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 article, we consider the setting of graph-structured data 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 preceding 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, and Mantas Šimkus. 2017. Managing Change in Graph-Structured Data Using Description Logics. ACM Trans. Comput. Logic 18, 4, Article 27 (November 2017), 35 pages.
https://doi.org/10.1145/3143803

This research was partially supported by FWF projects T515, P25518, P30360, and W1255; WWTF project ICT12-015; EU IP Project Optique FP7-318338; and the Wolfgang Pauli Institute.

Authors’ addresses: S. Ahmetaj, M. Ortiz, and M. Šimkus, Institut für Informationssysteme 184/3, Technische Universität Wien, Favoritenstraße 9-11, A-1040 Wien, Austria; emails: ahmetaj@dbai.tuwien.ac.at, ortiz@kr.tuwien.ac.at, simkus@dbai.tuwien.ac.at; D. Calvanese, Faculty of Computer Science, Free University of Bozen-Bolzano, Piazza Domenicani 3, I-39100 Bolzano, Italy; email: calvanese@inf.unibz.it.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

© 2017 ACM 1529-3785/2017/11-ART27 $15.00
https://doi.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. Technologies for data storage and access have not ceased to evolve and mature on the solid theoretical basis of the principles of data management; however, these technologies often disregard the fundamental fact that data is intrinsically dynamic and evolves constantly, mainly as the result 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 faced by the Foundations of Data Management community in the coming 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 article, we also aim at developing tools that allow one to reason about evolving data, particularly 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 Paredes 2011), which has roots in work done in the early 1990s (e.g., see 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, such as testing whether for every possible data instance some given property is guaranteed to hold after applying some given updates.

The languages that 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 (also known as concepts) and binary relations (also known as roles), and hence the first-order interpretations of a DL 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 Marae 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 article, 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, for example, (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 article, 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 preconditions that capture the effect of action sequences on data states. Although 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 \( ALCHOIQbr \). This DL is an extension of the well-known \( ALCHOIQ \) and, as we will see, is closely related to the two-variable fragment of first-order logic with counting quantifiers.

— 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 \( ALCHOIQbr \) 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 preconditions 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 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 article 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 ALCHOIQbr 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 article 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 preconditions 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 ALCHOIQ as a Constraint Language for GSD

To express constraints on GSD, we choose the expressive DL ALCHOIQ. It is an extension of the basic DL 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 in addition to
In the following, we use $\text{br}$ assertions as $N$. In $\text{br}$ assertions, $\top$, $\bot$, and $\{ \}$ are individual names. $\exists r.C$, $\forall r.C$, $\leq n r.C$, and $\geq n r.C$ are roles, whereas the latter assert the membership of specific (pairs of) individuals in concepts.

Syntax and Semantics. We recall the standard definition of $\text{ALCHOIQ}$, summarized in the left column of Table 1. Countably infinite sets $N_R$ of role names, $N_C$ of concept names, and $N_I$ 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, whereas the latter assert the membership of specific (pairs of) individuals in concepts and roles. A KB is a conjunction of inclusions and assertions. In the following, we use $C_1 \sqsubseteq C_2$ as a shortcut for the conjunction $(C_1 \sqsubseteq C_2) \land (C_2 \sqsubseteq C_1)$ of two concept inclusions and 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) nonempty 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

---

Table 1. Syntax of $\text{ALCHOIQ}$ and $\text{ALCHOIQbr}$

| $\text{ALCHOIQ}$ | $\text{ALCHOIQbr}$ |
|-------------------|---------------------|
| Roles $r(i)$ and concepts $C(i)$: | |
| $r \rightarrow p \mid p^{-}$ | $r \rightarrow p \mid p^{-} \mid \{t_1, t_2\}$ |
| $C \rightarrow A \mid T \mid \perp \mid \{o\}$ | $C \rightarrow A \mid T \mid \perp \mid \{t\}$ |
| $C_1 \sqcap C_2 \mid C_1 \sqcup C_2 \mid \neg C$ | $C_1 \sqcap C_2 \mid C_1 \sqcup C_2 \mid \neg C$ |
| $\exists r.C \mid \forall r.C \mid n r.C \mid \geq n r.C$ | $\exists r.C \mid \forall r.C \mid n r.C \mid \geq n r.C$ |

where $p \in N_R$, $A \in N_C$, $o \in N_I$, $(t, t_1, t_2) \in N_I \cup N_V$, and $n \geq 0$ is an integer.

Assertions and inclusions:

- $C_1 \sqsubseteq C_2$ for concept inclusions
- $r_1 \sqsubseteq r_2$ for role inclusions
- $o : C$ for concept assertions
- $(o_1, o_2) : r$ for role assertions

where $C_1, C_2$ are concepts, $r_1, r_2$ are roles, $(o, o_1, o_2) \subseteq N_I$, and $(t, t_1, t_2) \subseteq N_I \cup N_V$.

Concepts, roles, inclusions, and assertions with no variables are called ordinary.

KBs: Formula: (KBs are formulas with no variables)

- $\mathcal{K} \rightarrow \alpha \mid \mathcal{K}_1 \land \mathcal{K}_2$
- $\mathcal{K} \rightarrow \alpha \mid \mathcal{K}_1 \land \mathcal{K}_2 \mid \neg \mathcal{K}_1 \lor \mathcal{K}_2$, where $\alpha$ denotes a (concept or role) inclusion or a (concept or role) assertion.
Table 2. Semantics of $\text{ALCHOIQ}$ and $\text{ALCHOIQbr}$

| Concept constructors in $\text{ALCHOIQ}$: |
|------------------------------------------|
| $\top^I = \Delta^I$                     |
| $\bot^I = \emptyset$                    |
| $(\neg C_1)^I = \Delta^I \setminus C_1^I$ | |
| $(C_1 \cap C_2)^I = C_1^I \cap C_2^I$   |
| $(C_1 \cup C_2)^I = C_1^I \cup C_2^I$   |
| $(\exists r.C)^I = \{ e_1 | \text{for some } e_2 \in \Delta^I, (e_1, e_2) \in r^I \text{ and } e_2 \in C^I \}$ |
| $(\forall r.C)^I = \{ e_1 | \text{for all } e_2 \in \Delta^I, (e_1, e_2) \in r^I \text{ implies } e_2 \in C^I \}$ |
| $(\leq n r.C)^I = \{ e_1 | |\{ e_2 | (e_1, e_2) \in r^I \text{ and } e_2 \in C^I \}| \leq n \}$ |
| $(\geq n r.C)^I = \{ e_1 | |\{ e_2 | (e_1, e_2) \in r^I \text{ and } e_2 \in C^I \}| \geq n \}$ |

| Role constructors in $\text{ALCHOIQ}$: |
|----------------------------------------|
| $(r)^{-1} = \{ (e_1, e_2) | (e_2, e_1) \in r^I \}$ |

| Additional role constructors in $\text{ALCHOIQbr}$: |
|---------------------------------------------------|
| $\{(o_1, o_2)\}^I = \{ (o_1^I, o_2^I) \}$ |
| $(r_1 \cup r_2)^I = r_1^I \cup r_2^I$ |
| $(r_1 \setminus r_2)^I = r_1^I \setminus r_2^I$ |
| $(r_1 \cap r_2)^I = r_1^I \cap r_2^I$ |
| $(r|_{\alpha})^I = \{ (e_1, e_2) \in r^I | e_1 \in C^I \}$ |
| $(r\alpha)^I = \{ (e_1, e_2) \in r^I | e_2 \in C^I \}$ |

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

DL interpretations. We define next the semantics of $\text{ALCHOIQ}$ using the standard notion of an interpretation, which in what follows we call a (GSD) instance.

**Definition 2.1.** An instance is a pair $I = \langle \Delta^I, \cdot^I \rangle$ where $\Delta^I \neq \emptyset$ is the domain, $A^I \subseteq \Delta^I$ for each $A \in N_C$, $p^I \subseteq \Delta^I \times \Delta^I$ for each $p \in N_R$, and $o^I \subseteq \Delta^I$ for each $o \in N_t$. The function $\cdot^I$ is extended to all $\text{ALCHOIQ}$ concepts and roles as usual (Table 2).

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

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

**Example 2.2.** Suppose that some institution wants to ensure the following constraints on its database: (1) projects and departments are disjoint, and projects are active projects or finished projects; (2) hasMember, hasHead, and isLeader 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, and similarly the leader of a project also works for that project; (4) the domain of worksFor is 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, and 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 following KB:

\[(\Prj \sqcap \Dept \subseteq \bot) \land (\Prj \equiv \ActivePrj \sqcup \FinishedPrj) \land \]
\[(\text{hasMember} \equiv \text{isMemberOf}^-) \land (\text{hasHead} \equiv \text{isHeadOf}^-) \land (\text{hasLeader} \equiv \text{isLeaderOf}^-) \land \]
\n\[(\text{hasHead} \subseteq \text{hasMember}) \land (\text{isLeaderOf} \subseteq \text{worksFor}) \land \]
\n\[(\exists \text{worksFor}. \top \subseteq \Empl) \land (\top \subseteq \forall \text{worksFor}. \Prj) \land \]
\n\[(\exists \text{hasMember}. \top \subseteq \Dept) \land (\top \subseteq \forall \text{hasMember}. \Empl) \land \]
\n\[(\Prj\Empl \equiv \exists \text{worksFor}. \ActivePrj) \land (\Empl \subseteq \Prj\Empl \sqcup \exists \text{isMemberOf}. \Dept) \land \]
\n\[(\Dept \subseteq \exists \text{hasHead}. \bot) \land (\Dept \subseteq 1 \text{hasHead}. \bot) \land \]
\n\[(\ActivePrj \subseteq \exists \text{hasLeader}. (\exists \text{isMemberOf}. \Dept) \) \]

\[2.2\text{ Extending }\text{ALCHOIQ}\text{ for Reasoning About Evolving GSD}\]
To make \text{ALCHOIQ} better suited 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 \textit{domain restriction} and \textit{range restriction}, respectively. We also allow for Boolean KBs, and we allow for variables to occur in the place of individuals.

\textit{Syntax}. Additionally to countably infinite sets of \(\mathbb{N}_R\), \(\mathbb{N}_C\), and \(\mathbb{N}_I\), we consider a countably infinite set \(\mathbb{N}_V\) of \textit{variables}; all of these sets are pairwise disjoint. The syntax of \text{ALCHOIQ}_{br} is summarized in the right column of Table 1.

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

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

\textit{Definition 2.3}. Consider an instance \(I\). The function \(\cdot^I\) is extended to the additional constructs in \text{ALCHOIQ}_{br} as given in Table 2. The semantics of \text{ALCHOIQ} presented in Definition 2.1 carries over to \text{ALCHOIQ}_{br} ordinary inclusions and assertions and is extended to \text{ALCHOIQ}_{br} KBs as follows:

\[
I \models \mathcal{K}_1 \land \mathcal{K}_2 \quad \text{if} \quad I \models \mathcal{K}_1 \text{ and } I \models \mathcal{K}_2;
\]
\[
I \models \mathcal{K}_1 \lor \mathcal{K}_2 \quad \text{if} \quad I \models \mathcal{K}_1 \text{ or } I \models \mathcal{K}_2;
\]
\[
I \models \neg \mathcal{K} \quad \text{if} \quad I \not\models \mathcal{K}.
\]
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 (respectively, unsatisfiability) problem is to decide, given a KB $\mathcal{K}$, if there exists (respectively, does not exist) a model $I$ of $\mathcal{K}$ with $\Delta^I$ finite.

We note that $\text{ALCHOIQ}_{br}$ does not enjoy the finite model property. In other words, 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 (e.g., see Calvanese (1996) and 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 $\text{ALCHOIQ}_{br}$. Please keep in mind that modelhood is only defined for KBs (i.e., with no variables); and that in the finite satisfiability problem the input is such a KB.

$\text{ALCHOIQ}_{br}$ and Other DLs. Although some features of $\text{ALCHOIQ}_{br}$ may not seem common, it can be seen as a rather standard DL with some “syntactic sugar.” In fact, KBs in $\text{ALCHOIQ}_{br}$ extend KBs in $\text{ALCHOIQ}$ 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 $\text{ALCHOIQ}_{br}$ and $\mathcal{C}^2$

We note that we have defined $\text{ALCHOIQ}_{br}$ 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 $\text{ALBOQ}^\text{id}$, which adds number restrictions to the DL $\text{ALBO}^\text{id}$, which is known to have the same expressiveness as the two-variable fragment of first-order logic with equality.

The crucial property of $\text{ALCHOIQ}_{br}$ is that there exists a polynomial translation from KBs in $\text{ALCHOIQ}_{br}$ to sentences in $\mathcal{C}^2$ that preserves (finite) models. This is what we use to establish Theorem 2.6, 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 $\text{ALCHOIQ}_{br}$ with the combination of the constructs expressible in $\text{ALBOQ}^\text{id}$ that we show in the Appendix (see Table 4).

Remark 2.5. All 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 $\text{ALCHOIQ}_{br}$ KBs, concepts, and roles are respectively replaced by KBs, concepts, and roles in $\text{ALBOQ}^\text{id}$
— When $\text{ALCHOIQ}_{br}$ KBs are replaced by $\mathcal{C}^2$ sentences, whereas concepts and roles are respectively replaced by $\mathcal{C}^2$ formulas with one and two free variables.

2.4 Complexity of Reasoning in $\text{ALCHOIQ}_{br}$

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

Theorem 2.6. Finite satisfiability of $\text{ALCHOIQ}_{br}$ KBs is $\text{NExpTime}$-complete.
Managing Change in Graph-Structured Data Using Description Logics

Proof (Sketch). A \( \text{NExpTime} \) lower bound for finite satisfiability in \( \mathcal{ALCHOIQ}_{br} \) follows from the work of Tobies (2000), who showed \( \text{NExpTime} \)-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 \( \text{NExpTime} \) upper bound for finite satisfiability of \( \mathcal{C}^2 \) formulas established by Pratt-Hartmann (2005). □

2.5 \( \mathcal{ALCHOIQ}_{br} \), an \( \text{ExpTime} \) Fragment

\( \mathcal{ALCHOIQ} \), the standard DL that we have chosen as a basis in this article, is known to have a \( \text{NExpTime} \)-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 respectively disallowing number restrictions (\( Q \)), nominals (\( O \)), and inverse roles (\( I \)), KB satisfiability is feasible in \( \text{ExpTime} \). 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 \( \mathcal{ALCHOIQ}_{br} \) be the DL that is defined just like \( \mathcal{ALCHOIQ}_{br} \), except concepts of the forms \( \leq n r.C \) and \( \geq n r.C \) 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 2.6.** Finite satisfiability of \( \mathcal{ALCHOIQ}_{br} \) KBs is \( \text{ExpTime} \)-complete.

Proof. \( \text{ExpTime} \) hardness holds already for KB satisfiability in \( \mathcal{ALC} \) (Schild 1991). The \( \text{ExpTime} \) membership follows from the translation of \( \mathcal{ALCHOIQ}_{br} \) 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 3 of an \( \mathcal{ALCHOIQ}_{br} \) KB yields a \( \mathcal{GF}^2 \) sentence with constants. Indeed, deciding (unrestricted) satisfiability in this fragment was shown to be in \( \text{ExpTime} \) 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 \( \mathcal{ALCHOIQ}_{br} \) KBs collapses to unrestricted satisfiability. □

For \( \mathcal{ALCHOQ}_{br} \), \( \text{ExpTime} \)-membership of (unrestricted) satisfiability can also be inferred from the literature, as 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 results for static verification and planning that we establish in the following for \( \mathcal{ALCHOIQ}_{br} \) 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 \( \text{ExpTime} \) (Pratt-Hartmann 2005), and the standard translation in Table 3 (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, as it does not support all of the constructs we argued for in Section 2.2. In particular, it does not have nominals, on which our techniques rely. Hence, the results that we give for \( \mathcal{ALCHOIQ}_{br} \) do not directly extend to \( \mathcal{ALCHIQ}_{br} \).
3 UPDATING GSD

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 \( \text{ALCHIQbr} \) 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 \( \beta \) are defined by the following grammar:

\[
\beta \rightarrow (A \oplus C) \mid (A \ominus C) \mid (p \oplus r) \mid (p \ominus r),
\]

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:

\[
\alpha \rightarrow \epsilon \mid \beta \cdot \alpha \mid (\mathcal{K} \sqcap \alpha \parallel \alpha) \cdot \alpha,
\]

where \( \beta \) is a basic action, \( \mathcal{K} \) is an arbitrary \( \text{ALCHIQbr} \)-formula, and \( \epsilon \) denotes the empty action.

A substitution is a function \( \sigma \) from \( \text{N}_V \) to \( \text{N}_I \). For a formula, an action, or an action sequence \( \gamma \), we use \( \sigma(\gamma) \) to denote the result of replacing in \( \gamma \) every occurrence of a variable \( x \) by the individual \( \sigma(x) \). An action \( \alpha' \) is ground if it has no variables, and \( \alpha' \) is a ground instance of an action \( \alpha \) if \( \alpha' = \sigma(\alpha) \) for some substitution \( \sigma \).

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

Formally, the semantics of actions is defined as follows.

Definition 3.2. Consider an instance \( I \), and let \( E \) be a concept or role name. If \( E \) is a concept, let \( W \subseteq \Delta^I \), and if \( E \) is a role, let \( W \subseteq \Delta^I \times \Delta^I \). Then \( I \oplus_E W \) (respectively, \( I \ominus_E W \)) denotes the instance \( I' \) such that \( \Delta^{I'} = \Delta^I \), and

\[
-E' = E \cup W \text{ (respectively, } E' = E \setminus W), \text{ and }
-E_1' = E_1, \text{ for all symbols } E_1 \neq E.
\]

Given a ground action \( \alpha \), we define a mapping \( S_\alpha \) from instances to instances as follows:

\[
S_\epsilon(I) = I, \quad S_{(A \oplus C)} \cdot \alpha(I) = S_{\alpha}(I \oplus_A C^I), \quad S_{(A \ominus C)} \cdot \alpha(I) = S_{\alpha}(I \ominus_A C^I), \quad S_{(p \oplus r)} \cdot \alpha(I) = S_{\alpha}(I \oplus_{p,r} r^I), \quad S_{(p \ominus r)} \cdot \alpha(I) = S_{\alpha}(I \ominus_{p,r} r^I),
\]

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

Example 3.3. The following instance \( I_1 \) 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:

\[
\begin{align*}
\text{Prj}^\mathcal{I}_1 &= \{p_1, p_2\}, \\
\text{Empl}^\mathcal{I}_1 &= \{e_1, e_3, e_7\}, \\
\text{worksFor}^\mathcal{I}_1 &= \{(e_1, p_1), (e_3, p_1), (e_7, p_1), (e_7, p_2)\}.
\end{align*}
\]

We consider individuals \(p_i\) with \(p_i^\mathcal{I} = p_i\) for projects and analogously individuals \(e_i\) for employees. The following action \(\alpha_1\) 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.

\[
\alpha_1 = (\text{ActivePrj} \ominus \{p_1\}) \cdot (\text{FinishedPrj} \oplus \{p_1\}) \cdot (\text{Empl} \ominus \forall \text{worksFor}.\{p_1\})
\]

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

\[
\begin{align*}
\text{Prj}^\mathcal{I}_1 &= \{p_1, p_2\}, \\
\text{Empl}^\mathcal{I}_1 &= \{e_7\}, \\
\text{worksFor}^\mathcal{I}_1 &= \{(e_1, p_1), (e_3, p_1), (e_7, p_1), (e_7, p_2)\}.
\end{align*}
\]

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 shown earlier. 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, for example, by Brenas et al. (2014). However, this would quickly lead to the undecidability of all reasoning problems considered in this article.

Note that we have not defined the semantics of actions with variables (i.e., of nonground 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, such as by grounding.

**Example 3.4.** The following action \(\alpha_2\) with variables \(x, y, z\) transfers the employee \(x\) from project \(y\) to project \(z\):

\[
\alpha_2 = (x : \text{Empl} \land y : \text{Prj} \land z : \text{Prj} \land (x, y) : \text{worksFor}) \wedge \\
((\text{worksFor} \ominus \{(x, y)\}) \cdot (\text{worksFor} \oplus \{(x, z)\})).
\]

Under the substitution \(\sigma\) with \(\sigma(x) = e_1\), \(\sigma(y) = p_1\), and \(\sigma(z) = p_2\), the action \(\alpha_2\) 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 worksFor link between \(e_1\) and \(p_1\), and creates a 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, whereas 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 we are interested only in finite sequences of actions in this article, 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.

ACM Transactions on Computational Logic, Vol. 18, No. 4, Article 27. Publication date: November 2017.
Example 3.5. Consider instance $I_1$ from Example 3.3, and let $\text{ADom}$ be a shorthand for the concept $\text{Empl} \sqcup \text{Prij} \sqcup \text{ActivePrij} \sqcup \text{FinishedPrij} \sqcup \exists \text{worksFor.}$ $\top \sqcup \exists \text{worksFor}. \top$. Intuitively, $\text{ADom}$ collects the active domain of the instance $I_1$.

The following action $a_3$ adds a new employee $x$ to work for project $p_2$:

$$a_3 = (x : \neg \text{ADom} \land p_2 : \text{Prij})?((\text{Empl} \oplus \{x\}) \cdot (\text{worksFor} \oplus \{(x, p_2)\})).$$

Under the substitution $\sigma$ with $\sigma(x) = e_8$, the action $a_3$ 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 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 $\text{TR}_a(\mathcal{K})$ that rewrites $\mathcal{K}$ by incorporating the possible effects of an action $a$. Intuitively, the models of $\text{TR}_a(\mathcal{K})$ are exactly the instances $I$ such that applying $a$ on $I$ leads to a model of $\mathcal{K}$. In other words, $\text{TR}_a(\mathcal{K})$ can be seen as the weakest precondition of $\mathcal{K}$ for $a$. Having $\text{TR}_a(\mathcal{K})$ as an $\mathcal{ALCHOI}Q b r$ 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 $\alpha$, we define $\text{TR}_\alpha(\mathcal{K})$ as follows:

$$\text{TR}_\alpha(\mathcal{K}) = \mathcal{K}$$

$$\text{TR}_{(A \oplus C)}(\mathcal{K}) = (\text{TR}_\alpha(\mathcal{K}))_{A \leftarrow A \land C}$$

$$\text{TR}_{(A \odot C)}(\mathcal{K}) = (\text{TR}_\alpha(\mathcal{K}))_{A \leftarrow A \lor C}$$

$$\text{TR}_{(p \oplus r)}(\mathcal{K}) = (\text{TR}_\alpha(\mathcal{K}))_{p \leftarrow p \lor r}$$

$$\text{TR}_{(p \odot r)}(\mathcal{K}) = (\text{TR}_\alpha(\mathcal{K}))_{p \leftarrow p \land r}$$

$$\text{TR}_{(K_1 \oplus K_2 \mid a_1 \sqcup a_2)}(\mathcal{K}) = (\neg K_1 \lor \text{TR}_\alpha(a_1 \mid K_1) \land (K_1 \lor \text{TR}_\alpha(a_2 \mid K_1))).$$

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

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

Proof. We prove the claim by induction on $\ell(\alpha)$, defined as follows: $\ell(\varepsilon) = 0$, $\ell(\beta \cdot \alpha) = 1 + \ell(\alpha)$, and $\ell(\mathcal{K} \sqcup a_1 \sqcup a_2 \cdot \alpha_3) = 1 + \ell(a_1) + \ell(a_2) + \ell(\alpha_3)$. In the base case where $\ell(\alpha) = 0$ and $\alpha = \varepsilon$, we have $S_\alpha(I) = I$ and $\text{TR}_\alpha(\mathcal{K}) = \mathcal{K}$ by definition, and thus the claim holds.

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

For the cases $\alpha = (A \odot C) \cdot \alpha'$, $\alpha = (p \oplus r) \cdot \alpha'$, and $\alpha = (p \odot r) \cdot \alpha'$, the argument is analogous.
Finally, we consider \( \alpha = (K_1 ? \alpha_1[\alpha_2]) \cdot \alpha' \) and assume an arbitrary \( I \). We consider the case where \( I \models K_1 \); the case where \( I \not\models K_1 \) is analogous. By definition, \( S_\alpha(I) = S_{\alpha_1}[\alpha_2](I) \). By the induction hypothesis, we know that \( S_{\alpha_1}[\alpha_2](I) = K \) iff \( I \models TR_{\alpha_1}[\alpha_2](K) \), so \( S_\alpha(I) = K \) iff \( I \models TR_{\alpha_1}[\alpha_2](K) \). Since \( I \models K_1 \) and \( TR(K_1 ? \alpha_1[\alpha_2]) \cdot \alpha'(K) = (\neg K_1 \lor TR_{\alpha_1}[\alpha_2](K)) \land (K_1 \lor TR_{\alpha_2}[\alpha_2](K)) \), it follows that \( S_\alpha(I) = K \) iff \( I \models TR(K_1 ? \alpha_1[\alpha_2]) \cdot \alpha'(K) \).

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

Example 4.3. The following KB \( K_1 \) expresses the following constraints on the project database of our running example: all projects are active or finished, the domain of \( \text{worksFor} \) are the employees, and its range are the projects.

\[
K_1 = (\text{Prj} \subseteq \text{ActivePrj} \sqcup \text{FinishedPrj}) \land (\exists \text{worksFor} \cdot \top \subseteq \text{Empl}) \land (\exists \text{worksFor}^- \cdot \top \subseteq \text{Prj})
\]

By applying our transformation to \( K_1 \) and the action \( \alpha_1 \) from Example 3.3, for example,

\[
\alpha_1 = (\text{ActivePrj} \ominus \{p_1\}) \cdot (\text{FinishedPrj} \ominus \{p_1\}) \cdot (\text{Empl} \ominus \forall \text{worksFor} \cdot \{p_1\}),
\]

we obtain the following KB \( TR_{\alpha_1}(K_1) \):

\[
(\text{Prj} \subseteq (\text{ActivePrj} \cap \neg \{p_1\}) \sqcup (\text{FinishedPrj} \cap \{p_1\})) \land (\exists \text{worksFor} \cdot \top \subseteq \text{Empl} \cap \exists \text{worksFor}^- \cdot \{p_1\}) \land (\exists \text{worksFor}^- \cdot \top \subseteq \text{Prj}).
\]

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 of checking whether the execution of an action always preserves the satisfaction of integrity constraints given by a KB.

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

- (SV) Given an action \( \alpha \) and a KB \( K \), is \( \alpha \) \( K \)-preserving?

Using the transformation \( TR_{\alpha}(K) \) shown previously, we can reduce static verification to finite (un)satisfiability of \( ALC\text{CHOI} Qbr \) KBs: an action \( \alpha \) is not \( K \)-preserving iff some finite model of \( K \) does not satisfy \( TR_{\alpha'}(K) \), where \( \alpha^* \) is a “canonical” grounding of \( \alpha \) obtained by replacing each variable with a fresh individual. Formally, we have the following theorem.

Theorem 5.2. Assume a (complex) action \( \alpha \) and a KB \( K \). Then the following are equivalent:

(i) The action \( \alpha \) is not \( K \)-preserving.

(ii) \( K \land \neg TR_{\alpha^*}(K) \) is finitely satisfiable, where \( \alpha^* \) is obtained from \( \alpha \) by replacing each variable with a fresh individual name not occurring in \( \alpha \) and \( K \).

Proof. (i) implies (ii). Assume that there exist a ground instance \( \alpha' \) of \( \alpha \) and a finite instance \( I \) such that \( I \models K \) and \( S_{\alpha'}(I) \not\models K \). Then by Theorem 4.2, \( I \not\models TR_{\alpha'}(K) \). Thus, \( I \models \neg TR_{\alpha'}(K) \). Suppose that \( \alpha_1 \rightarrow x_1, \ldots, \alpha_n \rightarrow x_n \) is the substitution that transforms \( \alpha \) into \( \alpha' \). Suppose also that \( \alpha'_1 \rightarrow x_1, \ldots, \alpha'_n \rightarrow x_n \) is the substitution that transforms \( \alpha \) into \( \alpha^* \). Take the instance \( I^* \) that coincides with \( I \) except for \( \alpha'_1 \rightarrow o^* \). Then \( I^* \models K \land \neg TR_{\alpha^*}(K) \).

(ii) implies (i). Assume that \( K \land \neg TR_{\alpha^*}(K) \) is finitely satisfiable—that is, there is an instance \( I \) such that \( I \models K \) and \( I \not\models TR_{\alpha^*}(K) \). Then by Theorem 4.2, \( S_{\alpha^*}(I) \not\models K \).

ACM Transactions on Computational Logic, Vol. 18, No. 4, Article 27. Publication date: November 2017.
Example 5.3. The action $\alpha_1$ from Example 3.3 is not $\mathcal{K}_1$-preserving: $I_1 \models \mathcal{K}_1$, but $S_{\alpha_i}(I_1) \not\models \mathcal{K}_1$ since the concept inclusion $\exists \text{worksFor}. \top \sqsubseteq \text{Empl}$ is violated. This is reflected in the fact that $I_1 \not\models \TR_{\alpha_i} (\mathcal{K}_1)$, as can be readily checked. Indeed, $(\exists \text{worksFor}. \top)^{I_1} = \{ e_1, e_2, e_3 \}$ while $(\text{Empl} \sqcap \exists \text{worksFor}. (\neg p_1))^{I_1} = \{ e_1 \}$, violating the second conjunct of $\TR_{\alpha_i} (\mathcal{K}_1)$. Intuitively, values removed from $\text{Empl}$ should also be removed from $\text{worksFor}$, as in the following $\mathcal{K}_1$-preserving action:

$$\alpha'_1 = (\text{ActivePrj} \sqcap \{ p_1 \}) \cdot (\text{FinishedPrj} \sqcap \{ p_1 \}) \cdot (\text{Empl} \sqcup \forall \text{worksFor} \cdot \{ p_1 \}) \cdot (\exists \text{worksFor} \sqcap \{ p_1 \}).$$

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

$$(\forall \text{Prj} \subseteq (\text{ActivePrj} \sqcap \neg \{ p_1 \}) \sqcup (\text{FinishedPrj} \sqcup \{ p_1 \})) \land$$

$$(\exists (\text{worksFor} \setminus (\text{worksFor} \cdot \{ p_1 \})) \cdot \top \sqsubseteq \text{Empl} \sqcap \exists (\text{worksFor} \setminus (\text{worksFor} \cdot \{ p_1 \})) \cdot \neg \{ p_1 \}) \land$$

$$(\exists (\text{worksFor} \setminus (\text{worksFor} \cdot \{ p_1 \})) \cdot \top \sqsubseteq \text{Prj} \sqsubseteq \text{Empl})$$

The preceding 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 $\mathcal{K}$ and $\mathcal{K} \land \neg \TR_{\alpha} (\mathcal{K})$ may be of size exponential in $\alpha$, we can avoid generating it all at once. More precisely, we use a nondeterministic polynomial-time many-one reduction that builds only $\mathcal{K} \land \neg \TR_{\alpha} (\mathcal{K})$ for a fragment $\TR_{\alpha} (\mathcal{K})$ that corresponds to one fixed way of choosing one of $\alpha_1$ or $\alpha_2$ for each conditional action $\mathcal{K}' \models \alpha_1 \parallel \alpha_2$ in $\alpha$ (intuitively, we can view $\TR_{\alpha} (\mathcal{K})$ as one conjunct of the DNF of $\neg \TR_{\alpha} (\mathcal{K})$, where axioms and assertions are treated as propositions). Such a $\TR_{\alpha} (\mathcal{K})$ has polynomial size, and it can be built nondeterministically in polynomial time. We can then show that $\mathcal{K} \land \neg \TR_{\alpha} (\mathcal{K})$ is polynomial size, and it can be built nondeterministically in polynomial time. Hence, we can solve the latter task can be done in nondeterministic exponential time, and 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 $\ALCHOIbr$ and ExpTime-complete when the input KB is expressed in $\ALCHOIbr$.

**Proof.** For the hardness, we note that for both $\ALCHOIbr$ and $\ALCHOIbr$, 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 $(\mathcal{K} \land (A \subseteq \neg A') \land (o : A))$-preserving, where $A$ and $A'$ are fresh concept names and $o$ is a fresh individual. Hence, hardness for $\ALCHOIbr$ follows from Theorem 2.6 and for $\ALCHOIbr$ from Proposition 2.7.

Obtaining a matching upper bound is slightly more involved. Theorem 5.2 establishes that verifying whether an action $\alpha$ is not $\mathcal{K}$-preserving, where $\mathcal{K}$ is an $\ALCHOIbr$ KBs, reduces to finite satisfiability of a KB $\mathcal{K} \land \neg \TR_{\alpha} (\mathcal{K})$ in $\ALCHOIbr$, 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, $\mathcal{K} \land \neg \TR_{\alpha} (\mathcal{K})$ is finitely satisfiable iff this set contains some $\mathcal{K}'$ such that $\mathcal{K} \land \mathcal{K}'$ is finitely satisfiable. Hence, the complement of the (SV) problem reduces to deciding whether there is some $\mathcal{K}'$ in the set for which $\mathcal{K} \land \mathcal{K}'$ is satisfiable. The desired upper bounds can now be easily inferred. In the case of $\ALCHOIbr$, we nondeterministically build the right $\mathcal{K}'$ in polynomial time and then test for finite satisfiability of $\mathcal{K} \land \mathcal{K}'$. Since the latter is an $\ALCHOIbr$ KB whose size is bounded polynomially by the input $\mathcal{K}$ and $\alpha$, deciding its finite satisfiability is in NExpTime (see Theorem 2.6). For $\ALCHOIbr$, we can build in exponential time all of the KBs $\mathcal{K}'$ in the set, which are exponentially many, but each of them is polynomial in the input $\mathcal{K}$ and $\alpha$. Then we iterate over the full set, and for each $\mathcal{K}'$ we decide in ExpTime.
whether \( K \land K' \) is finitely satisfiable (see Proposition 2.7). The upper bound then follows from the fact that doing exponentially many \( \text{ExpTime} \) tests is feasible in \( \text{ExpTime} \).

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

\[
\overline{\text{TR}}_e(K) = \neg K
\]

\[
\overline{\text{TR}}_{(A\oplus C)}(K) = (\text{TR}_a(K))_{A\rightarrow-ALIC}
\]

\[
\overline{\text{TR}}_{(A\oplus C)}(K) = (\text{TR}_a(K))_{A\rightarrow\neg C}
\]

\[
\overline{\text{TR}}_{(p\oplus r)}(K) = (\text{TR}_a(K))_{p\rightarrow\neg p, r}
\]

\[
\overline{\text{TR}}_{(\ell, a_1, a_2)}(K) = (\overline{\text{TR}}_{a_1}(K)) \lor (\overline{\text{TR}}_{a_2}(K))
\]

It can be shown by a straightforward induction on \( \ell(\alpha) \) (as defined in the proof of Theorem 4.2) that \( \overline{\text{TR}}_a(K) \) is logically equivalent to \( \neg \text{TR}_a(K) \) for every \( K \) and every \( \alpha \). Hence, by Theorem 4.2, \( K \land \overline{\text{TR}}_a(K) \) is finitely satisfiable iff \( K \land \neg \text{TR}_a(K) \) is finitely satisfiable iff \( \alpha \) is not \( K \)-preserving.

Now, we define a translation analogous to \( \overline{\text{TR}}_a(K) \), but in the last case, for the conditional axioms, we generate two separate KBs \( K_1 \land \overline{\text{TR}}_{a_1}(K) \) and \( \neg K_1 \land \overline{\text{TR}}_{a_2}(K) \) rather than considering the disjunction of both. We denote by \( \overline{\text{TR}}_a(K) \) the set of all the KBs obtained this way—that is,

\[
\overline{\text{TR}}_e(K) = \{ \neg K \}
\]

\[
\overline{\text{TR}}_{(A\oplus C)\land \alpha}(K) = \{ K'_{\alpha\rightarrow-ALIC} \mid K' \in \overline{\text{TR}}_a(K) \}
\]

\[
\overline{\text{TR}}_{(A\oplus C)\land \neg C}(K) = \{ K'_{\alpha\rightarrow\neg C} \mid K' \in \overline{\text{TR}}_a(K) \}
\]

\[
\overline{\text{TR}}_{(p\oplus r)\land \alpha}(K) = \{ K'_{p\rightarrow\neg p, r} \mid K' \in \overline{\text{TR}}_a(K) \}
\]

\[
\overline{\text{TR}}_{(p\oplus r)\land \neg C}(K) = \{ K'_{p\rightarrow\neg p, r} \mid K' \in \overline{\text{TR}}_a(K) \}
\]

\[
\overline{\text{TR}}_{(\ell, a_1, a_2)}(K) = \{ K_1 \land K' \mid K' \in \overline{\text{TR}}_{a_1}(K) \} \cup \{ \neg K_1 \land K' \mid K' \in \overline{\text{TR}}_{a_2}(K) \}
\]

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

We show this by induction on \( \ell(\alpha) \). The base case is straightforward: if \( \alpha = e \), then \( \overline{\text{TR}}_a(K) = \{ \overline{\text{TR}}_a(K) \} \). For the inductive step, we first consider \( \alpha' = (A \oplus C) \cdot \alpha \). First, we assume that \( I \models K' \land \overline{\text{TR}}_a(K) \). In other words, \( I \models (\overline{\text{TR}}_a(K))_{A\rightarrow-ALIC} \), and hence \( S_{(A\oplus C)}(I) \models \overline{\text{TR}}_a(K) \). We can apply the induction hypothesis to \( S_{(A\oplus C)}(I) \) and \( \overline{\text{TR}}_a(K) \) to infer that there exists \( K'' \in \overline{\text{TR}}_a(K) \) such that \( S_{(A\oplus C)}(I) \models K'' \), and hence \( I \models K''_{A\rightarrow-ALIC} \). Since \( \overline{\text{TR}}_a(K) = \{ K'_{A\rightarrow-ALIC} \mid K' \in \overline{\text{TR}}_a(K) \} \), we have that \( K'' = K''_{A\rightarrow-ALIC} \) is such that \( K'' \in \overline{\text{TR}}_a(K) \) and \( I \models K'' \), as desired. For the converse, if \( I \models K'' \) for some \( K'' \in \overline{\text{TR}}_a(K) \), by definition we have that there is some \( K' \in \overline{\text{TR}}_a(K) \) such that \( K'' = K'_{A\rightarrow-ALIC} \), and since \( I \models K'_{A\rightarrow-ALIC} \) we have that \( S_{(A\oplus C)}(I) \models K' \). Using the induction hypothesis on \( S_{(A\oplus C)}(I) \) and \( K' \), we get \( S_{(A\oplus C)}(I) \models \overline{\text{TR}}_a(K) \), and hence \( I \models (\overline{\text{TR}}_a(K))_{A\rightarrow-ALIC} \), that is, \( I \models \overline{\text{TR}}_a(K) \), as desired. The cases of \( \alpha' = (A \oplus C) \cdot \alpha \), \( \alpha' = (p \oplus r) \cdot \alpha \), and \( \alpha' = (p \ominus r) \cdot \alpha \) are analogous.
Finally, consider $\alpha' = (K_1 \uplus \alpha_2) \cdot \alpha$. We first show that if $I \models \overline{\text{TR}}_{\alpha'}(K)$, then there is some $K' \in \overline{\text{TR}}_{\alpha'}(K)$ such that $I \models K'$. By definition, $\overline{\text{TR}}_{\alpha'}(K) = (K_1 \land \overline{\text{TR}}_{\alpha_2}(K)) \lor (\neg K_1 \land \overline{\text{TR}}_{\alpha_1}(K))$. Thus, if $I \models \overline{\text{TR}}_{\alpha'}(K)$, then one of $I \models K_1 \land \overline{\text{TR}}_{\alpha_1}(K)$ or $I \models \neg K_1 \land \overline{\text{TR}}_{\alpha_2}(K)$ holds. In the former case, we can use the induction hypothesis to conclude that there exists some $K' \in \overline{\text{TR}}_{\alpha_1}(K)$ such that $I \models K'$. Since $K_1 \land K' \in \overline{\text{TR}}_{\alpha'}(K)$ by definition, the claim follows. The latter case is analogous. For the converse, we assume that there exists some $K' \in \overline{\text{TR}}_{\alpha'}(K)$ such that $I \models K'$. By definition, this $K'$ must be of the form $K_1 \land K''$ with $K'' \in \overline{\text{TR}}_{\alpha_2}(K)$, or of the form $\neg K_1 \land K''$ with $K'' \in \overline{\text{TR}}_{\alpha_1}(K)$. In the former case, it follows from the induction hypothesis that $I \models K_1 \land \overline{\text{TR}}_{\alpha_1}(K)$, and hence $I \models (K_1 \land \overline{\text{TR}}_{\alpha_1}(K)) \lor (\neg K_1 \land \overline{\text{TR}}_{\alpha_2}(K))$ and the claim follows. The second case, where $K'$ is of the form $\neg K_1 \land K''$, 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 $K$ expressing constraints. We can also consider other interesting variations of the problem where, for example, we have a pair of KBs $K_{\text{pre}}$ and $K_{\text{post}}$ instead of (or in addition to) $K$, and we want to decide whether executing the action on any model of $K_{\text{pre}}$ (and $K$) leads to a model of $K_{\text{post}}$ (and $K$). The reasoning techniques and upper bounds presented earlier 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, as 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 $(A_1 \subseteq \neg A'_1) \land \cdots \land (A_n \subseteq \neg A'_n)$, where each $A_i, A'_i$ is a concept name, and for ground sequences of basic actions of the forms $(A \uplus C)$ and $(A \land C)$, 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 $K$ and an action $\alpha$ such that $G$ is 3-colorable iff $\alpha$ is not $K$-preserving. For every $v \in V$, we use three concept names $A^0_v, A^1_v, A^2_v$ for the three possible colors of the vertex $v$. In addition, we employ a concept name $D$. Let $K$ be the following KB:

$$K = (D \subseteq \neg D) \land \bigwedge_{(u, v) \in E \land 0 \leq c \leq 2} (A^c_v \subseteq \neg A^c_u).$$

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

1. $\alpha_1 = (D \uplus \{o\}) \cdot (B_1 \uplus \{o\}) \cdots (B_n \uplus \{o\})$,
2. $\alpha_2 = (B_1 \uplus A^1_v) \cdot (B_1 \uplus A^1_v) \cdot (B_1 \uplus A^2_v)$, for all $i \in \{1, \ldots, n\}$, and
3. $\alpha_3 = (D \uplus B_1) \cdots (D \uplus B_n)$.

Assume that $I$ is a model of $K$ such that $S_\alpha(I) \not\models K$. We argue that then $G$ is 3-colorable. Indeed, since $\alpha$ does not modify the extensions of concepts $A^c_v, S_\alpha(I) \not\models K$ may only hold if $o^I$ is
in the extension of $D$ after applying $\alpha$ on $I$. Due to $\alpha$, this requires $o^T$ 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 $I$. Due to $\alpha_1$ and the various $\alpha_2^i$, it must be the case that $o^T$ is in the extension in $I$ of one of $A_0^1, A_1^1$, or $A_2^i$, for each $i \in \{1, \ldots, n\}$. For every $\nu \in V$, let $\text{col}(\nu) \in \{0, 1, 2\}$ be a value such that $o^T$ is in the extension of $A_2^\nu \cap \text{col}(\nu)$ in $I$. Since $I$ satisfies the disjointness axioms in $\mathcal{K}$, the function $\text{col}$ is a proper 3-coloring of $G$.

Suppose that $G$ is 3-colorable and a proper coloring of $G$ is given by a function $\text{col} : V \rightarrow \{0, 1, 2\}$. Take any instance $I$ with $\Lambda^I = \{e\}$ and such that (i) $|\alpha|^T = e$, (ii) $D^I = \emptyset$, and (iii) $e \in (A_0^\epsilon)^T$ iff $\text{col}(\nu) = c$. Since $\text{col}$ is a proper coloring of $G$, $I$ is a model of $\mathcal{K}$. As is easily seen, $S_\alpha(I) \not\models \mathcal{K}$.

We next present a rich variant of $DL$-$\text{Lite}_\mathcal{R}$, which we call $DL$-$\text{Lite}_{\mathcal{R}}^+$, 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 in the following, this allows us to express the effects of actions inside $DL$-$\text{Lite}_{\mathcal{R}}^+$ KBs.

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

- Concept inclusions have the form $C_1 \sqsubseteq C_2$ or $C_1 \not\sqsubseteq C_2$, with $\{C_1, C_2\} \subseteq \mathcal{C}_G \cup \{\exists p. \top | p \in \mathcal{R}_G\}$.
- Role inclusions have the form $r_1 \sqsubseteq r_2$ with $\{r_1, r_2\} \subseteq \mathcal{R}_G \cup \{p^- | p \in \mathcal{R}_G\}$.
- Role assertions are defined as for $ALCHQIbr$, but in concept assertions $o : C$, we require $C \in \mathcal{B}^+$, where $\mathcal{B}^+$ is the smallest set of concepts such that
  - $\mathcal{C}_G \subseteq \mathcal{B}^+$,
  - $\{o\} \in \mathcal{B}^+$, for all $o \in \mathcal{N}_I$,
  - $\exists p. \top \in \mathcal{B}^+$, for all roles $r$,
  - $\{B_1 \sqcup B_2, B_1 \sqcap B_2, \neg B_1\} \subseteq \mathcal{B}^+$, for all $B_1, B_2 \in \mathcal{B}^+$.
- Formulas and KBs are defined as for $ALCHQIbr$, but the operator $\neg$ may occur only in front of assertions.

A $DL$-$\text{Lite}_{\mathcal{R}}$ KB $\mathcal{K}$ is a $DL$-$\text{Lite}_{\mathcal{R}}^+$ 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 form $o : A$ with $A \in \mathcal{C}_G$, and $(o, o') : p$ with $p \in \mathcal{R}_G$.

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

**Theorem 5.7.** Finite satisfiability of $DL$-$\text{Lite}_{\mathcal{R}}^+$ 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 $DL$-$\text{Lite}_{\mathcal{R}}^+$ KB $\mathcal{K}$, there is a model $I$ of $\mathcal{K}$ such that (i) $Q^I = \emptyset$ for all concept and role names $Q$ that do not appear in $\mathcal{K}$, and (ii) $|\Lambda^I| \leq m + m \cdot k + 2 \cdot k$, where $m$ is the number of individuals and $k$ the number of (possibly complex) roles that appear in $\mathcal{K}$. A nondeterministic algorithm running in polynomial time in the size of $\mathcal{K}$ is then apparent: simply guess an interpretation $I$ with $|\Lambda^I| \leq m + m \cdot k + 2 \cdot k$ and then check that $I$ is a model of $\mathcal{K}$ (model checking requires only polynomial time because $DL$-$\text{Lite}_{\mathcal{R}}^+$ KBs correspond to first-order formulas with a bounded—namely at most $2$—number of variables (Vardi 1995)).

Consider a model $I = (\Lambda^I, \cdot)^I$ of $\mathcal{K}$. Let $\mathcal{T}$ be the set of inclusions $\alpha_1 \sqsubseteq \alpha_2$ appearing in $\mathcal{K}$ such that $I \models \alpha_1 \sqsubseteq \alpha_2$. Similarly, let $\mathcal{R}$ be the set of inclusions $\beta$ appearing in $\mathcal{K}$ of the form $r_1 \sqsubseteq r_2$ or $r_1 \not\sqsubseteq r_2$, with $\{r_1, r_2\} \subseteq \mathcal{R}_G \cup \{p^- | p \in \mathcal{R}_G\}$, such that $I \models \beta$. Let $\mathcal{R}^+$ be the closure of $\mathcal{R}$ under the following rules:
(a) if \( \alpha_1 \subseteq \alpha_2 \in \mathcal{R}^* \), then \( \alpha_1 \subseteq \alpha_1 \in \mathcal{R}^* \) and \( \alpha_2 \subseteq \alpha_2 \in \mathcal{R}^* \); 
(b) if \( \alpha_1 \subseteq \alpha_2 \in \mathcal{R}^* \) and \( \alpha_2 \subseteq \alpha_3 \in \mathcal{R}^* \), then \( \alpha_1 \subseteq \alpha_3 \in \mathcal{R}^* \); and 
(c) if \( \alpha_1 \subseteq \alpha_2 \in \mathcal{R}^* \), then \( \alpha_1^{-1} \subseteq \alpha_2^{-1} \in \mathcal{R}^* \).

Without loss of generality, we can assume for all concept and role names \( Q \) that if \( Q \) does not appear in \( \mathcal{K} \), then \( Q^J = \emptyset \). Let \( \Delta \equiv \{ o^I \mid \text{individual } o \text{ appears in } \mathcal{K} \} \). Let \( \Delta' \) be a \( \subseteq \)-minimal set such that \( \Delta \subseteq \Delta' \) and such that the following rule is obeyed:

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

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

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

1. \( \Delta^J = \Delta' \cup \{ \text{sink}(r) \mid r \in N \text{ and sink}(r) \text{ is defined} \} \); 
2. \( o^J = o^I \) for all individuals \( o \) that appear in \( \mathcal{K} \); 
3. \( A^J = A^I \cap \Delta^J \), for all concept names \( A \); 
4. \( p^J = p^I \) for all role names \( p \), and each \( \{ e_1, e_2 \} \subseteq \Delta^J \), we have that \( (e_1, e_2) \in p^J \), if one of the following holds:  
   (a) \( (e_1, e_2) \in p^I \), 
   (b) \( e_2 = \text{sink}(r) \) for some \( r \in N \) such that \( r \subseteq p \in \mathcal{R}^* \), and \( e_1 \in (\exists r.\top)^I \), \text{ or} 
   (c) \( e_1 = \text{sink}(r) \) for some \( r \in N \) such that \( r \subseteq p \in \mathcal{R}^* \), and \( e_2 \in (\exists r^- \top)^I \).

We now show that \( \mathcal{J} \) remains a model of \( \mathcal{K} \). For this, it suffices to show that (I) for every concept inclusion \( C_1 \subseteq C_2 \) that appears in \( \mathcal{K} \), \( I \models C_1 \subseteq C_2 \) implies \( J \models C_1 \subseteq C_2 \); (II) for every role inclusion \( r_1 \subseteq r_2 \) that appears in \( \mathcal{K} \), \( I \models r_1 \subseteq r_2 \) implies \( J \models r_1 \subseteq r_2 \); and (III) for every assertion \( \beta \) that appears in \( \mathcal{K} \), \( I \models \beta \) iff \( J \models \beta \). Points (I) through (III) are sufficient because in \( \mathcal{K} \) the operator \( \top \) is allowed to occur only in front of assertions.

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

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

Next, we prove Claim (II). Suppose that there exist \( e, e' \in \Delta^J \) such that \( (e, e') \in r_1^J \). There are two cases:
due to condition (c4.a) and the fact that \((e, e') \in r_1^J\). Then \((e, e') \in r_2^J\) due to condition (c4.a) and the fact that \((e, e') \in r_2^J\).

Suppose that \(r_1 \in \text{NR}\). Then it must be the case that \(e' = \text{sink}(r)\) for some \(r\) such that \((e, e') \in r^J\), \(e'\) is anonymous and \(r \subseteq r_1 \in \mathcal{R}^e\). Since \(r \subseteq r_2 \in \mathcal{R}^e\), due to condition (c4.b) it must be the case that \((e, \text{sink}(r)) \in r_2^J\) — for instance, \((e, e') \in r_2^J\). Showing that \((e, e') \in r_2^J\) in case \(r_1 \in \{p^- \mid p \in \text{NR}\}\) 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 \(\mathcal{J} \models (o, o') : r \iff I \models (o, o') : r\) holds for all role names \(r\) and all individuals \(o, o'\) that occur in \(\mathcal{K}\). Thus, Claim (III) trivially holds in case \(\beta\) in the claim is a role membership assertion. Suppose that \(\beta\) is an assertion \(o : C\) with \(C \in \mathcal{B}^+\) (recall Definition 5.6). Due to conditions (c2) and (c3), it only remains to make sure that for any individual \(o\) and any (possibly complex) role \(r\) that occur in \(\mathcal{K}\), we have \(o^I \in (\exists r. \top)^I\) if \(o^J \in (\exists r. \top)^J\). The “if” direction follows direction from condition (c4) in the construction of \(\mathcal{J}\). The “only if” direction follows from the fact that \(\Delta'\) 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 \(\mathcal{K} \land (\exists r. \top)\) can be a Boolean combination of assertions only. We observe that most of the examples presented in this article use the latter type of actions.

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

DL-Lite\(^+\)\(_R\) is expressive enough to allow us to reduce static verification for simple actions to finite unsatisfiability, and similarly as earlier, we can use a nondeterministic 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\(^+\)\(_R\) 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\(^+\)\(_R\), employing the same reduction as in the proof of Theorem 5.4.

For the upper bound, consider a DL-Lite\(^+\)\(_R\) KB \(\mathcal{K}\) and a simple action \(\alpha\). We proceed analogously to the proof of Theorem 5.4. From Theorem 5.2, we know that \(\alpha\) is not \(\mathcal{K}\)-preserving iff \(\mathcal{K} \land \neg \text{TR}_{\alpha}\)(\(\mathcal{K}\)) is finitely satisfiable. Moreover, we have shown that \(\mathcal{K} \land \neg \text{TR}_{\alpha}\)(\(\mathcal{K}\)) is finitely satisfiable iff there exists a \(\mathcal{K}' \in \overline{\text{TR}}_{\alpha}\)(\(\mathcal{K}\)) such that \(\mathcal{K} \land \mathcal{K}'\) is finitely satisfiable, and \(\mathcal{K}'\) can be obtained nondeterministically in polynomial time and is of size polynomial in \(\alpha\) and \(\mathcal{K}\). The KB \(\mathcal{K}'\) is not a DL-Lite\(^+\)\(_R\) KB, but it can be transformed into an equisatisfiable DL-Lite\(^+\)\(_R\) KB in linear time. To this end, turn \(\mathcal{K}'\) into negation normal form—for instance, push \(\neg\) inside so that \(\neg\) occurs in front of inclusions and assertions only. Then in the resulting KB \(\mathcal{K}'\), replace every occurrence of \(\neg((B_1 \subseteq B_2)\) by \(o : B_1 \cap \neg B_2\) and every occurrence of \(\neg(r_1 \subseteq r_2)\) by \(o, o' : r_1 \setminus r_2\), where \(o, o'\) are fresh individuals. Clearly, the preceding transformations preserve satisfiability. Moreover, since in \(\mathcal{K}\) the operator \(\neg\) may occur only in front of assertions, and \(\alpha\) is simple, every inclusion in the resulting \(\mathcal{K}'\) already appears in \(\mathcal{K}\). This implies that \(\mathcal{K}'\) is a DL-Lite\(^+\)\(_R\) 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 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 AI.

We use DLs to describe states of KBs, which may act as goals or preconditions. 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 \( I = \langle \Delta^I, \cdot^I \rangle \) 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 \( I \) (of length \( n \)), if there exists a finite instance \( I^* = \langle \Delta^{I^*}, \cdot^{I^*} \rangle \) such that \( S_{\alpha_1 \ldots \alpha_n}(I^*) \models \mathcal{K} \), where \( \Delta^I \subseteq \Delta^{I^*} \) and \( \cdot^I = \cdot^{I^*} \) for each \( A \in \mathcal{N}_C, p^I = p^{I^*} \) for each \( p \in \mathcal{N}_R \), and \( \sigma^I = \sigma^{I^*} \) for each \( \sigma \in \mathbb{N}_I \) 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 preceding definition of planning 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 \( I \), and a goal KB \( \mathcal{K} \), does there exist a plan for \( \mathcal{K} \) from \( I \) ?

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

(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 precondition 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, \( \mathcal{K}_{pre} = \mathcal{K}_{ic} \land (x : FinishedPrj) \) and \( \mathcal{K} = x : ActivePrj \) 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 \( I_1 \), and consider the following goal KB, which requires that \( p_1 \) is not an active project, and that \( e_1 \) is an employee.

\[
\mathcal{K} = \lnot(p_1 : ActivePrj) \land (e_1 : Empl)
\]

Assume that Act consists of the action \( \alpha_2 \) from Example 3.4 (which we recall for convenience), and of the following actions:

\[
\begin{align*}
\alpha_2 &= (x : Empl \land y : Prj \land z : Prj \land (x, y) : worksFor) \land \\
&((worksFor \lor \{(x, y)\}) \cdot (worksFor \lor \{(x, z)\})) \\
\alpha_3 &= (ActivePrj \lor \{y\}) \cdot (FinishedPrj \lor \{y\}) \cdot (Empl \lor \forall worksFor.\{y\}) \cdot \\
&(worksFor \lor worksFor\lnot\{y\}).
\end{align*}
\]

A plan for \( \mathcal{K} \) from \( I_1 \) is the sequence of actions \( \langle \sigma(\alpha_2), \sigma(\alpha_3) \rangle \), where \( \sigma \) is the substitution from Example 3.4, which maps \( x \) to \( e_1 \), \( y \) to \( p_1 \), and \( z \) to \( p_2 \). To obtain \( S_{\sigma(\alpha_2)\cdot\sigma(\alpha_3)}(I_1) \), we first apply \( \sigma(\alpha_2) \)
to \( I_1 \). The resulting instance \( S_{\sigma(\alpha_2)}(I_1) \) is as follows:

\[
\begin{align*}
\text{Prj}^{S_{\sigma(\alpha_2)}(I_1)} &= \{p_1, p_2\} \\
\text{ActivePrj}^{S_{\sigma(\alpha_2)}(I_1)} &= \{p_1, p_2\} \\
\text{Emp}^{S_{\sigma(\alpha_2)}(I_1)} &= \{e_1, e_3, e_7\} \\
\text{FinishedPrj}^{S_{\sigma(\alpha_2)}(I_1)} &= \{\} \\
\text{worksFor}^{S_{\sigma(\alpha_2)}(I_1)} &= \{(e_1, p_2), (e_3, p_1), (e_7, p_1), (e_7, p_2)\}.
\end{align*}
\]

Finally, we apply \( \sigma(\alpha_1) \) to \( S_{\sigma(\alpha_2)}(I_1) \), and obtain the instance \( S_{\sigma(\alpha_1)}(S_{\sigma(\alpha_2)}(I_1)) = S_{\sigma(\alpha_2) \cdot \sigma(\alpha_1)}(I_1) \) in the following, which reflects the status of the data after applying \( \sigma(\alpha_2), \sigma(\alpha_1) \) to \( I_1 \):

\[
\begin{align*}
\text{Prj}^{S_{\sigma(\alpha_2) \cdot \sigma(\alpha_1)}(I_1)} &= \{p_1, p_2\} \\
\text{ActivePrj}^{S_{\sigma(\alpha_2) \cdot \sigma(\alpha_1)}(I_1)} &= \{p_2\} \\
\text{Emp}^{S_{\sigma(\alpha_2) \cdot \sigma(\alpha_1)}(I_1)} &= \{e_1, e_7\} \\
\text{FinishedPrj}^{S_{\sigma(\alpha_2) \cdot \sigma(\alpha_1)}(I_1)} &= \{p_1\} \\
\text{worksFor}^{S_{\sigma(\alpha_2) \cdot \sigma(\alpha_1)}(I_1)} &= \{(e_1, p_2), (e_7, p_2)\}.
\end{align*}
\]

Clearly, \( S_{\sigma(\alpha_2) \cdot \sigma(\alpha_1)}(I_1) \) satisfies the goal KB as desired—that is, in \( S_{\sigma(\alpha_2) \cdot \sigma(\alpha_1)}(I_1) \), \( p_1 \) 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:

\[
\mathcal{K}_{pre} = (x : \text{Empl}) \land (y : \text{ActivePrj}) \land ((x, y) : \text{worksFor}).
\]

The preceding \( \sigma \) and \( \langle \sigma(\alpha_2), \sigma(\alpha_3) \rangle \) are the desired substitution and plan. Indeed, \( I_1 \models \sigma(\mathcal{K}_{pre}) \), and \( \langle \sigma(\alpha_2), \sigma(\alpha_3) \rangle \) is a plan for \( \sigma(\mathcal{K}) \) from \( I_1 \). 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 \( \mathcal{K}' = \neg(y : \text{ActivePrj}) \land (x : \text{Empl}) \) instead of \( \mathcal{K} \).

Unfortunately, the preceding 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 preconditions of the form \( \mathcal{K} ? \alpha_1 \) rather than full conditional actions of the form \( \mathcal{K} ? \alpha_1 \parallel \alpha_2 \parallel \alpha_3 \parallel \alpha_4 \).

**Theorem 6.3.** The problems (P1) and (P2) are undecidable, already for DL-Lite\(_R\) KBs and simple actions.

**Proof.** The proof is by a reduction from the Word problem. We reduce to (P1) and (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 \( M = (Q, \delta, q_0, q_a, q_f) \), where \( Q \) is a set of states, \( \delta : \{0, 1, b\} \times Q \rightarrow \{0, 1, b\} \times Q \times \{+1, -1\} \) 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 without loss of generality 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 \( \delta \). We also have a pair of actions that "extend" the tape with blank symbols as needed. For the reduction, we use the role \( \text{next} \) and concept names \( \text{Sym}_0, \text{Sym}_1, \text{Sym}_b \), and \( \text{St}_q \) for each \( q \in Q \).

The set \( \text{Act} \) 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

\[
\alpha_{\sigma, q} = ((x_1, x_2) : \text{next} \land x_2 : \text{Sym}_q \land x_2 : \text{St}_q \land (x_2, x_3) : \text{next}) \cdot ((\text{Sym}_\sigma \land (x_2)) \cdot (\text{Sym}_\sigma \lor (x_2)) \cdot (\text{St}_q \lor (x_2)) \cdot (\text{St}_{q'} \lor (x_2 + D))).
\]

To extend the tape with blank symbols, we have the actions \( \alpha_r \) and \( \alpha_e \). In particular,

\[
\alpha_r = (x : (\text{Sym}_0 \sqcup \text{Sym}_1 \sqcup \text{Sym}_b) \land y : \neg(\text{Sym}_0 \sqcup \text{Sym}_1 \sqcup \text{Sym}_b)) \cdot ((\text{next} \lor \{(x, y)\}) \cdot (\text{Sym}_b \lor \{y\})).
\]
The action $\alpha_e$ is obtained from $\alpha_r$ by replacing $(next \oplus \{(x, y)\})$ with $(next \oplus \{(y, x)\})$. Finally, we have an initialization action $\alpha_{init}$, which stores the initial configuration of $M$ in the database. In particular,

$$\alpha_{init} = \left( \bigwedge_{1 \leq i \neq j \leq m} a_i : \neg(a_j) \wedge a_1 : \neg(Sym_0 \sqcup Sym_1 \sqcup Sym_0) \right) \wedge$$

$$(St_q \oplus \{a_1\}) \cdot (Sym_{\sigma_1} \oplus \{a_1\}) \cdot (Sym_{\sigma_2} \oplus \{a_2\}) \cdots (Sym_{\sigma_m} \oplus \{a_m\}) \cdot$$

$$(next \oplus \{(a_1, a_2)\}) \cdot (next \oplus \{(a_2, a_3)\}) \cdots (next \oplus \{(a_{m-1}, a_m)\})),$$

where $w = \sigma_1 \cdots \sigma_m$. We let $\mathcal{K} = a_1 : St_{q_0}$ and the initial database $I$ be empty—for instance, 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 $I$ with $s$ fresh elements is sufficient to find a plan for $\mathcal{K}$ using the actions in $Act$.

The preceding reduction also applies to (P2). It suffices to define a precondition KB $\mathcal{K}_{pre}$ that describes the preceding $I$. Simply let $\mathcal{K}_{pre}$ be the conjunction of $(Sym_0 \sqcup Sym_1 \sqcup Sym_0 \sqcup \exists next \cdot T \sqcup \exists next^- \cdot T \sqsubseteq \bot)$ and $\sqcup_{q \leq q_0} St_q \sqsubseteq \bot$. $\square$

We observe that the preceding proof works for $DL$-Lite KBs—that is, $\mathcal{K}$ and $\mathcal{K}_{pre}$ are $DL$-Lite KBs, but we use $DL$-Lite 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 $I$, but it becomes decidable if the size of $\Delta$ 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 $I$, a goal KB $\mathcal{K}$, and a positive integer $\ell$, does there exist a plan for $\mathcal{K}$ from $I$ where $|\Delta^I| \leq \ell$?

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

Example 6.4. Recall Example 6.2. For $\ell \geq 2$, the substitution $\sigma$ 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 that the goal KB is as follows:

$$\mathcal{K}_1 = \neg(p_1 : ActivePrj) \wedge (e_7 : Empl).$$

Then $\langle \sigma'(\alpha_3) \rangle$ would be a plan of length one for $\sigma(\mathcal{K}_1)$ from instance $I_1$, where $\sigma'$ is the substitution with $\sigma'(x) = e_7$ and $\sigma'(y) = p_1$ and is such that $I_1 \models \sigma'(\mathcal{K}_{pre})$.

We now study the complexity of these problems, assuming that the input bounds $\ell$ 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 $ALCHOI Qbr$ and $ALCHOI br$ KBs.

Proof. It suffices to prove the lower bound for $ALCHOI br$ and the upper bound for $ALCHOI Qbr$. To prove the PSPACE lower bound for $ALCHOI br$, 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 $\ell$, then one can easily simulate Turing machines that use space bounded by $\ell$. For a Turing machine $M$ and a word $w$, we define the set $Act$ of actions, the input instance $I$ and the goal KB $K$ 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 iff there exists a plan from an instance whose domain is bounded by $p(|w|)$.

For the upper bound, we employ a nondeterministic polynomial space procedure that stores in memory a finite instance and nondeterministically 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 $\mathcal{ALCHOIQ}$-formulae is feasible in polynomial space.

We note that we could make the reduction even simpler by removing the actions $a_r$ and $a_{r'}$ 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 $K_{pre}$ and $K$ are in $\mathcal{ALCHOIQ}$ or $\mathcal{ALCHOI}$ and for the restricted case of DL-Lite$^+_R$ 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 $\mathcal{ALCHOIQ}$ KBs and ExpTime-complete for $\mathcal{ALCHOI}$ KBs. It is NP-complete if $K_{pre}$ and $K$ are expressed in DL-Lite$^+_R$ and all actions in Act are simple.

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

For the upper bound for $\mathcal{ALCHOIQ}$, we first guess a variable substitution $\sigma$ and a sequence $P = \langle a_1, \ldots, a_m \rangle$ of at most $\ell$ actions. By Theorem 4.2, it follows that $P$ is a plan as desired iff $\sigma(K_{pre}) \land TR_{a_1 \ldots a_m}(\sigma(K))$ 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 $TR_{a_1 \ldots a_m}(\sigma(K))$ is not negated, and hence, intuitively, we need to decide the existence of an instance that satisfies the negation of all formulas in $TR_\sigma(K)$ rather than satisfying just one of them. To be able to check the finite satisfiability of $\sigma(K_{pre}) \land TR_{a_1 \ldots a_m}(\sigma(K))$ within the desired bounds, we proceed similarly as earlier and consider a procedure that nondeterministically builds a polynomial KB $K'$ such that $\sigma(K_{pre}) \land K'$ is finitely satisfiable iff $\sigma(K_{pre}) \land TR_{a_1 \ldots a_m}(\sigma(K))$ is so.

More in detail, we define a set $TR_\sigma^\wedge(K)$ of KBs that is similar to $TR_\sigma(K)$ except it contains the negation of the formulas in the latter and uses conjunctions rather than implications for the conditional axioms.

\[
\begin{align*}
TR_\sigma^\wedge(K) &= \{K\} \\
TR_\sigma^\wedge(\langle A_{BC} \rangle_{a}) &= \{K'_{A_{BC}} \mid K' \in TR_\sigma(K)\} \\
TR_\sigma^\wedge(\langle A_{\neg IC} \rangle_{a}) &= \{K'_{A_{\neg IC}} \mid K' \in TR_\sigma(K)\} \\
TR_\sigma^\wedge(\langle p_{Br} \rangle_{a}) &= \{K'_{p_{Br}} \mid K' \in TR_\sigma(K)\}
\end{align*}
\]
\[ TR_{\langle p \circ r \rangle, \alpha}^{\land} (\mathcal{K}) = \{ \mathcal{K}'_p \mid \mathcal{K}' \in TR_{\alpha}^{\land} (\mathcal{K}) \} \]
\[ TR_{\langle \mathcal{K} \cdot \alpha \| \{ \alpha \} \rangle}^{\land} (\mathcal{K}) = \{ \mathcal{K}_1 \land \mathcal{K}' \mid \mathcal{K}' \in TR_{\alpha_1, \alpha}^{\land} (\mathcal{K}) \} \cup \{ \neg \mathcal{K}_1 \land \mathcal{K}' \mid \mathcal{K}' \in TR_{\alpha_2, \alpha}^{\land} (\mathcal{K}) \} \]

Similarly as earlier, \(|TR_{\alpha}^{\land} (\mathcal{K})|\) may be exponential but each \(\mathcal{K}' \in TR_{\alpha}^{\land} (\mathcal{K})\) is polynomial and can be built in polynomial time. Next we show the following claim:

(\(\dagger\)) For every \(I\) and every \(\mathcal{K}\), we have that \(I \models TR_{\alpha} (\mathcal{K})\) iff there exists some \(\mathcal{K}' \in TR_{\alpha}^{\land} (\mathcal{K})\) such that \(I \models \mathcal{K}'\).

With (\(\dagger\)), we can easily show that \(\sigma(\mathcal{K}_{\text{pre}}) \land TR_{\alpha_1, \ldots, \alpha_m} (\sigma(\mathcal{K}))\) is finitely satisfiable iff there exists some \(\mathcal{K}' \in TR_{\alpha_1, \ldots, \alpha_m} (\sigma(\mathcal{K}))\) such that \(\sigma(\mathcal{K}_{\text{pre}}) \land \mathcal{K}'\) is finitely satisfiable. For the “only if” direction, assume that \(\sigma(\mathcal{K}_{\text{pre}}) \land TR_{\alpha_1, \ldots, \alpha_m} (\sigma(\mathcal{K}))\) is finitely satisfiable. Then there exists some finite \(\bar{I}\) such that \(\bar{I} \models \sigma(\mathcal{K}_{\text{pre}})\) and \(\bar{I} \models TR_{\alpha_1, \ldots, \alpha_m} (\sigma(\mathcal{K}))\). By (\(\dagger\)), for this \(\bar{I}\) we have that \(\bar{I} \models TR_{\alpha_1, \ldots, \alpha_m} (\sigma(\mathcal{K}))\) iff there is some \(\mathcal{K}' \in TR_{\alpha_1, \ldots, \alpha_m} (\sigma(\mathcal{K}))\) such that \(\bar{I} \models \mathcal{K}'\). We choose this \(\mathcal{K}'\). It follows that \(\bar{I} \models \mathcal{K}'\), and since \(\bar{I} \models \sigma(\mathcal{K}_{\text{pre}})\), we can conclude that \(\sigma(\mathcal{K}_{\text{pre}}) \land \mathcal{K}'\) is finitely satisfiable. For the other direction, let \(\mathcal{K}' \in TR_{\alpha_1, \ldots, \alpha_m} (\sigma(\mathcal{K}))\) be such that \(\sigma(\mathcal{K}_{\text{pre}}) \land \mathcal{K}'\) is finitely satisfiable. Then there exists some finite \(\bar{I}\) such that \(\bar{I} \models \sigma(\mathcal{K}_{\text{pre}})\) and \(\bar{I} \models \mathcal{K}'\). By (\(\dagger\)), for this \(\bar{I}\) it follows that \(\bar{I} \models TR_{\alpha} (\sigma(\mathcal{K}))\), and since \(\bar{I} \models \sigma(\mathcal{K}_{\text{pre}})\), we have that \(\sigma(\mathcal{K}_{\text{pre}}) \land TR_{\alpha_1, \ldots, \alpha_m} (\sigma(\mathcal{K}))\) is finitely satisfiable. Having shown this, the upper bound follows directly from the complexity of deciding finite satisfiability of \(\sigma(\mathcal{K}_{\text{pre}}) \land \mathcal{K}'\), and the fact that \(\mathcal{K}'\) is of polynomial size and can be obtained nondeterministically in polynomial time.

It is only left to show (\(\ddagger\)), 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 \(TR_{\alpha}^{\land} (\mathcal{K}) = \{ \mathcal{K} \} \) and \(TR_{\alpha} (\mathcal{K}) = \mathcal{K}\), so we can set \(\mathcal{K}' = \mathcal{K}\) and the claim follows.

For the cases of \(\alpha' = (A \circ C) \cdot \alpha\), \(\alpha' = (A \circ C) \cdot \alpha\), \(\alpha' = (p \circ r) \cdot \alpha\), and \(\alpha' = (p \circ r) \cdot \alpha\), we can proceed exactly as in the proof of Theorem 5.4. The remaining case is when \(\alpha' = (\mathcal{K}_1 \cdot \alpha_1 \| \alpha_2) \cdot \alpha\). For the “only if” direction, assume that \(I \models TR_{\alpha'} (\mathcal{K})\). In this case, the choice of \(\mathcal{K}'\) depends on \(I\).

We distinguish two cases:

- If \(I \models \mathcal{K}_1\), since \(TR_{\alpha'} (\mathcal{K}) = (\neg \mathcal{K}_1 \lor TR_{\alpha_1, \alpha} (\mathcal{K})) \land (\mathcal{K}_1 \lor TR_{\alpha_2, \alpha} (\mathcal{K}))\), we have that \(I \models TR_{\alpha_1, \alpha} (\mathcal{K})\). By induction hypothesis, there exists \(\mathcal{K}'' \in TR_{\alpha_1, \alpha}^{\land} (\mathcal{K})\) such that \(I \models \mathcal{K}''\). We set \(\mathcal{K}' = \mathcal{K}_1 \land \mathcal{K}''\). By induction hypothesis, \(\mathcal{K}' \in TR_{\alpha_2, \alpha}^{\land} (\mathcal{K})\) by definition, the claim follows.

- Otherwise, if \(I \not\models \mathcal{K}_1\), we have that \(I \not\models TR_{\alpha_1, \alpha} (\mathcal{K})\). By induction hypothesis, there exists \(\mathcal{K}'' \in TR_{\alpha_1, \alpha}^{\land} (\mathcal{K})\) such that \(I \models \mathcal{K}''\). Set \(\mathcal{K}' = \neg \mathcal{K}_1 \land \mathcal{K}''\) and since we have again that \(\mathcal{K}' \in TR_{\alpha}^{\land} (\mathcal{K})\) by definition, the claim follows.

For the “if” direction, assume that there exists some \(\mathcal{K}' \in TR_{\alpha}^{\land} (\mathcal{K})\) such that \(I \models \mathcal{K}'\). We distinguish again two cases:

- If \(\mathcal{K}' = \mathcal{K}_1 \land \mathcal{K}''\), for some \(\mathcal{K}'' \in TR_{\alpha}^{\land} (\mathcal{K})\), then \(I \models \mathcal{K}_1\), and by induction hypothesis, \(I \models TR_{\alpha_1, \alpha} (\mathcal{K})\). Hence, \(I \models (\neg \mathcal{K}_1 \lor TR_{\alpha_1, \alpha} (\mathcal{K})) \land (\mathcal{K}_1 \lor TR_{\alpha_2, \alpha} (\mathcal{K}))\), and since \(TR_{\alpha} (\mathcal{K}) = (\neg \mathcal{K}_1 \lor TR_{\alpha_1, \alpha} (\mathcal{K})) \land (\mathcal{K}_1 \lor TR_{\alpha_2, \alpha} (\mathcal{K}))\), the claim follows.

- Otherwise, if \(\mathcal{K}' = \neg \mathcal{K}_1 \land \mathcal{K}''\), for some \(\mathcal{K}'' \in TR_{\alpha}^{\land} (\mathcal{K})\), then \(I \models \neg \mathcal{K}_1\), and by induction hypothesis, \(I \models TR_{\alpha_2, \alpha} (\mathcal{K})\). Hence, \(I \models (\neg \mathcal{K}_1 \lor TR_{\alpha_1, \alpha} (\mathcal{K})) \land (\mathcal{K}_1 \lor TR_{\alpha_2, \alpha} (\mathcal{K}))\), and again the claim follows.

This concludes the proof of the upper bound for \(\mathcal{ALCHOIQbr}\).

For the upper bound for \(\mathcal{ALCHOIQbr}\), one can build in exponential time all possible sequences of actions of length \(\ell\), and for each such sequence of actions there are exponentially many substitutions \(\sigma\) (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 earlier.

For the NP upper bound for $DL$-$Lite^+_R$ 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 $DL$-$Lite^+_R$ 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, as we only know that it satisfies a given precondition but we 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 precondition.

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

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

$$\mathcal{K}_{\text{pre}} = x : \text{Empl} \land y : \text{ActivePrj} \land x : \geq 2 \text{ worksFor.} \top$$

$$\mathcal{K}_2 = x : \text{Empl} \land \neg (y : \text{ActivePrj}).$$

The sequence $\sigma(\alpha_3)$ is a plan for $\mathcal{K}_2$ for every substitution $\sigma$ and for every input instance $I$ with $I \models \sigma(\mathcal{K}_{\text{pre}})$. Indeed, the precondition $\mathcal{K}_{\text{pre}}$ ensures that every such input instance $I$ contains an active project $\sigma(y)$ and an employee $\sigma(x)$ that works for at least two projects. Then $\sigma(\alpha_3)$ removes $\sigma(y)$ from the active projects and removes all employees working only for $\sigma(y)$; the precondition 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 precondition. 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 formulas $\mathcal{K}_{\text{pre}}$ 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 $I$ with $I \models \sigma(\mathcal{K}_{\text{pre}})$, for every possible substitution $\sigma$?

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

We observe that although similar in appearance, there is a crucial difference between problems (P2) and (S), as the former asks for the existence of a plan for some substitution from some input instance, whereas the latter asks for the existence of a “universal” plan that reaches the goal for every substitution and for every instance that satisfies the precondition. 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 $\mathcal{K}_{\text{pre}}$ 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\(_R\) KBs and simple actions.
—Problems (C) and (Sb) are coNExpTime-complete for ALCHOIQ\(_{br}\) and ExpTime-complete for ALCHOIQ\(_{br}\).
—If \(\mathcal{K}\) and \(\mathcal{K}'\) are expressed in DL-Lite\(_R^+\) and all actions in Act are simple, then (C) is coNP-complete and (Sb) is NP\(^{NP}\)-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 ALCHOIQ\(_{br}\) and ALCHOIQ\(_{br}\), respectively.

For the upper bounds, we first observe that (C) reduces to validity testing in ALCHOIQ\(_{br}\) and ALCHOIQ\(_{br}\), respectively: an instance of (C) (as described previously) is positive iff the formula \(\sigma(\neg\mathcal{K}'_{\text{pre}}) \lor \text{TR}_{a_1 \ldots a_n}(\sigma(\mathcal{K}'))\) is valid, where \(\mathcal{K}'_{\text{pre}}\) and \(\mathcal{K}'\) are respectively obtained from \(\mathcal{K}_{\text{pre}}\) and \(\mathcal{K}\) by replacing every variable by a fresh individual. Deciding validity of \(\sigma(\neg\mathcal{K}'_{\text{pre}}) \lor \text{TR}_{a_1 \ldots a_n}(\sigma(\mathcal{K}'))\) in turn reduces to deciding whether \(\sigma(\mathcal{K}'_{\text{pre}}) \land \neg\text{TR}_{a_1 \ldots a_n}(\sigma(\mathcal{K}'))\) 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 \(\mathcal{K}' \land \neg\text{TR}_o(\mathcal{K})\) shown in the proofs of Theorems 5.4 and 5.9.

Negative instances of (Sb), where \(\mathcal{K}_{\text{pre}}\) is the precondition and \(\mathcal{K}\) is the goal, can be recognized in NExpTime for ALCHOIQ\(_{br}\) and in ExpTime for ALCHOIQ\(_{br}\). Such a test comprises building an exponentially large set of all candidate action sequences of length at most \(\ell\) and then making sure that each candidate is invalidated. In other words, each candidate action sequence \(P\) induces an instance of (C), which can be shown negative in NExpTime for ALCHOIQ\(_{br}\) and ExpTime for ALCHOIQ\(_{br}\). In the case of DL-Lite\(_R^+\) and simple actions, we can guess nondeterministically a sequence of actions of length at most \(\ell\) and then check that the induced instance of (C) is positive, which is a test in coNP.

To show that the NP\(^{NP}\) upper bound is tight, we use a polynomial-time reduction from evaluating quantified Boolean formulas of the form \(\gamma = \exists p_1 \ldots \exists p_n \forall q_1 \ldots \forall q_m \psi\), where \(\psi\) is a Boolean combination over propositional variables \(V = \{p_1, \ldots, p_n, q_1, \ldots, q_m\}\). We can assume that negation in \(\psi\) 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 \(\mathcal{K}_{\text{pre}} = (\land_{1 \leq i \leq n} o_{p_i} : \neg(T \lor F)) \land (\land_{1 \leq i \leq m} o_{q_i} : (T \lor F) \cap (\neg T \lor \neg F))\). 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:

\[
\alpha_i = (o_{p_i} : \neg F) \land (T \lor o_{p_i}), \quad \alpha'_i = (o_{p_i} : \neg T) \land (F \lor o_{p_i}).
\]

We finally let \(\ell = n\) and let \(\mathcal{K}\) be the KB obtained from \(\psi\) 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 \(\gamma\) evaluates to \(\text{true}\) 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 and deletions, and that allow one to introduce fresh values (De Giacomo et al. 2012; Bagheri Hariri et al. 2013a). Most works view the database modification as an encapsulated process defined declaratively in terms of pre- and postconditions (see the aforementioned works and references therein). In practice, however, the actual data manipulation that leads the database from a state where the precondition holds to one ensuring the postcondition 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 $\mu$-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, whereas 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 $C^2$-based formalism. Additionally, 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. In other words, they do not guarantee inertia, and one instead faces the frame problem and needs to explicitly enumerate all tuples that should remain unchanged. In this sense, and despite the syntactic similarity, their actions are more in the declarative pre-/postcondition style described previously.

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 $SROIQ^\sigma$ (an extension of the standard DL $SROIQ$ with substitutions, used to express weakest preconditions). 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 postconditions, and tackle the problem of verifying a Hoare triple $\{\varphi_{pre}\}\alpha\{\varphi_{post}\}$ (which is defined analogously to our static verification but allows 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 article 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 KBs, and logic theories in general, is a classic topic in knowledge representation, discussed extensively in the literature (see (Fagin et al. 1986; Katsuno and Mendelzon 1991). The updates described by our action language are similar in spirit to the KB updates studied in other works, particularly 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) KBs, 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 nondeterministic. A central problem considered is the projection problem, which consists of 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 nondeterministic 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, preconditions, 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 GSD 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 that 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 that 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 ALCHOIQbr 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. For example, extensions of DL-LiteR with role functionality would be very useful in the context of GSD.

In this article, we have used DLs for describing constraints on GSD, data states, and queries in basic actions. As explained in Section 2.3, all of our results hold if ALCHOIQbr is replaced by C2. However, we chose to use DLs instead of C2, 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. It is our hope that 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, such as 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, such as 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 ALCHOIQbr, e.g., 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, toward 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.

A APPENDIX

We first show that KBs in ALCHOIQbr can be seen as KBs in a standard DL and that many of the added features are only syntactic sugar. Let ALCHOIQ∪\{U,D\} be the extension of ALCHOIQ with the role constructors of union and difference. In other words, ALCHOIQ∪\{U,D\} is defined as
Table 3. Standard Translation From $\mathcal{ALCHOIQ}_{br}$ to $C^2$

| Translating concepts | Translating roles | Translating assertions, inclusions, and KBs |
|----------------------|------------------|------------------------------------------|
| $ST_x(\tau) = (x = x)$ | $ST_{xy}(p) = \hat{p}(x, y)$ | $ST(o:C) = ST_x(C)[x \mapsto \hat{o}]$ |
| $ST_x(\bot) = \neg(x = x)$ | $ST_{xy}(\{o, o'\}) = (x = \hat{o}) \wedge (y = \hat{o'})$ | $ST((o_1, o_2):r) = ST_{xy}(p)[x \mapsto \hat{o}_1, y \mapsto \hat{o}_2]$ |
| $ST_x(A) = \hat{A}(x)$ | $ST_{xy}(r^-) = ST_{yx}(r) \wedge \neg ST_{xy}(r)$ | $ST(K_1 \wedge K_2) = ST(K_1) \wedge ST(K_2)$ |
| $ST_x(\neg C) = \neg ST_x(C)$ | $ST_{xy}(r_1 \cup r_2) = ST_{xy}(r_1) \cup ST_{xy}(r_2)$ | $ST(\neg K) = \neg ST(K)$ |
| $ST_x(C_1 \cap C_2) = ST_x(C_1) \wedge ST_x(C_2)$ | $ST_{xy}(r_1 \cap r_2) = ST_{xy}(r_1) \wedge ST_{xy}(r_2)$ | $ST(C_1 \subseteq C_2) = \forall x. (ST_x(C_1) \rightarrow ST_x(C_2))$ |
| $ST_x(C_1 \cup C_2) = ST_x(C_1) \lor ST_x(C_2)$ | $ST_{xy}(r | C) = ST_{xy}(r) \wedge ST_x(C)$ | $ST((r_1 \in r_2) = \forall x y. (ST_{xy}(r_1) \rightarrow ST_{xy}(r_2))$ |
| $ST_{xy}(3r.C) = \exists y. (ST_{xy}(r) \wedge ST_y(C))$ | $ST_{yx}(r^-) = ST_{yx}(r) \wedge \neg ST_{xy}(r)$ | $ST(K_1 \lor K_2) = ST(K_1) \lor ST(K_2)$ |
| $ST_{xy}(4r.C) = \forall y. (ST_{xy}(r) \rightarrow ST_y(C))$ | $ST_{xy}(r_1 | C) = ST_{xy}(r_1) \wedge ST_x(C)$ | $ST((r_1 \in r_2) = \forall x y. (ST_{xy}(r_1) \rightarrow ST_{xy}(r_2))$ |
| $ST_{xy}(n r.C) = \exists n^xy. (ST_{xy}(r) \wedge ST_y(C))$ | $ST_{yx}(x) = ST_{yx}(x) \wedge \neg ST_{xy}(x)$ | $ST(C_1 \subseteq C_2) = \forall x. (ST_x(C_1) \rightarrow ST_x(C_2))$ |
| $ST_{xy}(\geq n r.C) = \exists n^xy. (ST_{xy}(r) \wedge ST_y(C))$ | $ST_{yx}(r_1 | C) = ST_{yx}(r_1) \wedge ST_x(C)$ | $ST((r_1 \in r_2) = \forall x y. (ST_{xy}(r_1) \rightarrow ST_{xy}(r_2))$ |

Note: $C, C_1, C_2$ are ordinary concepts, $r, r_1, r_2$ ordinary roles, $K, K_1, K_2$ KBs, $A \in N_C, p \in N_R, \{o, o_1, o_2\} \subseteq N_i$.

in the left column of Table 1, except roles $r$ are defined as follows:

$r \rightarrow p \mid r^- \mid r_1 \cup r_2 \mid r_1 \setminus r_2$.

**Proposition A.1.** For every $\mathcal{ALCHOIQ}_{br}$ KB $\mathcal{K}$, there is an $\mathcal{ALCHOIQ}^{\bot \cup}$ 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 $\mathcal{ALCHOIQ}^{\bot \cup}$ KB $\mathcal{K}'$. We show this as follows:

ACM Transactions on Computational Logic, Vol. 18, No. 4, Article 27. Publication date: November 2017.
— Singleton role \{\langle o_1, o_2 \rangle \}: This constructor can be simulated in \( \mathcal{K}' \) by introducing a fresh role name \( p_{o_1,o_2} \), replacing every occurrence of \( \{\langle o_1, o_2 \rangle \} \) in \( \mathcal{K} \) with \( p_{o_1,o_2} \), and adding to \( \mathcal{K}' \) the following three inclusions:
\[
\{ o_1 \} \sqsubseteq \exists p_{o_1,o_2}.\top, \quad \exists p_{o_1,o_2}.\top \sqsubseteq \{ o_1 \}, \quad \exists p_{o_1,o_2}.\top \sqsubseteq \{ o_2 \}.
\]
One can verify that every model \( I \) of \( \mathcal{K} \) can be transformed into a model of \( \mathcal{K}' \) by additionally interpreting the role name \( p_{o_1,o_2} \) as \( p_{o_1,o_2}' = \{\langle o_1', o_2' \rangle \} = \{\langle o_1, o_2 \rangle \}^I \). In addition, 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 \( r|_C \) and \( r|_C \): First we note that by using role inverse, each of the two constructs can be expressed in terms of the other one. Specifically, \( r|_C \) can be equivalently replaced with \((r^\bot)|_C\). Hence, we can assume that \( \mathcal{K} \) contains no occurrences of \( r|_C \). To simulate \( r|_C \) in \( \mathcal{K}' \), we introduce two fresh role names \( r_C \) and \( r_{C'} \), substitute every occurrence of \( r|_C \) in \( \mathcal{K} \) with \( r_C \), and add to \( \mathcal{K}' \) the following inclusions:
\[
r \sqsubseteq r_C \cup r_{C'}, \quad r_C \cup r_{C'} \sqsubseteq r, \quad \exists ! r_C.\top \sqsubseteq C, \quad \exists ! r_{C'}.\top \sqsubseteq \neg C.
\]
One can verify that every model \( I \) of \( \mathcal{K} \) can be transformed into a model of \( \mathcal{K}' \) by additionally interpreting the role name \( r_C \) as \( r^C_I = r|_C^I \) and the role name and \( r_{C'} \) as \( r_{C'}^I = r^\bot \setminus r|_C^I \). In addition, 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 \( a_1, \ldots, a_n \) be all concept and role inclusions that occur in \( \mathcal{K} \) (note that the same \( a_i \) can occur in \( \mathcal{K} \) several times, possibly negated). For each \( a_i \), let \( A_i \) be a new concept name. We define \( \text{Conc}(\mathcal{K}) \) as the \( \text{ALCHOIQ} \) concept obtained from \( \mathcal{K} \) by replacing each \( a_i \) by \( A_i \), each \( \lor \) by \( \sqcup \), each \( \land \) by \( \sqcap \), and each \( \neg \) by \( \neg \). The \( \text{ALCHOIQ}^{1,\bot,\sqcup,\sqcap,\neg} \) 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:
\[
\{ o \} \sqsubseteq \text{Conc}(\mathcal{K}), \quad \top \sqsubseteq \exists ! s.\top \setminus \{ o \}, \quad \\
\{ o \} \sqcap A_i \sqsubseteq \forall s.(\neg C_1 \sqcup C_2), \quad \text{for all } a_i = C_1 \sqsubseteq C_2, \quad \\
\{ o \} \sqcap \neg A_i \sqsubseteq \forall s.(C_1 \sqcap \neg C_2), \quad \text{for all } a_i = C_1 \sqsubseteq C_2, \quad \\
\{ o \} \sqcap A_i \sqsubseteq \forall s.(\exists (r_1 \setminus r_2).\top), \quad \text{for all } a_i = r_1 \sqsubseteq r_2.
\]
Note that the fresh role \( s \) connects \( o \) with every element of the domain. The underlying intuition of the preceding encoding is the following: whenever an inclusion \( a_i \) in \( \mathcal{K} \) is set to true, \( o \) satisfies the corresponding \( A_i \) in \( \mathcal{K}' \), and the role \( s \) ensures that every element of the domain satisfies the inclusion \( a_i \). Conversely, if the inclusion \( a_i \) in \( \mathcal{K} \) is set to false (i.e., \( \neg a_i \) is set to true), \( o \) 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 \( a_i \).

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

— For all \( a_i = C_1 \sqsubseteq C_2, \) set
\[
A_i^I = \{ o^I \}, \quad \text{if } I \models a_i, \quad \text{and} \quad A_i^I = \emptyset, \quad \text{if } I \not\models a_i.
\]
Table 4. Additional Constructors in \( \mathcal{ALBOQ}^id \)

| Syntax     | Semantics                                                                 | Translation to \( C^2 \)              |
|------------|---------------------------------------------------------------------------|---------------------------------------|
| Role negation | \( \neg r \) = \((\Delta_I^I \times \Delta_I^I) \setminus r^I \)    | \( ST_{xy}(\neg r) = \neg ST_{xy}(r) \) |
| Identity role | \( id \) = \{(e,e) | e \in \Delta_I^I \}   | \( ST_{xy}(id(C)) = ST_x(C) \land x = y \) |

**Definable constructors**

| Syntax | Definition                                   |
|--------|---------------------------------------------|
| Top role | \( \nabla \equiv p \cup \neg p \) for some role name \( p \) |
| Bottom role | \( \Delta \equiv \neg \nabla \) |
| Left cylindrification | \( D^c \equiv \nabla |_D \) |
| Right cylindrification | \( \nabla \downarrow |_D \) |
| Cross product | \( C \times D \equiv C^c \cap \nabla \downarrow |_D \) |
| Test | \( C? \equiv \text{id} |_D \) |

**Concept constructors**

- For all \( \alpha_i = r_1 \subseteq r_2 \), set
  \[
  A_i^I' = \Delta_I^I, \quad \text{if } I \models \alpha_i, \text{ and} \\
  A_i^I' = \emptyset, \quad \text{if } I \not\models \alpha_i. 
  \]
- \( s^I' = \{o^I\} \times \Delta_I^I \).
- \( I' \) coincides with \( I \) in the interpretation of all other individual names, concept names, and role names.

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

A.1 \( \text{ALCHQI}OhQbr \) and \( C^2 \)

The so-called **standard translation** from DLs to \( C^2 \) can be found in several works (e.g., Borgida (1996), Schmidt and Tishkovsky (2014), and Hustadt et al. (2004)) and is summarized in Table 3.

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 a free variable and \( ST_y \) into a formula with \( y \) as a free variable. Note that since the translation is to \( C^2 \), 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 formulas with two free variables. Again, we have two analogous translation functions: \( ST_{xy} \) produces a \( C^2 \) formula with the pair \( (x,y) \) as free variables, whereas \( ST_{yx} \) produces a \( C^2 \) formula with the pair \( (y,x) \).

Each assertion, inclusion, and knowledge base (KB) \( \alpha \) is translated into a \( C^2 \) sentence \( ST(\alpha) \). If \( \alpha \) is an assertion, then \( ST(\alpha) \) is obtained by substituting free variables with constants. For 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 upper bounds in this article would still hold if we use as our constraint language a more expressive DL that is still captured by $C^2$ (or even $C^2$ itself). This means that we can add to $\mathcal{ALCHOIQbr}$ (any combination of) the constructors in the upper part of Table 4, as they are all expressible in $C^2$. In particular, our results hold for $\mathcal{ALBOQ}^d$, which adds number restrictions to $\mathcal{ALBO}^d$. $\mathcal{ALBO}^d$ 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 4. 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), and Schmidt and Tishkovsky (2014)) and references therein.

REFERENCES

Serge Abiteboul and Paris Kanellakis. 1989. Object identity as a query language primitive. In Proceedings of the ACM SIGMOD International Conference on Management of Data. 159–173.

Serge Abiteboul and Victor Vianu. 1999 Regular path queries with constraints. Journal of Computer and System Sciences 58, 3, 428–452. DOI: http://dx.doi.org/10.1006/jcss.1999.1627

Shigiponja Ahmetaj, Diego Calvanese, Magdalena Ortiz, and Mantas Šimkus. 2014. Managing change in graph-structured data using description logics. In Proceedings of the 28th AAAI Conference on Artificial Intelligence (AAAI). 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, 39–56. DOI: http://dx.doi.org/10.4230/DagRep.6.4.39

Alessandro Artale. 2006. Reasoning on temporal class diagrams: Undecidability results. Annals of Mathematics and Artificial Intelligence 46, 3, 265–288.

Alessandro Artale, Diego Calvanese, Roman Kontchakov, Vladislav Ryzhikov, and Michael Zakharyaschev. 2007. Reasoning over extended ER models. In Conceptual Modeling—ER 2007. 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 Proceedings of the 8th International Symposium on Frontiers of Combining Systems (FroCoS’11). 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, Cambridge, England.

Franz Baader, Silvio Ghilardi, and Carsten Lutz. 2012. LTL over description logic axioms. ACM Transactions on Computational Logic 13, 3, 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 Logic for Programming, Artificial Intelligence, and Reasoning. 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 Proceedings of the 20th National Conference on Artificial Intelligence (AAAI’05). 572–577.

Franz Baader and Benjamin Zarrieß. 2013. Verification of Golog programs over description logic actions. In Frontiers of Combining Systems. 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 Proceedings of the 32nd ACM SIGACT SIGMOD SIGSAL Symposium on Principles of Database Systems (PODS’13). 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. Journal of Artificial Intelligence Research 46, 651–686.

Mira Balaban and Azzam Maraee. 2013. Finite satisfiability of UML class diagrams with constrained class hierarchy. ACM Transactions on Software Engineering and Methodology 22, 3, 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, 70–118.

Anthony J. Bonner and Michael Kifer. 1994. An overview of transaction logic. Theoretical Computer Science 133, 2, 205–265.

Alex Borgida. 1996. On the relative expressiveness of description logics and predicate logics. Artificial Intelligence 82, 1-2, 353–367.

T. Bray, J. Paoli, and C. M. Sperberg-McQueen. 1998. Extensible Markup Language (XML) 1.0: 10-February-1998. World Wide Web Consortium. Available at http://www.w3.org/TR/1998/REC-xml-19980210
Jon Hael Brenas, Rachid Echahed, and Martin Strecker. 2014. A Hoare-like calculus using the SROIQ\textsuperscript{S} logic on transformations of graphs. In *Theoretical Computer Science*. Lecture Notes in Computer Science, Vol. 8705. Springer, 164–178. DOI: http://dx.doi.org/10.1007/978-3-662-44602-7_14

D. 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, 165–204.

Diego Calvanese. 1996. Finite model reasoning in description logics. In *Proceedings of the 5th International Conference on the Principles of Knowledge Representation and Reasoning (KR’96)*, 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. *Journal of Automated Reasoning* 39, 3, 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*, G. Lakemeyer and S. A. McIlraith (Eds.). College Publications, London, England, 1–16.

Diego Calvanese, Giuseppe De Giacomo, and Marco Montali. 2013a. Foundations of data-aware process analysis: A database theory perspective. In *Proceedings of the 32nd ACM SIGACT SIGMOD SIGAI Symposium on Principles of Database Systems (PODS’13)*, 1–12.

Diego Calvanese, Thomas Eiter, and Magdalena Ortiz. 2009. Regular path queries in expressive description logics with nominals. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI’09)*, 714–720.

Diego Calvanese, Wolfgang Fischl, Reinhard Pichler, Emanuel Sallinger, and Mantas Simkus. 2014. Capturing relational schemas and functional dependencies in RDFS. In *Proceedings of the 28th AAAI Conference on Artificial Intelligence (AAAI’14)*, 1003–1011.

Diego Calvanese, Maurizio Lenzerini, and Daniele Nardi. 1999. Unifying class-based representation formalisms. *Journal of Artificial Intelligence Research* 11, 199–240.

Diego Calvanese, Magdalena Ortiz, and Mantas Simkus. 2013b. Evolving graph databases under description logic constraints. In *Informal Proceedings of the 26th International Workshop on Description Logics*. CEUR Workshop Proceedings, Vol. 1014. CEUR-WS, 120–131.

Diego Calvanese, Magdalena Ortiz, and Mantas Simkus. 2016. Verification of evolving graph-structured data under expressive path constraints. In *Proceedings of the 19th International Conference on Database Theory (ICDT’16)*. Leibniz International Proceedings in Informatics (IPIPCs), Vol. 48. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 15:1–15:19. DOI: http://dx.doi.org/10.4230/LIPIcs.ICDT.2016.15

Mariano P. Consens and Alberto O. Mendelzon. 1990. GraphLog: A visual formalism for real life recursion. In *Proceedings of the 9th ACM SIGACT SIGMOD SIGART Symposium on Principles of Database Systems (PODS’90)*, 404–416.

Giuseppe De Giacomo, Riccardo De Masellis, and Riccardo Rosati. 2012. Verification of conjunctive artifact-centric services. *International Journal of Cooperative Information Systems* 21, 2, 111–140.

Alin Deutsch, Richard Hull, and Victor Vianu. 2014. Automatic verification of database-centric systems. *ACM SIGMOD Record* 43, 3, 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, Greenwich, CT, 1–18.

Yolanda Gil. 2005. Description logics and planning. *AI Magazine* 26, 2, 73–84.

Gösta Grahne and Alex Thomo. 2003. Query containment and rewriting using views for regular path queries under constraints. In *Proceedings of the 22nd ACM SIGACT SIGMOD SIGART Symposium on Principles of Database Systems (PODS’03)*, 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. *ACM Computing Surveys* 19, 3, 201–260.

Ullrich Hustadt, Renate A. Schmidt, and Lilie Georgieva. 2004. A survey of decidable first-order fragments and description logics. *Journal on Relational Methods in Computer Science* 1, 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. 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 *Proceedings of the 2nd International Conference on the Principles of Knowledge Representation and Reasoning (KR’91)*, 387–394.

Yevgeny Kazakov. 2004. A polynomial translation from the two-variable guarded fragment with number restrictions to the guarded fragment. In *Logics in Artificial Intelligence*. 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. *Journal of Computer and System Sciences* 79, 6, 835–872.
Robert A. Kowalski, Fariba Sadri, and Paul Soper. 1987. Integrity checking in deductive databases. In *Proceedings of the 13th International Conference on Very Large Data Bases (VLDB’87)*, 61–69.

Maurizio Lenzerini. 2011. Ontology-based data management. In *Proceedings of the 20th International Conference on Information and Knowledge Management (CIKM’11)*, 5–6.

Hector J. Levesque. 1984. Foundations of a functional approach to knowledge representation. *Artificial Intelligence* 23, 155–212.

H. J. Levesque, R. Reiter, Y. Lesperance, F. Lin, and R. Scherl. 1997. GOLOG: A logic programming language for dynamic domains. *Journal of Logic Programming* 31, 59–84.

Hongkai Liu, Carsten Lutz, Maja Milicic, and Frank Wolter. 2006. Reasoning about actions using description logics with general TBoxes. In *Logics in Artificial Intelligence*. Lecture Notes in Computer Science, Vol. 4160. Springer, 266–279. DOI: [http://dx.doi.org/10.1007/11853886_23](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, 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, 132–171.

Carsten Lutz, Ulrike Sattler, and Frank Wolter. 2001. Description logics and the two-variable fragment. In *Proceedings of the 14th International Workshop on Description Logics (DL’01)*. CEUR Electronic Workshop Proceedings, Vol. 49. CEUR-WS, 66–75. [http://CEUR-WS.org/Vol-49/LutzSattlerWolter-66start.ps](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* 21 1, 135–140.

Pablo Muñoz, Nils Vortmeier, and Thomas Zeume. 2016. Dynamic graph queries. In *Proceedings of the 19th International Conference on Database Theory (ICDT’16)*. Leibniz International Proceedings in Informatics (LIPIcs), Vol. 48. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 14:1–14:18.

Ian Pratt-Hartmann. 2005. Complexity of the two-variable fragment with counting quantifiers. *Journal of Logic, Language and Information* 14, 3, 369–395.

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

Klaus Schild. 1991. A correspondence theory for terminological logics: Preliminary report. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence (IJCAI’91)*. 466–471.

Renate A. Schmidt and Dmitry Tishkovsky. 2014. Using Tableau to decide description logics with full role negation and identity. *ACM Transactions on Computational Logic* 15, 1, 7:1–7:31.

Tim Sheard and David Stemple. 1989. Automatic verification of database transaction safety. *ACM Transactions on Database Systems* 14, 3, 322–368.

David Spelt and Herman Balsters. 1998. Automatic verification of transactions on an object-oriented database. In *Database Programming Languages*. Lecture Notes in Computer Science, Vol. 1369. Springer, 396–412.

Balder ten Cate and Massimo Franceschet. 2005. Guarded fragments with constants. *Journal of Logic, Language and Information* 14, 3, 281–288.

Stephan Tobies. 2000. The complexity of reasoning with cardinality restrictions and nominals in expressive description logics. *Journal of Artificial Intelligence Research* 12, 199–217.

Moshe Y. Vardi. 1995. On the complexity of bounded-variable queries (extended abstract). In *Proceedings of the 14th ACM SIGACT SIGMOD SIGART Symposium on Principles of Database Systems (PODS’95)*. 266–276. DOI: [http://dx.doi.org/10.1145/212433.212474](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.

Received January 2017; revised August 2017; accepted August 2017