee $\sigma(x)$ that works for at least 2 projects. Then $\sigma(\alpha_3)$ removes $\sigma(y)$ from the active projects and removes all employees working only for $\sigma(y)$; the pre-condition makes sure that $\sigma(x)$ is not removed and remains an employee.

Finally, we are interested in the existence of a plan that always achieves the goal, for every possible state satisfying the pre-condition. Solving this problem corresponds to the automated *synthesis* of a program for reaching a certain condition. We formulate the problem with and without a bound on the length of the plans we are looking for.

(S) Given a finite set Act of actions and formulae $\mathcal{K}_{pre}$ and $\mathcal{K}$, does there exist a sequence P of actions such that $\sigma(P)$ is a plan for $\sigma(\mathcal{K})$ from every finite instance $\mathcal{I}$ with $\mathcal{I} \models \sigma(\mathcal{K}_{pre})$, for every possible substitution σ?

(Sb) Given a finite set Act of actions, formulae $\mathcal{K}_{pre}$ and $\mathcal{K}$, and a positive integer ℓ, does there exist a sequence P of actions such that $\sigma(P)$ is of length at most ℓ and is a plan for $\sigma(\mathcal{K})$ from every finite instance $\mathcal{I}$ with $\mathcal{I} \models \sigma(\mathcal{K}_{pre})$, for every possible substitution σ?

We observe that, although similar in appearance, there is a crucial difference between problems (P2) and (S), since the former asks for the existence of a plan for *some* substitution from *some* input instance, while the latter asks for the existence of a "universal" plan that reaches the goal for *every* substitution and for *every* instance that satisfies the pre-condition. The same holds for (P2b) and (Sb).

Example 6.8. Following Example 6.7, $\langle\alpha_3\rangle$ is the desired sequence for the formulas $\mathcal{K}_{pre}$ and $\mathcal{K}_2$ and the set $Act = \{\alpha_2, \alpha_3\}$ of actions from Example 6.2.

We conclude with the complexity of these problems:

THEOREM 6.9. *The following hold:*

— *Problem (S) is undecidable, already for DL-Lite$_{\mathcal{R}}$ KBs and simple actions.*
— *Problems (C) and (Sb) are* coNEXPTIME*-complete for $\mathcal{ALCHOIQ}br$ and* EXPTIME*-complete for $\mathcal{ALCHOI}br$.*
— *If $\mathcal{K}_{pre}$ and $\mathcal{K}$ are expressed in DL-Lite$^{+}_{\mathcal{R}}$ and all actions in Act are simple, then (C) is* coNP*-complete and (Sb) is* NP$^{\text{NP}}$*-complete.*

PROOF. Problem (S) can be shown to be undecidable by employing the same reduction as for (P2) in Theorem 6.3. The coNEXPTIME and EXPTIME lower bounds for (C) and (Sb) trivially follow from those for finite satisfiability in $\mathcal{ALCHOIQ}br$ and $\mathcal{ALCHOI}br$, respectively.

For the upper bounds, we first observe that (C) reduces to validity testing in $\mathcal{ALCHOIQ}br$ and $\mathcal{ALCHOI}br$, respectively: an instance of (C) (as described above) is positive iff the formula $\sigma(\dot{\neg}\mathcal{K}'_{pre}) \vee \mathsf{TR}_{\alpha_1\cdots\alpha_n}(\sigma(\mathcal{K}'))$ is valid, where $\mathcal{K}'_{pre}$ and $\mathcal{K}'$ are respectively obtained from $\mathcal{K}_{pre}$ and $\mathcal{K}$ by replacing every variable by a fresh individual. Deciding validity of $\sigma(\dot{\neg}\mathcal{K}'_{pre}) \vee \mathsf{TR}_{\alpha_1\cdots\alpha_n}(\sigma(\mathcal{K}'))$ in turn reduces to deciding whether $\sigma(\mathcal{K}'_{pre}) \wedge \dot{\neg}\mathsf{TR}_{\alpha_1\cdots\alpha_n}(\sigma(\mathcal{K}'))$ is finitely unsatisfiable. The upper bounds for (C) then follow from the NEXPTIME, EXPTIME, and NP upper bounds for the satisfiability of KBs of the form $\mathcal{K}' \wedge \dot{\neg}\mathsf{TR}_\alpha(\mathcal{K})$ shown in the proofs of Theorems 5.4 and 5.9.

Negative instances of (Sb), where $\mathcal{K}_{pre}$ is the pre-condition and $\mathcal{K}$ is the goal, can be recognized in NEXPTIME for $\mathcal{ALCHOIQ}br$ and in EXPTIME for $\mathcal{ALCHOI}br$. Such a test comprises building an exponentially large set of all candidate action sequences of length at most ℓ, and then making sure that each candidate is invalidated. That is, each candidate action sequence P induces an instance of (C), which can be shown negative in NEXPTIME for $\mathcal{ALCHOIQ}br$ and EXPTIME for $\mathcal{ALCHOI}br$. In the case of *DL-Lite*$^+_\mathcal{R}$ and simple actions, we can guess non-deterministically a sequence of actions of length at most ℓ and then check that the induced instance of (C) is positive, which is a test in coNP.

To show that the $\mathrm{NP}^{\mathrm{NP}}$ upper bound is tight, we use a polynomial time reduction from evaluating QBFs of the form $\gamma = \exists p_1 \ldots \exists p_n \forall q_1 \ldots \forall q_m.\psi$, where ψ is a Boolean combination over propositional variables $V = \{p_1, \ldots, p_n, q_1, \ldots, q_m\}$. We can assume that negation in ψ occurs in front of propositional variables only. For the reduction to (Sb), we employ concept names T and F, and individual names o_v for each propositional variable $v \in V$. We let $\mathcal{K}_{pre} = \big(\bigwedge_{1\leq i\leq n} o_{p_i} : \neg(T \sqcup F)\big) \wedge \big(\bigwedge_{1\leq i\leq m} o_{q_i} : (T \sqcup F) \sqcap (\neg T \sqcup \neg F)\big)$. Intuitively, each initial instance encodes an assignment for the variables $q_1, \ldots, q_m$, but does not say anything about $p_1, \ldots, p_n$. The latter is determined by choosing a candidate plan. To this end, for each $i \in \{1, \ldots, n\}$, we define the following actions:

$$\alpha_i = (o_{p_i} : \neg F) \,?\, (T \oplus \{o_{p_i}\}), \qquad \alpha'_i = (o_{p_i} : \neg T) \,?\, (F \oplus \{o_{p_i}\}).$$

We finally let $\ell = n$ and let $\mathcal{K}$ be the KB obtained from ψ by replacing each negative literal $\neg v$ by $(o_v : F)$ and each positive literal v by $(o_v : T)$. It is not difficult to see that γ evaluates to *true* iff the constructed instance of (Sb) is positive. □

7. RELATED WORK

We discuss separately related work from the databases and verification communities, and from the knowledge representation and reasoning community.

7.1. Databases and Automated Verification

In recent years, there has been an increasing body of work on verification of dynamic systems that access and modify databases [Calvanese et al. 2013a; Deutsch et al. 2014]. These systems analyze dynamic processes, while taking into account a data component, which can be a fully-fledged relational database that may evolve via actions that perform insertions, deletions, and that allow one to introduce fresh values [De Giacomo et al. 2012; Bagheri Hariri et al. 2013a]. The vast majority of works view the database modification as an encapsulated process defined declaratively in terms of pre- and post-conditions (see the aforementioned works and references therein). In practice, however, the actual data manipulation that leads the database from a state where the pre-condition holds to one ensuring the post-condition, needs to be realized by a procedural program, script, or transaction. Our work focuses on ensuring the correctness of the concrete actions manipulating the database. In that sense, it can be seen as a tool for verifying the correctness of the actual procedural implementation of an action description, or for automatically synthesizing such an implementation from a given set of basic actions. Another common feature of most existing works is that they employ modified model checking techniques to verify complex temporal properties, written in temporal logics or in the μ-calculus. In contrast, we only consider simpler formulas that state that some goal state will possibly or necessarily be reached. However, in several of the works the verification tasks are performed on a single initial data instance, while we quantify globally over all input databases.

Of the aforementioned works, the most similar to ours is the one by De Giacomo et al. [2012], who consider a similar setting that allows for inserting and deleting tuples during the database evolution. However, there are some crucial differences. For example, they consider conjunctive queries as basis for their action language, orthogonally to our $\mathcal{C}^2$-based formalism. Also, in our work we only modify those parts of the database that are explicitly changed by the actions. In contrast, in their setting, at every time instant, the whole data instance is substituted with a totally new one, provided that it satisfies the desired effects. That is, they do not guarantee *inertia*, and one instead faces the *frame problem* and needs to explicitely enumerate all tuples that should remain unchanged. In this sense, and despite the syntactic similarity, their actions are more in the declarative pre-/post-condition style described above.

A work that is close in spirit to ours is the one by Brenas et al. [2014], although their formalism is described as a programming language tailored for graph transformations, rather than an action language for database updates. Their language also allows one to manipulate graphs via additions and removals of individual edges or nodes, possibly also using the usual *if* and *while* constructs. The authors present a Hoare-style program verification calculus that relates their programming language with the DL $\mathcal{SROIQ}^\sigma$ (an extension of the standard DL $\mathcal{SROIQ}$ with *substitutions*, used to express *weakest pre-conditions*). Unfortunately, their work does not establish any complexity results, and correctness can only be verified under the assumption that programs terminate.

Very recently, an approach related to ours was presented by Itzhaky et al. [2016], who propose a scripting language that allows one to modify databases via embedded SQL statements. They use a logic closely related to FO^2, the two-variable fragment of first-order logic (without counting quantifiers) to express pre- and post-conditions, and tackle the problem of verifying a Hoare triple $\{\varphi_{pre}\}\alpha\{\varphi_{post}\}$ (which is defined analogously to our static verification, but allowing for different constraints to be verified before and after the program execution, as discussed in Section 5). Despite the different appearance, it is not hard to see that their scripting language is closely related, although orthogonal, to our action language. The authors reduce the mentioned verification problem to the satisfiability of a sentence in FO^2, by capturing weakest preconditions in a similar way as we have done in this paper, and in our previous works [Ahmetaj et al. 2014; Calvanese et al. 2016]. An interesting contribution of their work is that they have implemented this style of verification using a dedicated FO^2 solver, and have conducted preliminary experiments on selected use cases that suggest that this approach is actually feasible in practice.

Our work provides a tool for a-priori verification of updates on GSD, and is hence closely related to *verifying consistency of transactions*, a crucial problem that has been studied extensively in Databases. It has been considered for different kinds of transactions and constraints, over traditional relational databases [Sheard and Stemple 1989], object-oriented databases [Spelt and Balsters 1998; Bonner and Kifer 1994], and deductive databases [Kowalski et al. 1987], to name a few. Most of these works adopt expressive formalisms like (extensions of) first or higher order predicate logic [Bonner and Kifer 1994], or undecidable tailored languages [Sheard and Stemple 1989] to express the constraints and the operations on the data. Verification systems are often implemented using theorem provers, and complete algorithms cannot be devised.

7.2. Knowledge Representation and Reasoning About Change

Using DLs to understand the properties of systems while fully taking into account both structural and dynamic aspects is very challenging [Wolter and Zakharyaschev 1999]. Reasoning in DLs extended with a temporal dimension becomes quickly undecidable [Artale 2006], unless severe restrictions on the expressive power of the DL are imposed

[Artale et al. 2011]. An alternative approach to achieve decidability is to take a so-called "functional view of KBs" [Levesque 1984], according to which each state of the KB can be queried via logical implication, and the KB is progressed from one state to the next through forms of update [Calvanese et al. 2011]. This makes it possible (under suitable conditions) to *statically verify* (temporal) integrity constraints over the evolution of a system [Baader et al. 2012; Bagheri Hariri et al. 2013b].

Updating knowledge bases, and logic theories in general, is a classic topic in knowledge representation, discussed extensively in the literature, cf. [Fagin et al. 1986; Katsuno and Mendelzon 1991]. The updates described by our action language are similar in spirit to the knowledge base updates studied in other works, and in particular, the ABox updates considered by Liu et al. [2011] and Kharlamov et al. [2013]. As our updates are done directly on instances rather than on (the instance level of) knowledge bases, we do not encounter the expressibility and succinctness problems faced there.

As mentioned, the problems studied in Section 6 are closely related to automated planning, a topic extensively studied in AI. DLs have been employed to reason about actions, goals, and plans, as well as about the application domains in which planning is deployed, see the work by Gil [2005] and the references therein. Most relevant to us is the significant body of work on DL-based action languages [Baader et al. 2005; Milicic 2008; Baader et al. 2010; Liu et al. 2011; Baader and Zarrieß 2013]. In these formalisms, DL constructs are used to give conditions on the effects of action execution, which are often non-deterministic. A central problem considered is the *projection problem*, which consists in deciding whether every possible execution of an action sequence on a possibly incomplete state will lead to a state that satisfies a given property. Clearly, our certification problem (C), which involves an incomplete initial state, is a variation of the projection problem. However, we do not face the challenge of having to consider different possible executions of non-deterministic actions. Many of our other reasoning problems are similar to problems considered in these works, in different forms and contexts. A crucial difference is that our well-behaved action language allows us to obtain decidability even when we employ full-fledged TBoxes for specifying goals, pre-conditions, and domain constraints. For instance, the work by Liu et al. [2006] deals with general TBoxes, but *strong consistency*—a reasoning task close in spirit to static verification here—was shown to be undecidable in the proposed formalism.

8. CONCLUSIONS AND OUTLOOK

We have considered graph structured data that evolve as a result of updates expressed in a powerful yet well-behaved action language. We have studied several reasoning problems that support the static analysis of actions and their effects on the state of the data, for any possible input data instance. We have shown the decidability of most problems, and have characterized their computational complexity. In the cases where the general problem is undecidable, we have identified decidable restrictions. For the expressive DLs we have considered, most problems have a relatively high worst-case complexity. This is not surprising, given that we are considering static analysis tasks in which we reason about all possible data instances. However, to improve these bounds, we have also considered a suitable variant of *DL-Lite*. We believe this work provides powerful tools for the static analysis of the effects of executing complex actions on databases, possibly in the presence of integrity constraints expressed in rich DLs. Our upper bounds rely on a novel KB transformation technique, which enables us to reduce most of the reasoning tasks to finite (un)satisfiability in a DL. This calls for developing finite model reasoners for DLs (we note that $\mathcal{ALCHOIQ}br$ does not have the finite model property). It also remains to better understand the complexity of finite model reasoning in different variations of *DL-Lite*. E.g., extensions of *DL-Lite*$^{+}_{\mathcal{R}}$ with role functionality would be very useful in the context of graph structured data.

In this paper, we have used DLs for describing constraints on GSD, data states, and queries in basic actions. As explained in Section 2.3, all our results hold if $\mathcal{ALCHOIQ}br$ is replaced by $\mathcal{C}^2$. However, we chose to use DLs instead of $\mathcal{C}^2$, mainly because this allowed us to identify cases where reasoning has a lower worst-case complexity. Apart from illustrating how interesting forms of constraints on GSD can be expressed succinctly in the DL syntax, the role of DLs in this work provides further evidence of their suitability for data management tasks. We hope this will contribute to ongoing efforts to bring the knowledge representation and the data management communities closer to each other.

Many other challenges remain for future work. For example, generalizing the positive results on decidability to logics that can express more expressive constraints on the data, and richer languages for describing the updates. In the former direction, some of the authors have recently obtained analogous results for the case where both the constraints and the updates are written in a DL that supports regular expressions over roles as a role constructor [Calvanese et al. 2016]. This allows one, for example, to capture some variants of *regular path constraints* [Abiteboul and Vianu 1999; Grahne and Thomo 2003]. Unfortunately, allowing regular path queries in the basic action makes the formalism undecidable in general, but decidability is preserved if their use is restricted to act as tests in the concepts occurring in basic actions. Logics with powerful identification constraints, like the ones considered by Calvanese et al. [2014], would also be of practical importance. Given that the considered problems are intractable even for weak fragments of the core *DL-Lite* and very restricted forms of actions, it remains to explore how feasible these tasks are in practice, and whether there are meaningful restrictions that make them tractable. Another interesting addition to the basic actions that we are exploring, is to allow for conjunctive queries in the place of complex concepts and roles, similarly to the tasks considered by De Giacomo et al. [2012]. We note, however, that *tree-shaped* conjunctive queries (over unary and binary predicates) are readily expressible as DL concepts (even in very weak fragments of $\mathcal{ALCHOIQ}br$, such as $\mathcal{EL}$), and thus covered already by our results. Apart from other forms of basic actions, also a more expressive language for complex actions could be considered. Our results naturally extend, for example, to the parallel application of basic actions. But other natural extensions, such as 'while' loops in actions, would easily yield an undecidable formalism. Finally, towards the spirit of verification of data-centric systems, a natural next step would be to allow for more complex temporal properties in the goal formulas.

REFERENCES

Serge Abiteboul and Paris Kanellakis. 1989. Object Identity as a Query Language Primitive. In *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*. 159–173.

Serge Abiteboul and Victor Vianu. 1999. Regular Path Queries with Constraints. *J. of Computer and System Sciences* 58, 3 (1999), 428–452. DOI:http://dx.doi.org/10.1006/jcss.1999.1627

Shqiponja Ahmetaj, Diego Calvanese, Magdalena Ortiz, and Mantas Šimkus. 2014. Managing Change in Graph-structured Data Using Description Logics. In *Proc. of the 28th AAAI Conf. on Artificial Intelligence (AAAI)*. AAAI Press, 966–973.

Marcelo Arenas, Richard Hull, Wim Marten, Tova Milo, and Thomas Schwentick. 2016. Foundations of Data Management (Dagstuhl Perspectives Workshop 16151). *Dagstuhl Reports* 6, 4 (2016), 39–56. DOI:http://dx.doi.org/10.4230/DagRep.6.4.39

Alessandro Artale. 2006. Reasoning on Temporal Class Diagrams: Undecidability Results. *Ann. of Mathematics and Artificial Intelligence* 46, 3 (2006), 265–288.

Alessandro Artale, Diego Calvanese, Roman Kontchakov, Vladislav Ryzhikov, and Michael Zakharyaschev. 2007. Reasoning over Extended ER Models. In *Proc. of the 26th Int. Conf. on Conceptual Modeling (ER) (Lecture Notes in Computer Science)*, Vol. 4801. Springer, 277–292.

Alessandro Artale, Roman Kontchakov, Vladislav Ryzhikov, and Michael Zakharyaschev. 2011. Tailoring Temporal Description Logics for Reasoning over Temporal Conceptual Models. In *Proc. of the 8th Int. Symp. on Frontiers of Combining Systems (FroCoS)*. Springer, 1–11.

Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter F. Patel-Schneider (Eds.). 2003. *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press.

Franz Baader, Silvio Ghilardi, and Carsten Lutz. 2012. LTL over Description Logic Axioms. *ACM Trans. on Computational Logic* 13, 3 (2012), 21:1–21:32.

Franz Baader, Marcel Lippmann, and Hongkai Liu. 2010. Using Causal Relationships to Deal with the Ramification Problem in Action Formalisms Based on Description Logics. In *Proc. of the 17th Int. Conf. on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR)*. Lecture Notes in Computer Science, Vol. 6397. Springer, 82–96.

Franz Baader, Carsten Lutz, Maja Milicic, Ulrike Sattler, and Frank Wolter. 2005. Integrating Description Logics and Action Formalisms: First Results. In *Proc. of the 20th Nat. Conf. on Artificial Intelligence (AAAI)*. 572–577.

Franz Baader and Benjamin Zarrieß. 2013. Verification of Golog Programs over Description Logic Actions. In *Proc. of the 9th Int. Symp. on Frontiers of Combining Systems (FroCoS)*. Lecture Notes in Computer Science, Vol. 8152. Springer, 181–196.

Babak Bagheri Hariri, Diego Calvanese, Giuseppe De Giacomo, Alin Deutsch, and Marco Montali. 2013a. Verification of Relational Data-Centric Dynamic Systems with External Services. In *Proc. of the 32nd ACM SIGACT SIGMOD SIGAI Symp. on Principles of Database Systems (PODS)*. 163–174.

Babak Bagheri Hariri, Diego Calvanese, Marco Montali, Giuseppe De Giacomo, Riccardo De Masellis, and Paolo Felli. 2013b. Description Logic Knowledge and Action Bases. *J. of Artificial Intelligence Research* 46 (2013), 651–686.

Mira Balaban and Azzam Maraee. 2013. Finite Satisfiability of UML Class Diagrams with Constrained Class Hierarchy. *ACM Trans. on Software Engineering and Methodology* 22, 3 (2013), 24. DOI:http://dx.doi.org/10.1145/2491509.2491518

Daniela Berardi, Diego Calvanese, and Giuseppe De Giacomo. 2005. Reasoning on UML Class Diagrams. *Artificial Intelligence* 168, 1–2 (2005), 70–118.

Anthony J. Bonner and Michael Kifer. 1994. An Overview of Transaction Logic. *Theoretical Computer Science* 133, 2 (1994), 205–265.

Alex Borgida. 1996. On the Relative Expressiveness of Description Logics and Predicate Logics. *Artificial Intelligence* 82, 1–2 (1996), 353–367.

Tim Bray, Jean Paoli, and C. M. Sperberg-McQueen. 1998. *Extensible Markup Language (XML) 1.0*. W3C Recommendation. World Wide Web Consortium. Available at http://www.w3.org/TR/1998/REC-xml-19980210.

Jon Haël Brenas, Rachid Echahed, and Martin Strecker. 2014. A Hoare-Like Calculus Using the SROIQ$^\sigma$ Logic on Transformations of Graphs. In *Proc. of the 8th IFIP TC 1/WG 2.2 Int. Conf. on Theoretical Computer Science (TCS) (Lecture Notes in Computer Science)*, Vol. 8705. Springer, 164–178. DOI:http://dx.doi.org/10.1007/978-3-662-44602-7_14

Dan Brickley and R. V. Guha. 2004. *RDF Vocabulary Description Language 1.0: RDF Schema*. W3C Recommendation. World Wide Web Consortium. Available at http://www.w3.org/TR/rdf-schema/.

Tom Bylander. 1994. The Computational Complexity of Propositional STRIPS Planning. *Artificial Intelligence* 69 (1994), 165–204.

Diego Calvanese. 1996. Finite Model Reasoning in Description Logics. In *Proc. of the 5th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR)*. 292–303.

Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati. 2007. Tractable Reasoning and Efficient Query Answering in Description Logics: The *DL-Lite* Family. *J. of Automated Reasoning* 39, 3 (2007), 385–429.

Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Riccardo Rosati. 2011. Actions and Programs over Description Logic Knowledge Bases: A Functional Approach. In *Knowing, Reasoning, and Acting: Essays in Honour of Hector Levesque*, Gerhard Lakemeyer and Sheila A. McIlraith (Eds.). College Publications.

Diego Calvanese, Giuseppe De Giacomo, and Marco Montali. 2013. Foundations of Data-Aware Process Analysis: A Database Theory Perspective. In *Proc. of the 32nd ACM SIGACT SIGMOD SIGAI Symp. on Principles of Database Systems (PODS)*. 1–12.

Diego Calvanese, Thomas Eiter, and Magdalena Ortiz. 2009. Regular Path Queries in Expressive Description Logics with Nominals. In *Proc. of the 21st Int. Joint Conf. on Artificial Intelligence (IJCAI)*. 714–720.

Diego Calvanese, Wolfgang Fischl, Reinhard Pichler, Emanuel Sallinger, and Mantas Šimkus. 2014. Capturing Relational Schemas and Functional Dependencies in RDFS. In *Proc. of the 28th AAAI Conf. on Artificial Intelligence (AAAI)*. AAAI Press, 1003–1011.

Diego Calvanese, Maurizio Lenzerini, and Daniele Nardi. 1999. Unifying Class-Based Representation Formalisms. *J. of Artificial Intelligence Research* 11 (1999), 199–240.

Diego Calvanese, Magdalena Ortiz, and Mantas Simkus. 2016. Verification of Evolving Graph-structured Data under Expressive Path Constraints. In *Proc. of the 19th Int. Conf. on Database Theory (ICDT) (Leibniz International Proceedings in Informatics (LIPIcs))*, Vol. 48. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 15:1–15:19. DOI:http://dx.doi.org/10.4230/LIPIcs.ICDT.2016.15

Diego Calvanese, Magdalena Ortiz, and Mantas Šimkus. 2013. Evolving Graph Databases under Description Logic Constraints. In *Proc. of the 26th Int. Workshop on Description Logics (DL) (CEUR Workshop Proceedings)*, Vol. 1014. 120–131.

Mariano P. Consens and Alberto O. Mendelzon. 1990. GraphLog: a Visual Formalism for Real Life Recursion. In *Proc. of the 9th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS)*. 404–416.

Giuseppe De Giacomo, Riccardo De Masellis, and Riccardo Rosati. 2012. Verification of Conjunctive Artifact-Centric Services. *Int. J. of Cooperative Information Systems* 21, 2 (2012), 111–140.

Alin Deutsch, Richard Hull, and Victor Vianu. 2014. Automatic Verification of Database-Centric Systems. *SIGMOD Record* 43, 3 (2014), 5–17.

Ronald Fagin, Gabriel M. Kuper, Jeffrey D. Ullman, and Moshe Y. Vardi. 1986. Updating logical databases. In *Advances in Computing Research*. JAI Press, 1–18.

Yolanda Gil. 2005. Description Logics and Planning. *AI Magazine* 26, 2 (2005), 73–84.

Gösta Grahne and Alex Thomo. 2003. Query Containment and Rewriting Using Views for Regular Path Queries Under Constraints. In *Proc. of the 22nd ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS)*. 111–122. DOI:http://dx.doi.org/10.1145/773153.773165

Richard B. Hull and Roger King. 1987. Semantic Database Modelling: Survey, Applications and Research Issues. *Comput. Surveys* 19, 3 (Sept. 1987), 201–260.

Ullrich Hustadt, Renate A. Schmidt, and Lilie Georgieva. 2004. A Survey of Decidable First-Order Fragments and Description Logics. *J. on Relational Methods in Computer Science* 1 (2004), 251–276.

Shachar Itzhaky, Tomer Kotek, Noam Rinetzky, Mooly Sagiv, Orr Tamir, Helmut Veith, and Florian Zuleger. 2016. *On the Automated Verification of Web Applications With Embedded SQL*. CoRR Technical Report arXiv:1610.02101. arXiv.org e-Print archive. http://arxiv.org/abs/1610.02101 Available at http://arxiv.org/abs/1610.02101.

Hirofumi Katsuno and Alberto O. Mendelzon. 1991. On the Difference between Updating a Knowledge Base and Revising It. In *Proc. of the 2nd Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR)*. 387–394.

Yevgeny Kazakov. 2004. A Polynomial Translation from the Two-Variable Guarded Fragment with Number Restrictions to the Guarded Fragment. In *Proc. of the 9th Eur. Conf. on Logics in Artificial Intelligence (JELIA) (Lecture Notes in Computer Science)*, Vol. 3229. Springer, 372–384.

Evgeny Kharlamov, Dmitriy Zheleznyakov, and Diego Calvanese. 2013. Capturing Model-Based Ontology Evolution at the Instance Level: The Case of *DL-Lite*. *J. of Computer and System Sciences* 79, 6 (2013), 835–872.

Robert A. Kowalski, Fariba Sadri, and Paul Soper. 1987. Integrity Checking in Deductive Databases. In *Proc. of the 13th Int. Conf. on Very Large Data Bases (VLDB)*. 61–69.

Maurizio Lenzerini. 2011. Ontology-based Data Management. In *Proc. of the 20th Int. Conf. on Information and Knowledge Management (CIKM)*. 5–6.

Hector J. Levesque. 1984. Foundations of a Functional Approach to Knowledge Representation. *Artificial Intelligence* 23 (1984), 155–212.

H. J. Levesque, R. Reiter, Y. Lesperance, F. Lin, and R. Scherl. 1997. GOLOG: A Logic Programming Language for Dynamic Domains. *J. of Logic Programming* 31 (1997), 59–84.

Hongkai Liu, Carsten Lutz, Maja Milicic, and Frank Wolter. 2006. Reasoning About Actions Using Description Logics with General TBoxes. In *Proc. of the 10th Eur. Conf. on Logics in Artificial Intelligence (JELIA) (Lecture Notes in Computer Science)*, Vol. 4160. Springer, 266–279. DOI: http://dx.doi.org/10.1007/11853886_23

Hongkai Liu, Carsten Lutz, Maja Milicic, and Frank Wolter. 2011. Foundations of Instance Level Updates in Expressive Description Logics. *Artificial Intelligence* 175, 18 (2011), 2170–2197.

Carsten Lutz, Ulrike Sattler, and Lidia Tendera. 2005. The Complexity of Finite Model Reasoning in Description Logics. *Information and Computation* 199, 1–2 (2005), 132–171.

Carsten Lutz, Ulrike Sattler, and Frank Wolter. 2001. Description Logics and the Two-Variable Fragment. In *Proc. of the 14th Int. Workshop on Description Logics (DL) (CEUR Electronic Workshop Proceedings,* http://ceur-ws.org/*)*, Vol. 49. 66–75. http://CEUR-WS.org/Vol-49/LutzSattlerWolter-66start.ps

Maja Milicic. 2008. *Action, Time and Space in Description Logics*. Ph.D. Dissertation. TU Dresden.

Michael Mortimer. 1975. On Languages with Two Variables. *Mathematical Logic Quarterly (formerly: Zeitschrift für Mathematische Logik und Grundlagen der Mathematik)* 21, 1 (1975), 135–140.

Pablo Muñoz, Nils Vortmeier, and Thomas Zeume. 2016. Dynamic Graph Queries. In *Proc. of the 19th Int. Conf. on Database Theory (ICDT) (Leibniz International Proceedings in Informatics (LIPIcs))*, Vol. 48. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 14:1–14:18.

Ian Pratt-Hartmann. 2005. Complexity of the Two-Variable Fragment with Counting Quantifiers. *J. of Logic, Language and Information* 14, 3 (2005), 369–395.

Sherif Sakr and Eric Pardede (Eds.). 2011. *Graph Data Management: Techniques and Applications*. IGI Global.

Klaus Schild. 1991. A Correspondence Theory for Terminological Logics: Preliminary Report. In *Proc. of the 12th Int. Joint Conf. on Artificial Intelligence (IJCAI)*. Morgan Kaufmann, 466–471.

Renate A. Schmidt and Dmitry Tishkovsky. 2014. Using Tableau to Decide Description Logics With Full Role Negation and Identity. *ACM Trans. on Computational Logic* 15, 1 (2014), 7:1–7:31.

Tim Sheard and David Stemple. 1989. Automatic Verification of Database Transaction Safety. *ACM Trans. on Database Systems* 14, 3 (1989), 322–368.

David Spelt and Herman Balsters. 1998. Automatic Verification of Transactions on an Object-Oriented Database. In *Proc. of the 6th Int. Workshop on Database Programming Languages (DBPL) (Lecture Notes in Computer Science)*, Vol. 1369. Springer, 396–412.

Balder ten Cate and Massimo Franceschet. 2005. Guarded Fragments with Constants. *J. of Logic, Language and Information* 14, 3 (2005), 281–288.

Stephan Tobies. 2000. The Complexity of Reasoning with Cardinality Restrictions and Nominals in Expressive Description Logics. *J. of Artificial Intelligence Research* 12 (2000), 199–217.

Moshe Y. Vardi. 1995. On the Complexity of Bounded-variable Queries (Extended Abstract). In *Proc. of the 14th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS)*. 266–276. DOI: http://dx.doi.org/10.1145/212433.212474

Frank Wolter and Michael Zakharyaschev. 1999. Temporalizing Description Logic. In *Frontiers of Combining Systems*, D. Gabbay and M. de Rijke (Eds.). Studies Press/Wiley, 379–402.

A. APPENDIX

We first show that KBs in $\mathcal{ALCHOIQ}br$ can be seen as KBs in a standard DL and that many of the added features are only syntactic sugar. Let $\mathcal{ALCHOIQ}^{\cup,\setminus}$ be the extension of $\mathcal{ALCHOIQ}$ with the role constructors of union and difference. That is, $\mathcal{ALCHOIQ}^{\cup,\setminus}$ is defined as in the left column of Table I, except that roles r are defined as follows:

$$r \longrightarrow p \mid r^- \mid r_1 \cup r_2 \mid r_1 \setminus r_2$$

PROPOSITION A.1. *For every $\mathcal{ALCHOIQ}br$ KB $\mathcal{K}$, there is an $\mathcal{ALCHOIQ}^{\cup,\setminus}$ KB $\mathcal{K}'$ such that the following hold:*

(1) every model of $\mathcal{K}'$ is a model of $\mathcal{K}$, and

(2) every model of $\mathcal{K}$ can be transformed into a model of $\mathcal{K}'$, by possibly modifying the interpretation of symbols in $\mathcal{K}'$ not mentioned in $\mathcal{K}$.

PROOF. Let $\mathcal{K}$ be an $\mathcal{ALCHOIQ}br$ KB. It suffices to prove that, except for role union and role difference, every added construct in $\mathcal{ALCHOIQ}br$ (namely singleton roles, role intersection, role restriction, disjunction and negation of KBs), can be simulated in an $\mathcal{ALCHOIQ}^{\cup,\setminus}$ KB $\mathcal{K}'$. We show this as follows:

— Singleton role $\{(o_1, o_2)\}$: this constructor can be simulated in $\mathcal{K}'$ by introducing a fresh role name $p_{o_1o_2}$, replacing every occurrence of $\{(o_1, o_2)\}$ in $\mathcal{K}$ with $p_{o_1o_2}$, and adding to $\mathcal{K}'$ the following three inclusions:

$$\{o_1\} \sqsubseteq \exists p_{o_1o_2}.\top \qquad \exists p_{o_1o_2}.\top \sqsubseteq \{o_1\} \qquad \exists p^-_{o_1o_2}.\top \sqsubseteq \{o_2\}$$

One can verify that every model $\mathcal{I}$ of $\mathcal{K}$ can be transformed into a model of $\mathcal{K}'$ by additionally interpreting the role name $p_{o_1o_2}$ as $p^{\mathcal{I}}_{o_1,o_2} = \{(o_1^{\mathcal{I}}, o_2^{\mathcal{I}})\} = \{(o_1, o_2)\}^{\mathcal{I}}$. Also, every model of $\mathcal{K}'$, when restricted to the symbols in $\mathcal{K}$, is also a model of $\mathcal{K}$.

— Role intersection: $r_1 \cap r_2$ can be equivalently replaced by $r_1 \setminus (r_1 \setminus r_2)$.

— Domain and range restrictions $r{\restriction_C}$ and $r{\downharpoonright_C}$: first we note that, by using role inverse, each of the two constructs can be expressed in terms of the other one. Specifically, $r{\restriction_C}$ can be equivalently replaced with $(r^-{\downharpoonright_C})^-$. Hence, we can assume that $\mathcal{K}$ contains no occurrences of $r{\restriction_C}$. To simulate $r{\downharpoonright_C}$ in $\mathcal{K}'$, we introduce two fresh role names r_C and $r_{\bar{C}}$, substitute every occurrence of $r{\downharpoonright_C}$ in $\mathcal{K}$ with r_C, and add to $\mathcal{K}'$ the following inclusions:

$$r \sqsubseteq r_C \cup r_{\bar{C}} \qquad r_C \cup r_{\bar{C}} \sqsubseteq r \qquad \exists r_C^-.\top \sqsubseteq C \qquad \exists r_{\bar{C}}^-.\top \sqsubseteq \neg C$$

One can verify that every model $\mathcal{I}$ of $\mathcal{K}$ can be transformed into a model of $\mathcal{K}'$ by additionally interpreting the role name r_C as ${r_C}^{\mathcal{I}} = r{\downharpoonright_C}^{\mathcal{I}}$ and the role name and $r_{\bar{C}}$ as ${r_{\bar{C}}}^{\mathcal{I}} = r^{\mathcal{I}} \setminus r{\downharpoonright_C}^{\mathcal{I}}$. Also, every model of $\mathcal{K}'$, when restricted to the symbols in $\mathcal{K}$, is also a model of $\mathcal{K}$.

— Boolean KBs: we assume without loss of generality that only concept and role inclusions occur in $\mathcal{K}$. Indeed, every assertion $o : C$ can be equivalently expressed as the inclusion $\{o\} \sqsubseteq C$, and every assertion $(o_1, o_2) : r$ as the inclusion $\{o_1\} \sqsubseteq \exists r.\{o_2\}$. Let $\alpha_1, \ldots, \alpha_n$ be all the concept and role inclusions that occur in $\mathcal{K}$ (note that the same α_i can occur in $\mathcal{K}$ several times, possibly negated). For each α_i, let A_i be a new concept name. We define $\mathsf{Conc}(\mathcal{K})$ as the $\mathcal{ALCHOIQ}$ concept obtained from $\mathcal{K}$ by replacing each α_i by A_i, each $\vee$ by $\sqcup$, each $\wedge$ by $\sqcap$, and each $\dot{\neg}$ by $\neg$. The $\mathcal{ALCHOIQ}^{\cup,\setminus}$ KB $\mathcal{K}'$ is then defined as the conjunction of the following inclusions, where o is a fresh individual name and s is a fresh role name:

$$\begin{array}{rcll} \{o\} & \sqsubseteq & \mathsf{Conc}(\mathcal{K}) & \\ \top & \sqsubseteq & \exists s^{-}.\{o\} & \\ \{o\} \sqcap A_i & \sqsubseteq & \forall s.(\neg C_1 \sqcup C_2), & \text{for all } \alpha_i = C_1 \sqsubseteq C_2 \\ \{o\} \sqcap \neg A_i & \sqsubseteq & \exists s.(C_1 \sqcap \neg C_2), & \text{for all } \alpha_i = C_1 \sqsubseteq C_2 \\ \{o\} \sqcap A_i & \sqsubseteq & \forall s.A_i, & \text{for all } \alpha_i = r_1 \sqsubseteq r_2 \\ r_1|_{A_i} & \sqsubseteq & r_2, & \text{for all } \alpha_i = r_1 \sqsubseteq r_2 \\ \{o\} \sqcap \neg A_i & \sqsubseteq & \exists s.(\exists (r_1 \setminus r_2).\top), & \text{for all } \alpha_i = r_1 \sqsubseteq r_2. \end{array}$$

Note that the fresh role s connects o with every element of the domain. The underlying intuition of the above encoding is the following: whenever an inclusion α_i in $\mathcal{K}$ is set to true then o satisfies the corresponding A_i in $\mathcal{K}'$, and the role s ensures that every element of the domain satisfies the inclusion α_i. Conversely, if the inclusion α_i in $\mathcal{K}$ is set to false, i.e., $\dot{\neg}\alpha_i$ is set to true, then o does not belong to the corresponding A_i in $\mathcal{K}'$ and in this case the s role makes sure that there exists an element of the domain that does not satisfy α_i.

More formally, let $\mathcal{I}$ be a model of $\mathcal{K}$. We construct a model $\mathcal{I}'$ of $\mathcal{K}'$ as follows:

— for all $\alpha_i = C_1 \sqsubseteq C_2$ set:

$$\begin{array}{ll} A_i^{\mathcal{I}'} = \{o^{\mathcal{I}}\}, & \text{if } \mathcal{I} \models \alpha_i, \text{ and} \\ A_i^{\mathcal{I}'} = \emptyset, & \text{if } \mathcal{I} \not\models \alpha_i \end{array}$$

— for all $\alpha_i = r_1 \sqsubseteq r_2$ set:

$$\begin{array}{ll} A_i^{\mathcal{I}'} = \Delta^{\mathcal{I}}, & \text{if } \mathcal{I} \models \alpha_i, \text{ and} \\ A_i^{\mathcal{I}'} = \emptyset, & \text{if } \mathcal{I} \not\models \alpha_i \end{array}$$

— $s^{\mathcal{I}'} = \{o^{\mathcal{I}}\} \times \Delta^{\mathcal{I}}$

— $\mathcal{I}'$ coincides with $\mathcal{I}$ in the interpretation of all other individual names, concept names, and role names.

One can easily check that $\mathcal{I}'$ is indeed a model of $\mathcal{K}'$. Moreover, by construction we have that every model of $\mathcal{K}'$ is also a model of $\mathcal{K}$. □

A.1. $\mathcal{ALCHOIQ}br$ and $\mathcal{C}^2$

The so-called *standard translation* from DLs to $\mathcal{C}^2$ can be found in several works, e.g., [Borgida 1996; Schmidt and Tishkovsky 2014; Hustadt et al. 2004], and is summarized in Table III. It uses a unary predicate $\hat{A}$ for each concept name A, a binary predicate $\hat{p}$ for each role name p, and a constant $\hat{o}$ for each individual o. Complex concepts are translated into formulas with one free variable using two translation functions: ST_x translates a concept C into a formula with x as free variable, and ST_y into a formula with y as free variable. Note that, since the translation is to $\mathcal{C}^2$, the only variables it uses are x and y, which may be reused as quantified variables inside the translated formulas $ST_x(C)$ and $ST_y(C)$. Complex roles are translated into formulae with two free variables. Again, we have two analogous translation functions: ST_{xy} produces a $\mathcal{C}^2$ formula with the pair (x, y) as free variables, while ST_{yx} produces a $\mathcal{C}^2$ formula with the pair (y, x). Each assertion, inclusion, and knowledge base α is translated into a $\mathcal{C}^2$ sentence $ST(\alpha)$. If α is an assertion, then $ST(\alpha)$ is obtained by substituting free variables with constants. For α an inclusion, we quantify universally over the free variables. Finally, KBs are translated into the Boolean combinations of the sentences translating their inclusions and assertions.

As mentioned, all the upper bounds in this paper would still hold if we use as our constraint language a more expressive DL that is still captured by $\mathcal{C}^2$ (or even $\mathcal{C}^2$ itself). This means that we can add to $\mathcal{ALCHOIQ}br$ (any combination of) the constructors in

Table III. The standard translation from $\mathcal{ALCHOIQ}br$ to $\mathcal{C}^2$.

Translating concepts	
$ST_x(\top) = (x = x)$	$ST_y(\top) = (y = y)$
$ST_x(\bot) = \neg(x = x)$	$ST_y(\bot) = \neg(y = y)$
$ST_x(A) = \hat{A}(x)$	$ST_y(A) = \hat{A}(y)$
$ST_x(\{o\}) = (x = \hat{o})$	$ST_y(\{o\}) = (y = \hat{o})$
$ST_x(\neg C) = \neg ST_x(C)$	$ST_y(\neg C) = \neg ST_y(C)$
$ST_x(C_1 \sqcap C_2) = ST_x(C_1) \land ST_x(C_2)$	$ST_y(C_1 \sqcap C_2) = ST_y(C_1) \land ST_y(C_2)$
$ST_x(C_1 \sqcup C_2) = ST_x(C_1) \lor ST_x(C_2)$	$ST_y(C_1 \sqcup C_2) = ST_y(C_1) \lor ST_y(C_2)$
$ST_x(\exists r.C) = \exists y.(ST_{xy}(r) \land ST_y(C))$	$ST_y(\exists r.C) = \exists x.(ST_{yx}(r) \land ST_x(C))$
$ST_x(\forall r.C) = \forall y.(ST_{xy}(r) \rightarrow ST_y(C))$	$ST_y(\forall r.C) = \forall x.(ST_{yx}(r) \rightarrow ST_x(C))$
$ST_x(\leqslant n\, r.C) = \exists^{\leq n} y.(ST_{xy}(r) \land ST_y(C))$	$ST_y(\leqslant n\, r.C) = \exists^{\leq n} x.(ST_{yx}(r) \land ST_x(C))$
$ST_x(\geqslant n\, r.C) = \exists^{\geq n} y.(ST_{xy}(r) \land ST_y(C))$	$ST_y(\geqslant n\, r.C) = \exists^{\geq n} x.(ST_{yx}(r) \land ST_x(C))$
Translating roles	
$ST_{xy}(p) = \hat{p}(x, y)$	$ST_{yx}(p) = \hat{p}(y, x)$
$ST_{xy}(\{o, o'\}) = (x = \hat{o}) \land (y = \hat{o}')$	$ST_{yx}(\{o, o'\}) = (y = \hat{o}) \land (x = \hat{o}')$
$ST_{xy}(r^-) = ST_{yx}(r)$	$ST_{yx}(r^-) = ST_{xy}(r)$
$ST_{xy}(r_1 \cup r_2) = ST_{xy}(r_1) \lor ST_{xy}(r_2)$	$ST_{yx}(r_1 \cup r_2) = ST_{yx}(r_1) \lor ST_{yx}(r_2)$
$ST_{xy}(r_1 \setminus r_2) = ST_{xy}(r_1) \land \neg ST_{xy}(r_2)$	$ST_{yx}(r_1 \setminus r_2) = ST_{yx}(r_1) \land \neg ST_{yx}(r_2)$
$ST_{xy}(r_1 \cap r_2) = ST_{xy}(r_1) \land ST_{xy}(r_2)$	$ST_{yx}(r_1 \cap r_2) = ST_{yx}(r_1) \land ST_{yx}(r_2)$
$ST_{xy}(r\restriction_C) = ST_{xy}(r) \land ST_x(C)$	$ST_{yx}(r\restriction_C) = ST_{yx}(r) \land ST_y(C)$
$ST_{xy}(r\downharporight_C) = ST_{xy}(r) \land ST_y(C)$	$ST_{yx}(r\downharpoonright_C) = ST_{yx}(r) \land ST_x(C)$
Translating assertions, inclusions, and KBs	
$ST(o : C) = ST_x(C)[x \mapsto \hat{o}]$	$ST(C_1 \sqsubseteq C_2) = \forall x.(ST_x(C_1) \rightarrow ST_x(C_2))$
$ST((o_1, o_2) : r) = ST_{xy}(r)[x \mapsto \hat{o_1}, y \mapsto \hat{o_2}]$	$ST(r_1 \sqsubseteq r_2) = \forall x \forall y.(ST_{xy}(r_1) \rightarrow ST_{xy}(r_2))$
$ST(\mathcal{K}_1 \land \mathcal{K}_2) = ST(\mathcal{K}_1) \land ST(\mathcal{K}_2)$	$ST(\mathcal{K}_1 \lor \mathcal{K}_2) = ST(\mathcal{K}_1) \lor ST(\mathcal{K}_2)$
$ST(\dot\neg \mathcal{K}) = \neg ST(\mathcal{K})$	

C, C_1, C_2 are ordinary concepts, r, r_1, r_2 ordinary roles, $\mathcal{K}, \mathcal{K}_1, \mathcal{K}_2$ KBs, $A \in \mathsf{N_C}$, $p \in \mathsf{N_R}$, $\{o, o_1, o_2\} \subseteq \mathsf{N_I}$.

the upper part of Table IV, since they are all expressible in $\mathcal{C}^2$. In particular, our results hold for $\mathcal{ALBOQ}^{id}$, which adds number restrictions to $\mathcal{ALBO}^{id}$. $\mathcal{ALBO}^{id}$ is known to have the same expressiveness as the two-variable fragment of first-order predicate logic with equality. The additional concept and role constructors supported in these logics are given in Table IV. For a more detailed discussion of the relationship between the two-variable fragment and these expressive DLs, as well as the interdefinability of the different constructors, we refer the reader to the literature, e.g., [Borgida 1996; Lutz et al. 2001; Hustadt et al. 2004; Schmidt and Tishkovsky 2014] and references therein.

Table IV. Additional constructors in $\mathcal{ALBOQ}^{id}$

	Syntax	Semantics	Translation to $\mathcal{C}^2$
role negation	$\neg r$	$(\neg r)^{\mathcal{I}} = (\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}) \setminus r^{\mathcal{I}}$	$ST_{xy}(\neg r) = \neg ST_{xy}(r)$
identity role	id	$(id)^{\mathcal{I}} = \{(e,e) \mid e \in \Delta^{\mathcal{I}}\}$	$ST_{xy}(id(C)) = ST_x(C) \wedge x = y$

Definable constructors

	Syntax	Definition	
Role constructors			
Top role	$\triangledown$	$\triangledown \equiv p \cup \neg p$	for some role name p
Bottom role	$\triangle$	$\triangle \equiv \neg\triangledown$	
Left cylindrification	D^c	$D^c \equiv \triangledown\upharpoonleft_D$	
Right cylindrification	${}^cD$	${}^cD \equiv \triangledown\downharpoonleft_D$	
Cross product	$C \times D$	$C \times D \equiv C^c \cap {}^cD$	
Test	$C?$	$C? \equiv id\upharpoonleft_D$	
Concept constructors			
Universal modality	$\Box C$	$\Box C \equiv \forall\triangledown.C$	
Sufficiency (or window)	$\overline{\forall} R.C$	$\overline{\forall} R.C \equiv \neg\exists\neg R.C$	