Actions over Core-Closed Knowledge Bases

Claudia Cauli[1,2(✉)], Magdalena Ortiz[3], and Nir Piterman[1]

[1] University of Gothenburg, Gothenburg, Sweden
claudiacauli@gmail.com
[2] Amazon Web Services, Seattle, USA
[3] TU Wien, Vienna, Austria

Abstract. We present new results on the application of semantic- and knowledge-based reasoning techniques to the analysis of cloud deployments. In particular, to the security of *Infrastructure as Code* configuration files, encoded as description logic knowledge bases. We introduce an action language to model *mutating actions*; that is, actions that change the structural configuration of a given deployment by adding, modifying, or deleting resources. We mainly focus on two problems: the problem of determining whether the execution of an action, no matter the parameters passed to it, will not cause the violation of some security requirement (*static verification*), and the problem of finding sequences of actions that would lead the deployment to a state where (un)desirable properties are (not) satisfied (*plan existence* and *plan synthesis*). For all these problems, we provide definitions, complexity results, and decision procedures.

1 Introduction

The use of automated reasoning techniques to analyze the properties of cloud infrastructure is gaining increasing attention [4–7,18]. Despite that, more effort needs to be put into the modeling and verification of generic security requirements over cloud infrastructure pre-deployment. The availability of formal techniques, providing strong security guarantees, would assist complex system-level analyses such as threat modeling and data flow, which now require considerable time, manual intervention, and expert domain knowledge.

We continue our research on the application of semantic-based and knowledge-based reasoning techniques to cloud deployment *Infrastructure as Code* configuration files. In [14], we reported on our experience using expressive description logics to model and reason about Amazon Web Services' proprietary Infrastructure as Code framework (AWS CloudFormation). We used the rich constructs of these logics to encode domain knowledge, simulate closed-world reasoning, and express mitigations and exposures to security threats. Due to the high complexity of basic tasks [3,26], we found reasoning in such a framework to be not efficient at cloud scale. In [15], we introduced *core-closed knowledge*

C. Cauli—This work was done prior to joining Amazon.

J. Blanchette et al. (Eds.): IJCAR 2022, LNAI 13385, pp. 281–299, 2022.
https://doi.org/10.1007/978-3-031-10769-6_17

bases—a lightweight description logic combining closed- and open-world reasoning that is tailored to model cloud infrastructure and efficiently query its security properties. Core-closed knowledge bases enable partially-closed predicates whose interpretation is closed over a *core* part of the knowledge base but open elsewhere. To encode potential exposure to security threats, we studied the query satisfiability problem and (together with the usual query entailment problem) applied it to a new class of conjunctive queries that we called Must/May queries. We were able to answer such queries over core-closed knowledge bases in LogSpace in data complexity and NP in combined complexity, improving the required NExptime complexity for satisfiability over $\mathcal{ALCOIQ}$ (used in [14]).

Here, we enhance the quality of the analyses done over pre-deployment artifacts, giving users and practitioners additional precise insights on the impact of potential changes, fixes, and general improvements to their cloud projects. We enrich core-closed knowledge bases with the notion of *core-completeness*, which is needed to ensure that updates are consistent. We define the syntax and semantics of an action language that is expressive enough to encode *mutating* API calls, i.e., operations that change a cloud deployment configuration by creating, modifying, or deleting existing resources. As part of our effort to improve the quality of automated analysis, we also provide relevant reasoning tools to identify and predict the consequences of these changes. To this end, we consider procedures that determine whether the execution of a mutating action always preserves given properties (*static verification*); determine whether there exists a sequence of operations that would lead a deployment to a configuration meeting certain requirements (*plan existence*); and find such sequences of operations (*plan synthesis*).

The paper is organized as follows. In Sect. 2, we provide background on core-closed knowledge bases, conjunctive queries, and Must/May queries. In Sect. 3, we motivate and introduce the notion of *core-completeness*. In Sect. 4, we define the action language. In Sect. 5, we describe the static verification problem and characterize its complexity. In Sect. 6, we address the planning problem and concentrate on the synthesis of minimal plans satisfying a given requirement expressed using Must/May queries. We discuss related works in Sect. 7 and conclude in Sect. 8. Results and proofs that are omitted in this paper are found in the full version [16].

2 Background

Description logics (DLs) are a family of logics for encoding knowledge in terms of concepts, roles, and individuals; analogous to first-order logic unary predicates, binary predicates, and constants, respectively. Standard DL knowledge bases (KBs) have a set of axioms, called *TBox*, and a set of assertions, called *ABox*. The TBox contains axioms that relate to concepts and roles. The ABox contains assertions that relate individuals to concepts and pairs of individuals to roles. KBs are usually interpreted under the open-world assumption, meaning that the asserted facts are not assumed to be complete.

Core-Closed Knowledge Bases. In [15], we introduced core-closed knowledge bases (ccKBs) as a suitable description logic formalism to encode cloud deployments. The main characteristic of ccKBs is to allow for a combination of open- and closed-world reasoning that ensures tractability. A DL-Lite$^{\mathcal{F}}$ ccKB is the tuple $\mathcal{K} = \langle \mathcal{T}, \mathcal{A}, \mathcal{S}, \mathcal{M} \rangle$ built from the standard knowledge base $\langle \mathcal{T}, \mathcal{A} \rangle$ and the *core* system $\langle \mathcal{S}, \mathcal{M} \rangle$. The former encodes incomplete terminological and assertional knowledge. The latter is, in turn, composed of two parts: $\mathcal{S}$ (also called the *SBox*), containing axioms that encode the core structural specifications, and $\mathcal{M}$ (also called the *MBox*), containing positive concept and role assertions that encode the core configuration. Syntactically, $\mathcal{M}$ is similar to an ABox but, semantically, is assumed to be complete with respect to the specifications in $\mathcal{S}$.

The ccKB $\mathcal{K}$ is defined over the alphabets $\mathbf{C}$ (of concepts), $\mathbf{R}$ (of roles), and $\mathbf{I}$ (of individuals), all partitioned into an open subset and a partially-closed subset. That is, the set of concepts is partitioned into the open concepts $\mathbf{C}^{\mathcal{K}}$ and the closed (specification) concepts $\mathbf{C}^{\mathcal{S}}$; the set of roles is partitioned into open roles $\mathbf{R}^{\mathcal{K}}$ and closed (specification) roles $\mathbf{R}^{\mathcal{S}}$; and the set of individuals is partitioned into open individuals $\mathbf{I}^{\mathcal{K}}$ and closed (model) individuals $\mathbf{I}^{\mathcal{M}}$. We call $\mathbf{C}^{\mathcal{S}}$ and $\mathbf{R}^{\mathcal{S}}$ core-closed predicates, or partially-closed predicates, as their extension is closed over the core domain $\mathbf{I}^{\mathcal{M}}$ and open otherwise. In contrast, we call $\mathbf{C}^{\mathcal{K}}$ and $\mathbf{R}^{\mathcal{K}}$ open predicates. The syntax of concept and role expressions in DL-Lite$^{\mathcal{F}}$ [2,8] is as follows:

$$\mathsf{B} ::= \bot \mid \mathsf{A} \mid \exists \mathsf{p}$$

where A denotes a concept name and p is either a role name r or its inverse r^-. The syntax of axioms provides for the three following axioms:

$$\mathsf{B}^1 \sqsubseteq \mathsf{B}^2, \qquad \mathsf{B}^1 \sqsubseteq \neg \mathsf{B}^2, \qquad (\mathsf{funct}\ \mathsf{p}),$$

respectively called: *positive inclusion* axioms, *negative inclusion* axioms, and *functionality* axioms. These axioms are contained in the sets $\mathcal{S}$ and $\mathcal{T}$. To precisely denote the subsets of $\mathcal{S}$ and $\mathcal{T}$ having only axioms of a given type we use the notation $PI_{\mathcal{X}}$, $NI_{\mathcal{X}}$, and $F_{\mathcal{X}}$, for $\mathcal{X} \in \{\mathcal{S}, \mathcal{T}\}$, which respectively contain only positive inclusion axioms, negative inclusion axioms, and functionality axioms. From now on, we denote symbols from the alphabet $\mathbf{X}^{\mathcal{X}}$ with the subscript $\mathcal{X}$, and symbols from the generic alphabet $\mathbf{X}$ with no subscript. In core-closed knowledge bases, axioms and assertions fall into the scope of a different set depending on the predicates and individuals that they refer to, according to the set definitions below.

$$\begin{aligned}
\mathcal{M} &\subseteq \{\mathsf{A}_{\mathcal{S}}(a_{\mathcal{M}}),\ \mathsf{R}_{\mathcal{S}}(a_{\mathcal{M}}, a),\ \mathsf{R}_{\mathcal{S}}(a, a_{\mathcal{M}})\} \\
\mathcal{A} &\subseteq \{\mathsf{A}_{\mathcal{K}}(a_{\mathcal{K}}),\ \mathsf{R}_{\mathcal{K}}(a_{\mathcal{K}}, b_{\mathcal{K}}),\ \mathsf{A}_{\mathcal{S}}(a_{\mathcal{K}}),\ \mathsf{R}_{\mathcal{S}}(a_{\mathcal{K}}, b_{\mathcal{K}})\} \\
\mathcal{S} &\subseteq \{\mathsf{B}^1_{\mathcal{S}} \sqsubseteq \mathsf{B}^2_{\mathcal{S}},\ \mathsf{B}^1_{\mathcal{S}} \sqsubseteq \neg \mathsf{B}^2_{\mathcal{S}},\ \mathsf{Func}(\mathsf{P}_{\mathcal{S}})\} \\
\mathcal{T} &\subseteq \{\mathsf{B}^1 \sqsubseteq \mathsf{B}^2_{\mathcal{K}},\ \mathsf{B}^1 \sqsubseteq \neg \mathsf{B}^2_{\mathcal{K}},\ \mathsf{Func}(\mathsf{P}_{\mathcal{K}})\}
\end{aligned}$$

In the above definition of the set $\mathcal{M}$, role assertions link at least one individual from the core domain $\mathbf{I}^{\mathcal{M}}$ (denoted as $a_{\mathcal{M}}$) to one individual from the general set

$\mathbf{I}$ (denoted as a). Node a could either be an individual from the open partition $\mathbf{I}^{\mathcal{K}}$ or the closed partition $\mathbf{I}^{\mathcal{M}}$. When a is an element from the set $\mathbf{I}^{\mathcal{K}}$, we refer to it as a "boundary node", as it sits at the boundary between the core and the open parts of the knowledge base. As mentioned earlier, $\mathcal{M}$-assertions are assumed to be complete and consistent with respect to the terminological knowledge given in $\mathcal{S}$; whereas the usual open-world assumption is made for $\mathcal{A}$-assertions. The semantics of a DL-Lite$^{\mathcal{F}}$ core-closed KB is given in terms of interpretations $\mathcal{I}$, consisting of a non-empty domain $\Delta^{\mathcal{I}}$ and an interpretation function $\cdot^{\mathcal{I}}$. The latter assigns to each concept A a subset $\mathsf{A}^{\mathcal{I}}$ of $\Delta^{\mathcal{I}}$, to each role r a subset $\mathsf{r}^{\mathcal{I}}$ of $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$, and to each individual a a node $a^{\mathcal{I}}$ in $\Delta^{\mathcal{I}}$, and it is extended to concept expressions in the usual way. An interpretation $\mathcal{I}$ is a model of an inclusion axiom $\mathsf{B}_1 \sqsubseteq \mathsf{B}_2$ if $\mathsf{B}_1^{\mathcal{I}} \subseteq \mathsf{B}_2^{\mathcal{I}}$. An interpretation $\mathcal{I}$ is a model of a membership assertion $\mathsf{A}(a)$, (resp. $\mathsf{r}(a, b)$) if $a^{\mathcal{I}} \in \mathsf{A}^{\mathcal{I}}$ (resp. $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in \mathsf{r}^{\mathcal{I}}$). We say that $\mathcal{I}$ models $\mathcal{T}$, $\mathcal{S}$, and $\mathcal{A}$ if it models all axioms or assertions contained therein. We say that $\mathcal{I}$ models $\mathcal{M}$, denoted $\mathcal{I} \models^{\mathsf{CWA}} \mathcal{M}$, when it models an $\mathcal{M}$-assertion f *if and only if* $f \in \mathcal{M}$. Finally, $\mathcal{I}$ models $\mathcal{K}$ if it models $\mathcal{T}$, $\mathcal{S}$, $\mathcal{A}$, and $\mathcal{M}$. When $\mathcal{K}$ has at least one model, we say that $\mathcal{K}$ is satisfiable.

In the remainder of this paper, we will sometimes refer to the *lts* interpretation of $\mathcal{M}$. The *lts* interpretation of $\mathcal{M}$, denoted $lts(\mathcal{M})$, is the interpretation $(\Delta^{lts(\mathcal{M})}, \cdot^{lts(\mathcal{M})})$ defined only over concept and role names from the set $\mathbf{C}^{\mathcal{S}}$ and $\mathbf{R}^{\mathcal{S}}$, respectively, and over individual names from $\mathbf{I}^{\mathcal{K}}$ that appear in the scope of $\mathcal{M}$-assertions. The interpretation $lts(\mathcal{M})$ is the *unique* model of $\mathcal{M}$ such that $lts(\mathcal{M}) \models^{\mathsf{CWA}} \mathcal{M}$.

In the application presented in [14], description logic KBs are used to encode machine-readable deployment files containing multiple resource declarations. Every resource declaration has an underlying tree structure, whose leaves can potentially link to the roots of other resource declarations. Let $\mathbf{I}^r \subseteq \mathbf{I}^{\mathcal{M}}$ be the set of all resource nodes, we encode their resource declarations in $\mathcal{M}$, and formalize the resulting forest structure by partitioning $\mathcal{M}$ into multiple subsets $\{\mathcal{M}_i\}_{i \in \mathbf{I}^r}$, each representing a tree of assertions rooted at a resource node i (we generally refer to constants in $\mathcal{M}$ as nodes). For the purpose of this work, we will refer to core-closed knowledge bases where $\mathcal{M}$ is partitioned as described; that is, ccKBs such that $\mathcal{K} = \langle \mathcal{T}, \mathcal{A}, \mathcal{S}, \{\mathcal{M}_i\}_{i \in \mathbf{I}^r} \rangle$.

Conjunctive Queries. A *conjunctive query* (CQ) is an existentially-quantified formula $q[\vec{x}]$ of the form $\exists \vec{y}.conj(\vec{x}, \vec{y})$, where *conj* is a conjunction of positive atoms and potentially inequalities. A *union of conjunctive queries* (UCQ) is a disjunction of CQs. The variables in $\vec{x}$ are called *answer variables*, those in $\vec{y}$ are the existentially-quantified *query variables*. A tuple $\vec{c}$ of constants appearing in the knowledge base $\mathcal{K}$ is an answer to q if for all interpretations $\mathcal{I}$ model of $\mathcal{K}$ we have $\mathcal{I} \models q[\vec{c}]$. We call these tuples the *certain answers* of q over $\mathcal{K}$, denoted $ans(\mathcal{K}, q)$, and the problem of testing whether a tuple is a certain answer *query entailment*. A tuple $\vec{c}$ of constants appearing in $\mathcal{K}$ satisfies q if there exists an interpretation $\mathcal{I}$ model of $\mathcal{K}$ such that $\mathcal{I} \models q[\vec{c}]$. We call these tuples the *sat answers* of q over $\mathcal{K}$, denoted $sat{-}ans(\mathcal{K}, q)$, and the problem of testing whether a given tuple is a sat answer *query satisfiability*.

MUST/MAY *Queries.* A MUST/MAY query ψ [15] is a Boolean combination of nested UCQs in the scope of a MUST or a MAY operator as follows:

$$\psi ::= \neg\psi \mid \psi_1 \wedge \psi_2 \mid \psi_1 \vee \psi_2 \mid \text{MUST } \varphi \mid \text{MAY } \varphi_{\not\approx}$$

where φ and $\varphi_{\not\approx}$ are unions of conjunctive queries potentially containing inequalities. The reasoning needed for answering the nested queries can be decoupled from the reasoning needed to answer the higher-level formula: nested queries MUST φ are reduced to conjunctive query entailment, and nested queries MAY $\varphi_{\not\approx}$ are reduced to conjunctive query satisfiability. We denote by $\mathsf{ANS}(\psi, \mathcal{K})$ the answers of a MUST/MAY query ψ over the core-closed knowledge base $\mathcal{K}$.

3 Core-Complete Knowledge Bases

The algorithm Consistent presented in [15] computes satisfiability of DL-Lite$^{\mathcal{F}}$ core-closed knowledge bases relying on the assumption that $\mathcal{M}$ is complete and consistent with respect to $\mathcal{S}$. Such an assumption effectively means that the information contained in $\mathcal{M}$ is *explicitly* present and *cannot be completed by inference.* The algorithm relies on the existence of a theoretical object, the canonical interpretation, in which missing assertions can always be introduced when they are logically implied by the positive inclusion axioms. As a matter of fact, positive inclusion axioms are not even included in the inconsistency formula built for the satisfiability check, as it is proven that the canonical interpretation always satisfies them ([15], Lemma 3). When the assumption that $\mathcal{M}$ is consistent with respect to $\mathcal{S}$ is dropped, the algorithm Consistent becomes insufficient to check satisfiability. We illustrate this with an example.

Example 1 (Required Configuration). Let us consider the axioms constraining the AWS resource type S3::Bucket. In particular, the $\mathcal{S}$-axiom S3::Bucket $\sqsubseteq$ $\exists$loggingConfiguration prescribing that all buckets must have a *required* logging configuration. For a set $\mathcal{M} = \{\mathsf{S3::Bucket}(b)\}$, according to the partially-closed semantics of core-closed knowledge bases, the absence of an assertion $\mathsf{loggingConfiguration}(b, x)$, for some x, is interpreted as the assertion being false in $\mathcal{M}$, which is therefore not consistent with respect to $\mathcal{S}$. However, the algorithm Consistent will check the *lts* interpretation of $\mathcal{M}$ for an empty formula (as there are no negative inclusion or functionality axioms) and return *true.*

In essence, the algorithm Consistent does not compute the full satisfiability of the whole core-closed knowledge base, but only of its open part. Satisfiability of $\mathcal{M}$ with respect to the positive inclusion axioms in $\mathcal{S}$ needs to be checked separately. We introduce a new notion to denote when a set $\mathcal{M}$ is complete with respect to $\mathcal{S}$ that is distinct from the notion of consistency. Let $\mathcal{K} = \langle \mathcal{T}, \mathcal{A}, \mathcal{S}, \mathcal{M} \rangle$ be a DL-Lite$^{\mathcal{F}}$ core-closed knowledge base; we say that $\mathcal{K}$ is *core-complete* when $\mathcal{M}$ models *all* positive inclusion axioms in $\mathcal{S}$ under a closed-world assumption; we say that $\mathcal{K}$ is *open-consistent* when $\mathcal{M}$ and $\mathcal{A}$ model all negative inclusion and functionality axioms in $\mathcal{K}$'s negative inclusion closure. Finally, we say that $\mathcal{K}$ is *fully satisfiable* when is both *core-complete* and *open-consistent.*

Lemma 1. *In order to check* full satisfiability *of a DL-Lite$^{\mathcal{F}}$ core-closed KB, one simply needs to check if $\mathcal{K}$ is* core-complete *(that is, if $\mathcal{M}$ models all* positive axioms *in $\mathcal{S}$ under a closed-world assumption) and if $\mathcal{K}$ is* open-consistent *(that is, to run the algorithm* Consistent*).*

Proof. Dropping the assumption that $\mathcal{M}$ is consistent w.r.t. $\mathcal{S}$ causes Lemma 3 from [15] to fail. In particular, the canonical interpretation of $\mathcal{K}$, $can(\mathcal{K})$, would still be a model of $PI_{\mathcal{T}}$, $\mathcal{A}$, and $\mathcal{M}$, but may *not* be a model of $PI_{\mathcal{S}}$. This is due to the construction of the canonical model that is based on the notion of applicable axioms. In rules **c5-c8** of [15] Definition 1, axioms in $PI_{\mathcal{S}}$ are defined as applicable to assertions involving open nodes $a_{\mathcal{K}}$ but *not* to model nodes $a_{\mathcal{M}}$ in $\mathbf{I}^{\mathcal{M}}$. As a result, if the implications of such axioms on model nodes are not included in $\mathcal{M}$ itself, then they will not be included in $can(\mathcal{K})$ either, and $can(\mathcal{K})$ will not be a model of $PI_{\mathcal{S}}$. On the other hand, one can easily verify that Lemmas 1,2,4,5,6,7 and Corollary 1 would still hold as they do not rely on the assumption. However, since it is not guaranteed anymore that $\mathcal{M}$ satisfies all positive inclusion axioms from $\mathcal{S}$, the *if* direction of [15] Theorem 1 does not hold anymore: there can be an unsatisfiable ccKB $\mathcal{K}$ such that $db(\mathcal{A}) \cup lts(\mathcal{M}) \models cln(\mathcal{T} \cup \mathcal{S}), \mathcal{A}, \mathcal{M}$. For instance, the knowledge base from Example 1. We also note that the negative inclusion and functionality axioms from $\mathcal{S}$ will be checked anyway by the consistency formula, both on $db(\mathcal{A})$ and on $lts(\mathcal{M})$.

Lemma 2. *Checking whether a DL-Lite$^{\mathcal{F}}$ core-closed knowledge base is* core-complete *can be done in polynomial time in $\mathcal{M}$. As a consequence, checking full satisfiability is also done in polynomial time in $\mathcal{M}$.*

Proof. One can write an algorithm that checks *core-completeness* by searching for the existence of a positive inclusion axiom $\mathsf{B}^1_{\mathcal{S}} \sqsubseteq \mathsf{B}^2_{\mathcal{S}} \in PI_{\mathcal{S}}$ such that $\mathcal{M} \models \mathsf{B}^1_{\mathcal{S}}(a_{\mathcal{M}})$ and $\mathcal{M} \not\models \mathsf{B}^2_{\mathcal{S}}(a_{\mathcal{M}})$, where the relation $\models$ is defined over DL-Lite$^{\mathcal{F}}$ concept expressions as follows:

$$\begin{aligned}
\mathcal{M} \models & \bot(a_{\mathcal{M}}) && \leftrightarrow && \mathit{false} \\
\mathcal{M} \models & \mathsf{A}_{\mathcal{S}}(a_{\mathcal{M}}) && \leftrightarrow && \mathsf{A}_{\mathcal{S}}(a_{\mathcal{M}}) \in \mathcal{M} \\
\mathcal{M} \models & \exists \mathsf{r}_{\mathcal{S}}(a_{\mathcal{M}}) && \leftrightarrow && \exists b.\ \mathsf{r}_{\mathcal{S}}(a_{\mathcal{M}}, b) \in \mathcal{M} \\
\mathcal{M} \models & \exists \mathsf{r}^-_{\mathcal{S}}(a_{\mathcal{M}}) && \leftrightarrow && \exists b.\ \mathsf{r}_{\mathcal{S}}(b, a_{\mathcal{M}}) \in \mathcal{M}.
\end{aligned}$$

The knowledge base is *core-complete* if such a node cannot be found.

4 Actions

We now introduce a formal language to encode mutating actions. Let us remind ourselves that, in our application of interest, the execution of a mutating action modifies the configuration of a deployment by either adding new resource instances, deleting existing ones, or modifying their settings. Here, we introduce a framework for DL-Lite$^{\mathcal{F}}$ core-closed knowledge base updates, triggered by the execution of an action that enables all the above mentioned effects. The

only component of the core-closed knowledge base that is modified by the action execution is $\mathcal{M}$; while $\mathcal{T}$, $\mathcal{S}$, and $\mathcal{A}$ remain unchanged. As a consequence of updating $\mathcal{M}$, actions can introduce new individuals and delete old ones, thus updating the set $\mathbf{I}^{\mathcal{M}}$ as well. Note that this may force changes outside $\mathbf{I}^{\mathcal{M}}$ due to the axioms in $\mathcal{T}$ and $\mathcal{S}$. The effects of applying an action over $\mathcal{M}$ depend on a set of input parameters that will be instantiated at execution time, resulting in different assertions being added or removed from $\mathcal{M}$. As a consequence of assertions being added, fresh individuals might be introduced in the active domain of $\mathcal{M}$, including both model nodes from $\mathbf{I}^{\mathcal{M}}$ and boundary nodes from $\mathbf{I}^{B}$. Differently, as a consequence of assertions being removed, individuals might be removed from the active domain of $\mathcal{M}$, including model nodes from $\mathbf{I}^{\mathcal{M}}$ but *not* including boundary nodes from $\mathbf{I}^{B}$. In fact, boundary nodes are owned by the open portion of the knowledge base and are known to exist regardless of them being used in $\mathcal{M}$. We invite the reader to review the set definitions for $\mathcal{A}$- and $\mathcal{M}$-assertions (Sect. 2) to note that it is indeed possible for a generic boundary individual a involved in an $\mathcal{M}$-assertion to also be involved in an $\mathcal{A}$-assertion.

4.1 Syntax

An action is defined by a signature and a body. The signature consists of an action name and a list of formal parameters, which will be replaced with actual parameters at execution time. The body, or action effect, can include conditional statements and concatenation of atomic operations over $\mathcal{M}$-assertions. For example, let α be the action $act(\vec{x}) = \gamma$; that is, the action denoted by signature $act(\vec{x})$ and body γ, with signature name act, signature parameters $\vec{x}$, and body effect γ. Since it contains unbound parameters, or free variables, action α is ungrounded and needs to be instantiated with actual values in order to be executed over a set $\mathcal{M}$. In the following, we assume the existence of a set Var, of variable names, and consider a generic input parameters substitution $\vec{\theta} : \mathsf{Var} \to \mathbf{I}$, which replaces each variable name by an individual node. For simplicity, we will denote an ungrounded action by its effect γ, and a grounded action by the composition of its effect with an input parameter substitution $\gamma\vec{\theta}$. Action effects can either be *complex* or *basic*. The syntax of complex action effects γ and basic effects β is constrained by the following grammar.

$$\gamma ::= \epsilon \mid \beta \cdot \gamma \mid [\varphi \rightsquigarrow \beta] \cdot \gamma$$
$$\beta ::= \oplus_x S \mid \ominus_x S \mid \odot_{x_{new}} S \mid \ominus_x$$

The complex action effects γ include: the empty effect (ϵ), the execution of a basic effect followed by a complex one ($\beta \cdot \gamma$), and the conditional execution of a basic effect upon evaluation of a formula φ over the set $\mathcal{M}$ ($[\varphi \rightsquigarrow \beta] \cdot \gamma$). The basic action effects β include: the addition of a set S of $\mathcal{M}$-assertions to the subset $\mathcal{M}_x$ ($\oplus_x S$), the removal of a set S of $\mathcal{M}$-assertions from the subset $\mathcal{M}_x$ ($\ominus_x S$), the addition of a fresh subset $\mathcal{M}_{x_{new}}$ containing all the $\mathcal{M}$-assertions in the set S ($\odot_{x_{new}} S$), and the removal of an existing $\mathcal{M}_x$ subset in its entirety ($\ominus_x$). The set S, the formula φ, and the operators $\oplus/\ominus$ might contain *free*

variables. These variables are of two types: *(1)* variables that are replaced by the grounding of the action input parameters, and *(2)* variables that are the answer variables of the formula φ and appear in the nested effect β.

Example 2. The following is the definition of the action createBucket from the API reference of the AWS resource type S3::Bucket. The input parameters are two: the new bucket name "*name*" and the canned access control list "*acl*" (one of *Private*, *PublicRead*, *PublicReadWrite*, *AuthenticatedRead*, etc.). The effect of the action is to add a fresh subset $\mathcal{M}_x$ for the newly introduced individual x containing the two assertions $\mathsf{S3::Bucket}(x)$ and $\mathsf{accessControl}(x, y)$.

$$\mathsf{createBucket}(x : name, y : acl) = \odot_x\{\mathsf{S3::Bucket}(x), \mathsf{accessControl}(x, y)\} \cdot \epsilon$$

The action needs to be instantiated by a specific parameter assignment, for example the substitution $\theta = [\ x \leftarrow \mathit{DataBucket},\ y \leftarrow \mathit{Private}\ ]$, which binds the variable x to the node *DataBucket* and the variable y to the node *Private*, both taken from a pool of inactive nodes in $\mathbf{I}$.

Action Query φ. The syntax introduced in the previous paragraph allows for complex actions that conditionally execute a basic effect β depending on the evaluation of a formula φ over $\mathcal{M}$. This is done via the construct $[\,\varphi \rightsquigarrow \beta\,] \cdot \gamma$. The formula φ might have a set $\vec{y}$ of answer variables that appear free in its body and are then bound to concrete tuples of nodes during evaluation. The answer tuples are in turn used to instantiate the free variables in the nested effect β. We call φ the *action query* since we use it to select all the nodes that will be involved in the action effect. According to the grammar below, φ is a boolean combination of $\mathcal{M}$-assertions potentially containing free variables.

$$\varphi ::= \mathsf{A}_{\mathcal{S}}(t) \ \mid\ \mathsf{R}_{\mathcal{S}}(t_1, t_2) \mid\ \varphi_1 \wedge \varphi_2 \ \mid\ \varphi_2 \vee \varphi_2 \ \mid\ \neg\varphi$$

In particular, $\mathsf{A}_{\mathcal{S}}$ is a symbol from the set $\mathbf{C}^{\mathcal{S}}$ of partially-closed concepts; $\mathsf{R}_{\mathcal{S}}$ is a symbol from the set $\mathbf{R}^{\mathcal{S}}$ of partially-closed roles; and t, t_1, t_2 are either individual or variable names from the set $\mathbf{I} \uplus \mathsf{Var}$, chosen in such a way that the resulting assertion is an $\mathcal{M}$-assertion. Since the formula φ can only refer to $\mathcal{M}$-assertions, which are interpreted under a closed semantics, its evaluation requires looking at the content of the set $\mathcal{M}$. A formula φ with no free variables is a boolean formula and evaluates to either true or false. A formula φ with answer variables $\vec{y}$ and arity $ar(\varphi)$ evaluates to all the tuples $\vec{t}$, of size equal the arity of φ, that make the formula true in $\mathcal{M}$. The free variables of φ can only appear in the action β such that $\varphi \rightsquigarrow \beta$. We denote by $\mathsf{ANS}(\varphi, \mathcal{M})$ the set of answers to the action query φ over $\mathcal{M}$. It is easy to see that the maximum number of tuples that could be returned by the evaluation (that is, the size of the set $\mathsf{ANS}(\varphi, \mathcal{M})$) is bounded by $|\mathbf{I}^{\mathcal{M}} \uplus \mathbf{I}^{B}|^{ar(\varphi)}$, in turn bounded by $(\,2|\mathcal{M}|\,)^{2|\varphi|}$.

Example 3. The following example shows the encoding of the S3 API operation called deleteBucketEncryption, which requires as unique input parameter the name of the bucket whose encryption configuration is to be deleted. Since

a bucket can have multiple encryption configuration rules (each prescribing different encryption keys and algorithms to be used) we use an action query φ to select *all* the nodes that match the assertions structure to be removed.

$$\varphi[y,k,z](x) = \mathsf{S3::Bucket}(x) \wedge \mathsf{encrRule}(x,y) \wedge \mathsf{SSEKey}(y,k) \wedge \mathsf{SSEAlgo}(y,z)$$

The query φ is instantiated by the specific bucket instance (which will replace the variable x) and returns all the triples (y,k,z) of encryption rule, key, and algorithm, respectively, which identify the assertions corresponding to the different encryption configurations that the bucket has. The answer variables are then used in the action effect to instantiate the assertions to remove from $\mathcal{M}_x$:

$$\begin{aligned}&\mathsf{deleteBucketEncryption}(x : name)\\ &= [\varphi[y,k,z](x) \;\rightsquigarrow\; \ominus_x\{\mathsf{encrRule}(x,y), \mathsf{SSEKey}(y,k), \mathsf{SSEAlgo}(y,z)\}] \cdot \epsilon\end{aligned}$$

4.2 Semantics

So far, we have described the syntax of our action language and provided two examples that showcase the encoding of real-world API calls. Now, we define the semantics of action effects with respect to the changes that they induce over a knowledge base. Let us recall that given a substitution $\vec{\theta}$ for the input parameters of an action γ, we denote by $\gamma\vec{\theta}$ the grounded action where all the input variables are replaced according to what prescribed by $\vec{\theta}$. Let us also recall that the effects of an action apply only to assertions in $\mathcal{M}$ and individuals from $\mathbf{I}^{\mathcal{M}}$, and cannot affect nodes and assertions from the open portion of the knowledge base.

The execution of a grounded action $\gamma\vec{\theta}$ over a DL-Lite$^{\mathcal{F}}$ core-closed knowledge base $\mathcal{K} = (\mathcal{T}, \mathcal{A}, \mathcal{S}, \mathcal{M})$, defined over the set $\mathbf{I}^{\mathcal{M}}$ of partially-closed individuals, generates a new knowledge base $\mathcal{K}^{\gamma\vec{\theta}} = (\mathcal{T}, \mathcal{A}, \mathcal{S}, \mathcal{M}^{\gamma\vec{\theta}})$, defined over an updated set of partially-closed individuals $\mathbf{I}^{\mathcal{M}^{\gamma\vec{\theta}}}$. Let S be a set of $\mathcal{M}$-assertions, γ a complex action, $\vec{\theta}$ an input parameter substitution, and $\vec{\rho}$ a generic substitution that potentially replaces all free variables in the action γ. Let $\vec{\rho}_1$ and $\vec{\rho}_2$ be two substitutions with signature $\mathsf{Var} \to \mathbf{I}$ such that $dom(\vec{\rho}_1) \cap dom(\vec{\rho}_2) = \emptyset$; we denote their composition by $\vec{\rho}_1\vec{\rho}_2$ and define it as the new substitution such that $\vec{\rho}_1\vec{\rho}_2(x) = a$ if $\vec{\rho}_1(x) = a \vee \vec{\rho}_2(x) = a$, and $\vec{\rho}_1\vec{\rho}_2(x) = \bot$ if $\vec{\rho}_1(x) = \bot \wedge \vec{\rho}_2(x) = \bot$. We formalize the application of the grounded action $\gamma\vec{\theta}$ as the transformation $T_{\gamma\vec{\theta}}$ that maps the pair $\langle \mathcal{M}, \mathbf{I}^{\mathcal{M}} \rangle$ into the new pair $\left\langle \mathcal{M}', \mathbf{I}^{\mathcal{M}'} \right\rangle$. We sometimes use the notation $T_{\gamma\vec{\theta}}(\mathcal{M})$ or $T_{\gamma\vec{\theta}}(\mathbf{I}^{\mathcal{M}})$ to refer to the updated MBox or to the updated set of model nodes, respectively. The rules for applying the transformation depend on the structure of the action γ and are reported in Fig. 1. The transformation starts with an initial generic substitution $\vec{\rho} = \vec{\theta}$. As the transformation progresses, the generic substitution $\vec{\rho}$ can be updated only as a result of the evaluation of an action query φ over $\mathcal{M}$. Precisely, all the tuples $\vec{t_1}, ..., \vec{t_n}$ making φ true in $\mathcal{M}$ will be considered and composed with the current substitution $\vec{\rho}$ generating n fresh substitutions $\rho\vec{t}_1, ..., \rho\vec{t}_n$ which are used in the subsequent application of the nested effect β. Since the core $\mathcal{M}$ of the knowledge base $\mathcal{K}$ changes at every

action execution, its domain of model nodes $\mathbf{I}^{\mathcal{M}}$ changes as well. The execution of an action $\gamma\vec{\theta}$ over the knowledge base $\mathcal{K} = (\mathcal{T}, \mathcal{A}, \mathcal{S}, \mathcal{M})$ with set of model nodes $\mathbf{I}^{\mathcal{M}}$ could generate a new $\mathcal{K}^{\gamma\vec{\theta}} = (\mathcal{T}, \mathcal{A}, \mathcal{S}, \mathcal{M}^{\gamma\vec{\theta}})$ with a new set of model nodes $\mathbf{I}^{\mathcal{M}'}$ that is not *core-complete* or not *open-consistent* (see Sect. 3 for the corresponding definitions). We illustrate two examples next.

$$
\begin{aligned}
T_{\epsilon\vec{\rho}}(\mathcal{M}, \mathbf{I}^{\mathcal{M}}) &= (\mathcal{M}, \mathbf{I}^{\mathcal{M}}) \\
T_{\beta\cdot\gamma\vec{\rho}}(\mathcal{M}, \mathbf{I}^{\mathcal{M}}) &= T_{\gamma\vec{\rho}}(\ T_{\beta\vec{\rho}}(\mathcal{M}, \mathbf{I}^{\mathcal{M}})\ ) \\
T_{[\varphi \rightsquigarrow \beta]\cdot\gamma\vec{\rho}}(\mathcal{M}, \mathbf{I}^{\mathcal{M}}) &= \begin{cases} T_{\gamma\vec{\rho}}(T_{\beta\vec{\rho}}(\mathcal{M}, \mathbf{I}^{\mathcal{M}})\ ) & \text{if } \mathsf{ANS}(\varphi, \mathcal{M}) = tt \\ T_{\gamma\vec{\rho}}(\mathcal{M}, \mathbf{I}^{\mathcal{M}}) & \text{if } \mathsf{ANS}(\varphi, \mathcal{M}) = \emptyset \text{ or } ff \\ T_{\gamma\vec{\rho}}(T_{\beta\vec{\rho}\vec{t}_1 \cdots \beta\vec{\rho}\vec{t}_n}(\mathcal{M}, \mathbf{I}^{\mathcal{M}})) & \text{if } \mathsf{ANS}(\varphi, \mathcal{M}) = \{\vec{t}_1, .., \vec{t}_n\} \end{cases} \\
T_{\oplus_x S\vec{\rho}}(\mathcal{M}, \mathbf{I}^{\mathcal{M}}) &= (\ \{\mathcal{M}_i\}_{i \neq \vec{\rho}(x)} \cup \{\mathcal{M}_{\vec{\rho}(x)} \cup S_{\vec{\rho}}\}\ ,\ \mathbf{I}^{\mathcal{M}} \cup ind(S_{\vec{\rho}})\ ) \\
T_{\ominus_x S\vec{\rho}}(\mathcal{M}, \mathbf{I}^{\mathcal{M}}) &= (\ \{\mathcal{M}_i\}_{i \neq \vec{\rho}(x)} \cup \{\mathcal{M}_{\vec{\rho}(x)} \setminus S_{\vec{\rho}}\}\ ,\ \mathbf{I}^{\mathcal{M}} \setminus ind(S_{\vec{\rho}})\ ) \\
T_{\odot_x S\vec{\rho}}(\mathcal{M}, \mathbf{I}^{\mathcal{M}}) &= (\ \mathcal{M} \cup \{\mathcal{M}_{\vec{\rho}(x)} = S_{\vec{\rho}}\}\ ,\ \mathbf{I}^{\mathcal{M}} \cup ind(S_{\vec{\rho}})\ ) \\
T_{\ominus_x \vec{\rho}}(\mathcal{M}, \mathbf{I}^{\mathcal{M}}) &= (\ \mathcal{M} \setminus \mathcal{M}_{\vec{\rho}(x)}\ ,\ \mathbf{I}^{\mathcal{M}} \setminus ind(\mathcal{M}_{\vec{\rho}(x)})\ )
\end{aligned}
$$

Fig. 1. Semantic of the action language defined over the MBox $\mathcal{M}$ and set $\mathbf{I}^{\mathcal{M}}$.

Example 4 (Violation of core-completeness). Consider the case where the general specifications of the system require all objects of type bucket to have a logging configuration, and an action that removes the logging configuration from a bucket. Consider the core-closed knowledge base $\mathcal{K}$ where $\mathcal{S} = \{\mathsf{S3{::}Bucket} \sqsubseteq \exists\mathsf{loggingConfiguration}\}$ and $\mathcal{M} = \{\mathsf{S3{::}Bucket}(b), \mathsf{loggingConfiguration}(b, c)\}$ (consistent wrt $\mathcal{S}$) and the action γ defined as

$$
\begin{aligned}
&\mathsf{deleteLoggingConfiguration}(x : name) \\
&\quad = [(\varphi[y](x) = \mathsf{S3{::}Bucket}(x) \wedge \mathsf{loggingConfiguration}(x, y)) \\
&\qquad \rightsquigarrow \ominus_x\{\mathsf{loggingConfiguration}(x, y)\}] \cdot \epsilon
\end{aligned}
$$

For the input parameter substitution $\vec{\theta} = [x \leftarrow b]$, it is easy to see that the transformation $T_{\gamma\vec{\theta}}$ applied to $\mathcal{M}$ results in the update $\mathcal{M}^{\gamma\vec{\theta}} = \{\mathsf{S3{::}Bucket}(b)\}$, which is *not* core-complete.

Example 5 (Violation of open-consistency). Consider the case where an action application indirectly affects boundary nodes and their properties, leading to inconsistencies in the open portion of the knowledge base. For example, when the knowledge base prescribes that buckets used to store logs cannot be public; however, a change in the configuration of a bucket instance causes a second bucket (initially known to be public) to also become a log store. In particular, this happens when the knowledge base $\mathcal{K}$ contains the $\mathcal{T}$-axiom $\exists\mathsf{loggingDestination}^- \sqsubseteq \neg\mathsf{PublicBucket}$ and the $\mathcal{A}$-assertion $\mathsf{PublicBucket}(b)$, and

we apply an action that introduces a new bucket storing its logs to b, defined as follows:

$$\begin{aligned}&\mathsf{createBucketWithLogging}(x : name, y : log)\\ &\quad = \odot_x\{\mathsf{S3{::}Bucket}(x), \mathsf{loggingDestination}(x, y)\}\end{aligned}$$

For the input parameter substitution $\vec{\theta} = [x \leftarrow newBucket, y \leftarrow b]$, the result of applying the transformation $T_{\gamma\vec{\theta}}$ is the set $\mathcal{M} = \{\mathsf{S3{::}Bucket}(newBucket), \mathsf{loggingDestination}(newBucket, b)\}$ which, combined with the pre-existing and unchanged sets $\mathcal{T}$ and $\mathcal{A}$, causes the updated $\mathcal{K}^{\gamma\vec{\theta}}$ to be *not* open-consistent.

From a practical point of view, the examples highlight the need to re-evaluate core-completeness and open-consistency of a core-closed knowledge base after each action execution. Detecting a violation to core-completeness signals that we have modeled an action that is inconsistent with respect to the systems specifications, which most likely means that the action is missing something and needs to be revised. Detecting a violation to open-consistency signals that our action, even when consistent with respect to the specifications, introduces a change that conflicts with other assumptions that we made about the system, and generally indicates that we should either revise the assumptions or forbid the application of the action. Both cases are important to consider in the development life cycle of the core-closed KB and the action definitions.

5 Static Verification

In this section, we investigate the problem of computing whether the execution of an action, no matter the specific instantiation, always preserves given properties of core-closed knowledge bases. We focus on properties expressed as MUST/MAY queries and define the static verification problem as follows.

Definition 1 (Static Verification). *Let $\mathcal{K}$ be a DL-Lite$^{\mathcal{F}}$ core-closed knowledge base, q be a* MUST/MAY *query, and γ be an action with free variables from the language presented above. Let $\vec{\theta}$ be an assignment for the input variables of γ that transforms γ into the* grounded *action $\gamma\vec{\theta}$. Let $\mathcal{K}^{\gamma\vec{\theta}}$ be the DL-Lite$^{\mathcal{F}}$ core-closed knowledge base resulting from the application of the grounded action $\gamma\vec{\theta}$ onto $\mathcal{K}$. We say that the action γ* "preserves q over $\mathcal{K}$" *iff for every grounded instance $\gamma\vec{\theta}$ we have that $\mathsf{ANS}(q, \mathcal{K}) = \mathsf{ANS}(q, \mathcal{K}^{\gamma\vec{\theta}})$. The static verification problem is that of determining whether an action γ is q-preserving over $\mathcal{K}$.*

An action γ is *not* q-preserving over $\mathcal{K}$ iff there exists a grounding $\vec{\theta}$ for the input variables of γ such that $\mathsf{ANS}(q, \mathcal{K}) \neq \mathsf{ANS}(q, \mathcal{K}^{\gamma\vec{\theta}})$; that is, fixed the grounding $\vec{\theta}$ there exists a tuple $\vec{t}$ for q's answer variables such that $\vec{t} \in \mathsf{ANS}(q, \mathcal{K}) \setminus \mathsf{ANS}(q, \mathcal{K}^{\gamma\vec{\theta}})$ or $\vec{t} \in \mathsf{ANS}(q, \mathcal{K}^{\gamma\vec{\theta}}) \setminus \mathsf{ANS}(q, \mathcal{K})$.

Theorem 1 (Complexity of the Static Verification Problem). *The static verification problem, i.e. deciding whether an action γ is q-preserving over $\mathcal{K}$, can be decided in* PTIME *in data complexity and* EXPTIME *in the arities of γ and q.*

Proof. The proof relies on the fact that one could: enumerate all possible assignments $\vec{\theta}$; compute the updated knowledge bases $\mathcal{K}^{\gamma\vec{\theta}}$; check whether these are fully satisfiable; enumerate all tuples $\vec{t}$ for the query q; and, finally, check whether there exists at least one such tuple that satisfies q over $\mathcal{K}$ but not $\mathcal{K}^{\gamma\vec{\theta}}$ or vice versa. The number of assignments $\vec{\theta}$ is bounded by $\left(|\mathbf{I}^{\mathcal{M}} \uplus \mathbf{I}^{\mathcal{K}}| + ar(\gamma)\right)^{ar(\gamma)}$ as it is sufficient to replace each variable appearing in the action γ either by a known object from $\mathbf{I}^{\mathcal{M}} \uplus \mathbf{I}^{\mathcal{K}}$ or by a fresh one. The computation of the updated $\mathcal{K}^{\gamma\vec{\theta}}$ is done in polynomial time in $\mathcal{M}$ (and is exponential in the size of the action γ) as it may require the evaluation of an internal action query φ and the consecutive re-application of the transformation for a number of tuples that is bounded by a polynomial over the size of $\mathcal{M}$. As explained in Sect. 3, checking full satisfiability of the resulting core-closed knowledge base is also polynomial in $\mathcal{M}$. The number of tuples $\vec{t}$ is bounded by $\left(|\mathbf{I}^{\mathcal{M}} \uplus \mathbf{I}^{\mathcal{K}}| + ar(\gamma)\right)^{ar(q)}$ as it is enough to consider all those tuples involving known objects plus the fresh individuals introduced by the assignment $\vec{\theta}$. Checking whether a tuple $\vec{t}$ satisfies the query q over a core-closed knowledge base is decided in LogSpace in the size of $\mathcal{M}$ [15] which is, thus, also polynomial in $\mathcal{M}$.

6 Planning

As discussed throughout the paper, the execution of a mutating action modifies the configuration of a deployment and potentially changes its posture with respect to a given set of requirements. In the previous two sections, we introduced a language to encode mutating actions and we investigated the problem of checking whether the application of an action preserves the properties of a core-closed knowledge base. In this section, we investigate the plan existence and synthesis problems; that is, the problem of deciding whether there exists a sequence of grounded actions that leads the knowledge base to a state where a certain requirement is met, and the problem of finding a set of such plans, respectively. We start by defining a notion of transition system that is generated by applying actions to a core-closed knowledge base and then use this notion to focus on the mentioned planning problems. As in classical planning, the plan existence problem for plans computed over unbounded domains is undecidable [17,19]. The undecidability proof is done via reduction from the Word problem. The problem of deciding whether a deterministic Turing machine M accepts a word $w \in \{0,1\}^*$ is reduced to the plan existence problem. Since undecidability holds even for basic action effects, we can show undecidability over an unbounded domain by using the same encoding of [1].

Transition Systems. In the style of the work done in [10,21], the combination of a DL-Lite$^{\mathcal{F}}$ core-closed knowledge base and a set of actions can be viewed as the transition system it generates. Intuitively, the states of the transition system correspond to MBoxes and the transitions between states are labeled by grounded actions. A DL-Lite$^{\mathcal{F}}$ core-closed knowledge base $\mathcal{K} = (\mathcal{T}, \mathcal{A}, \mathcal{S}, \mathcal{M}_0)$, defined over the possibly infinite set of individuals $\mathbf{I}$ (and model nodes $\mathbf{I}_0^{\mathcal{M}} \subseteq \mathbf{I}$)

and the set Act of ungrounded actions, generates the transition system (TS) $\Upsilon_{\mathcal{K}} = (\mathbf{I}, \mathcal{T}, \mathcal{A}, \mathcal{S}, \Sigma, \mathcal{M}_0, \rightarrow)$ where Σ is a set of *fully satisfiable* (i.e., *core-complete* and *open-consistent*) MBoxes; $\mathcal{M}_0$ is the initial MBox; and $\rightarrow \subseteq \Sigma \times L_{\mathsf{Act}} \times \Sigma$ is a labeled transition relation with L_{Act} the set of all possible *grounded actions*. The sets Σ and $\rightarrow$ are defined by mutual induction as the smallest sets such that: if $\mathcal{M}_i \in \Sigma$ then for every grounded action $\gamma\vec{\theta} \in L_{\mathsf{Act}}$ such that the fresh MBox $\mathcal{M}_{i+1}$ resulting from the transformation $T_{\gamma\vec{\theta}}$ is core-complete and open-consistent, we have that $\mathcal{M}_{i+1} \in \Sigma$ and $(\mathcal{M}_i, \gamma\vec{\theta}, \mathcal{M}_{i+1}) \in \rightarrow$.

Since we assume that actions have input parameters that are replaced during execution by values from $\mathbf{I}$, which contains both known objects from $\mathbf{I}^{\mathcal{M}} \uplus \mathbf{I}^{\mathcal{K}}$ and possibly infinitely many fresh objects, the generated transition system $\Upsilon_{\mathcal{K}}$ is generally infinite. To keep the planning problem decidable, we concentrate on a known finite subset $\mathcal{D} \subset \mathbf{I}$ containing all the fresh nodes and value assignments to action variables that are of interest for our application. In the remainder of this paper, we discuss the plan existence and synthesis problem for finite transition systems $\Upsilon_{\mathcal{K}} = (\mathcal{D}, \mathcal{T}, \mathcal{A}, \mathcal{S}, \Sigma, \mathcal{M}_0, \rightarrow)$, whose states in Σ have a domain that is also bounded by $\mathcal{D}$.

The Plan Existence Problem. A plan is a sequence of grounded actions whose execution leads to a state satisfying a given property. Let $\mathcal{K} = (\mathcal{T}, \mathcal{A}, \mathcal{S}, \mathcal{M}_0)$ be a DL-Lite$^{\mathcal{F}}$ core-closed knowledge base; Act be a set of ungrounded actions; and let $\Upsilon_{\mathcal{K}} = (\mathcal{D}, \mathcal{T}, \mathcal{A}, \mathcal{S}, \Sigma, \mathcal{M}_0, \rightarrow)$ be its generated finite TS. Let π be a finite sequence $\gamma_1\vec{\theta}_1 \cdots \gamma_n\vec{\theta}_n$ of grounded actions taken from the set L_{Act}. We call the sequence π *consistent* iff there exists a run $\rho = \mathcal{M}_0 \xrightarrow{\gamma_1\vec{\theta}_1} \mathcal{M}_1 \xrightarrow{\gamma_2\vec{\theta}_2} \cdots \xrightarrow{\gamma_n\vec{\theta}_n} \mathcal{M}_n$ in $\Upsilon_{\mathcal{K}}$. Let q be a MUST/MAY query mentioning objects from $adom(\mathcal{K})$ and $\vec{t}$ a tuple from the set $adom(\mathcal{K})^{ar(q)}$. A consistent sequence π of grounded actions is a *plan* from $\mathcal{K}$ to $(\vec{t}, q)$ iff $\vec{t} \in \mathsf{ANS}(q, \mathcal{K}_n = (\mathcal{T}, \mathcal{A}, \mathcal{S}, \mathcal{M}_n))$ with $\mathcal{M}_n$ the final state of the run induced by π.

Definition 2 (Plan Existence). *Given a DL-Lite$^{\mathcal{F}}$ core-closed knowledge base $\mathcal{K}$, a tuple $\vec{t}$, and a* MUST/MAY *query q, the* plan existence *problem is that of deciding whether there exists a plan from $\mathcal{K}$ to $(\vec{t}, q)$.*

Example 6. Let us consider the transition system $\Upsilon_{\mathcal{K}}$ generated by the core-closed knowledge base $\mathcal{K} = (\mathcal{T}, \mathcal{A}, \mathcal{S}, \mathcal{M}_0)$ having the set of partially-closed assertions $\mathcal{M}_0$ defined as

$$\{\mathsf{S3::Bucket}(b),\ \mathsf{KMS::Key}(k),\ \mathsf{bucketEncryptionRule}(b, r),\ \mathsf{bucketKey}(r, k),\ \mathsf{bucketKeyEnabled}(r, true),\ \mathsf{enableKeyRotation}(k, false)\}$$

and the set of action labels Act containing the actions deleteBucket, createBucket, deleteKey, createKey, enableKeyRotation, putBucketEncryption, and deleteBucketEncryption. Let us assume that we are interested in verifying the existence of a sequence of grounded actions that when applied onto the knowledge base would configure the bucket node b to be encrypted with a rotating key. Formally, this is equivalent to checking the existence of a consistent plan π that when executed

on the transition system $\Upsilon_{\mathcal{K}}$ leads to a state $\mathcal{M}_n$ such that the tuple $\vec{t} = b$ is in the set $\mathsf{ANS}(q, \mathcal{K}_n = (\mathcal{T}, \mathcal{A}, \mathcal{S}, \mathcal{M}_n))$ for q the query

$$q[x] = \mathsf{S3::Bucket}(x) \ \wedge \ \textsc{Must}\ \big(\exists y, z.\ \mathsf{bucketSSEncryption}(x, y) \wedge \mathsf{bucketKey}(y, z) \ \wedge \ \mathsf{enableKeyRotation}(z, true)\big)$$

It is easy to see that the following three sequences of grounded actions are valid plans from $\mathcal{K}$ to (b, q):

$$\pi_1 = \mathsf{enableKeyRotation}(k)$$
$$\pi_2 = \mathsf{createKey}(k_1) \cdot \mathsf{enableKeyRotation}(k_1) \cdot \mathsf{putBucketEncryption}(b, k_1)$$
$$\pi_3 = \mathsf{deleteBucketEncryption}(b, k) \cdot \mathsf{createKey}(k_1) \cdot \mathsf{enableKeyRotation}(k_1) \cdot \mathsf{putBucketEncryption}(b, k_1)$$

If, for example, a bucket was only allowed to have one encryption (by means of a functional axiom in $\mathcal{S}$), then π_2 would not be a valid plan, as it would generate an inconsistent run leading to a state $\mathcal{M}_i$ that is not open-consistent w.r.t. $\mathcal{S}$.

Lemma 3. *The plan existence problem for a finite transition system $\Upsilon_{\mathcal{K}}$ generated by a DL-Lite$^{\mathcal{F}}$ core-closed knowledge base $\mathcal{K}$ and a set of actions* Act*, over a finite domain of objects $\mathcal{D}$, reduces to graph reachability over a graph whose number of states is at most exponential in the size of $\mathcal{D}$.*

The Plan Synthesis Problem. We now focus on the problem of finding plans that satisfy a given condition. As discussed in the previous paragraph, we are mostly driven by query answering; in particular, by conditions corresponding to a tuple (of objects from our starting deployment configuration) satisfying a given requirement expressed as a Must/May query. Clearly, this problem is meaningful in our application of interest because it corresponds to finding a set of potential sequences of changes that would allow one to reach a configuration satisfying (resp., not satisfying) one, or more, security mitigations (resp., vulnerabilities). We concentrate on DL-Lite$^{\mathcal{F}}$ core-closed knowledge bases and their generated finite transition systems, where potential fresh objects are drawn from a fixed set $\mathcal{D}$. We are interested in sequences of grounded actions that are minimal and ignore sequences that extend these. We sometimes call such minimal sequences *simple plans*. A plan π from an initial core-closed knowledge base $\mathcal{K}$ to a goal condition b is minimal (or simple) *iff* there does not exist a plan π' (from the same initial $\mathcal{K}$ to the same goal condition b) s.t. $\pi = \pi' \cdot \sigma$, for σ a non-empty suffix of grounded actions.

In Algorithm 1, we present a depth-first search algorithm that, starting from $\mathcal{K}$, searches for all simple plans that achieve a given target query membership condition. The transition system $\Upsilon_{\mathcal{K}}$ is computed, and stored, on the fly in the Successors sub-procedure and the graph is explored in a depth-first search traversal fashion.

Algorithm 1: FindPlans($\mathcal{K}, \mathcal{D}, \mathsf{Act}, \langle \vec{t}, q \rangle$)

Inputs : A ccKB $\mathcal{K} = (\mathcal{T}, \mathcal{A}, \mathcal{S}, \mathcal{M}_0)$, a domain $\mathcal{D}$, a set of actions Act and a pair $\langle \vec{t}, q \rangle$ of an answer tuple and a MUST/MAY query
Output: A possibly empty set Π of consistent simple plans

```
1  def FindPlans ( K, D, Act, ⟨t, q⟩):
2      Π := ∅;
3      S := ⊥;
4      AllPlanSearch(M_0, ϵ, ∅, K, D, Act, ⟨t, q⟩) ;
5      return Π;
6  def AllPlanSearch ( M, π, V, K, D, Act, ⟨t, q⟩):
7      if M ∈ V then
8          return;
9      if t ∈ ANS(q, ⟨T, A, S, M⟩) then
10         Π := Π ∪ {π};
11         return;
12     Q := ∅;
13     foreach ⟨γθ, M'⟩ ∈ Successors(M, Act, D) do
14         Q.push(⟨γθ, M'⟩);
15     V := V ∪ {M};
16     while Q ≠ ∅ do
17         ⟨γθ, M'⟩ = Q.pop();
18         AllPlanSearch(M', π · γθ, V, K, D, Act, ⟨t, q⟩);
19     V := V ∖ {M};
20     return;
21 def Successors (M, Act, D):
22     if S[M] is defined then
23         return S[M];
24     N := ∅;
25     foreach γ ∈ Act, θ ∈ D^ar(γ) do
26         M' := T_γθ(M);
27         if M' is fully satisfiable then
28             N := N ∪ {⟨γθ, M'⟩}
29     S[M] := N;
30     return N;
```

We note that the condition $\vec{t} \in \mathsf{ANS}(q, \langle \mathcal{T}, \mathcal{A}, \mathcal{S}, \mathcal{M} \rangle)$ (line 9) could be replaced by any other query satisfiability condition and that one could easily rewrite the algorithm to be parameterized by a more general boolean goal. For

example, the condition that a given tuple $\vec{t}$ is *not* an answer to a query q over the analyzed state, with the query q representing an undesired configuration, or a boolean formula over multiple query membership assertions. We also note that Algorithm 1 could be simplified to return only one simple plan, if a plan exists, or NULL, if a plan does not exist, thus solving the so-called *plan generation problem*. We refer the reader to the full version of this paper [16] containing the plan generation algorithm (full version, Appendix A.1) and the proofs of Theorem 2 and 3 below (full version, Appendices A.2 and A3, respectively).

Theorem 2 (Minimal Plan Synthesis Correctness). *Let $\mathcal{K}$ be a DL-Lite$^{\mathcal{F}}$ core-closed knowledge base, $\mathcal{D}$ be a fixed finite domain, Act be a set of ungrounded action labels, and $\langle \vec{t}, q \rangle$ be a goal. Then a plan π is returned by the algorithm $\mathsf{FindPlans}(\mathcal{K}, \mathcal{D}, \mathsf{Act}, \langle \vec{t}, q \rangle)$ if and only if π is a minimal plan from $\mathcal{K}$ to $\langle \vec{t}, q \rangle$.*

Theorem 3 (Minimal Plan Synthesis Complexity). *The* FindPlans *algorithm runs in polynomial time in the size of $\mathcal{M}$ and exponential time in size of $\mathcal{D}$.*

7 Related Work

The syntax of the action language that we presented in this paper is similar to that of [1,12,13]. Differently from their work, we disallow complex action effects to be nested inside conditional statements, and we define basic action effects that consist purely in the addition and deletion of concept and role $\mathcal{M}$-assertions. Thus, our actions are much less general than those used in their framework. The semantics of their action language is defined in terms of changes applied to instances, and the action effects are captured and encoded through a variant of $\mathcal{ALCHOIQ}$ called $\mathcal{ALCHOIQ}_{br}$. In our work, instead, the execution of an action updates a portion of the core-closed knowledge base $\mathcal{K}$—the core $\mathcal{M}$, which is interpreted under a close-world assumption and can be seen as a partial assignment for the interpretations that are models of $\mathcal{K}$. Since we directly manipulate $\mathcal{M}$, the semantics of our actions is more similar to that of [21] and, in general, to ABox updates [22,23]. Like the frameworks introduced in [9–11,20], our actions are parameterized and when combined with a core-closed knowledge base generate a transition system. In [11], the authors focus on a variant of *Knowledge and Action Bases* [21] called *Explicit-Input KABs* (eKABs); in particular, on finite and on state-bounded eKABs, for which planning existence is decidable. Our generated transition systems are an adaptation of the work done in *Description Logic based Dynamic Systems*, *KABs*, and *eKABs* to our setting of core-closed knowledge bases. In [24], the authors address decidability of the plan existence problem for logics that are subset of $\mathcal{ALCOI}$. Their action language is similar to the one presented in this paper; including pre-conditions, in the form of a set of ABox assertions, post-conditions, in the form of basic addition or removal of assertions, concatenation, and input parameters. In [11], the plan synthesis

problem is discussed also for lightweight description logics. Relying on the FOL-reducibility of DL-Lite$^{\mathcal{A}}$, it is shown that plan synthesis over DL-Lite$^{\mathcal{A}}$ can be compiled into an ADL planning problem [25]. This does not seem possible in our case, as not all necessary tests over core-closed knowledge bases are known to be FOL-reducible. In [10] and [9], the authors concentrate on verifying and synthesizing temporal properties expressed in a variant of μ-calculus over description logic based dynamic systems, both problems are relevant in our application scenario and we will consider them in future works.

8 Conclusion

We focused on the problem of analyzing cloud infrastructure encoded as description logic knowledge bases combining complete and incomplete information. From a practical standpoint, we concentrated on formalizing and foreseeing the impact of potential changes pre-deployment. We introduced an action language to encode mutating actions, whose semantics is given in terms of changes induced to the complete portion of the knowledge base. We defined the static verification problem as the problem of deciding whether the execution of an action, no matter the specific parameters passed, always preserves a set of properties of the knowledge base. We characterized the complexity of the problem and provided procedural steps to solve it. We then focused on three formulations of the classical AI planning problem: namely, plan existence, generation, and synthesis. In our setting, the planning problem is formulated with respect to the transition system arising from the combination of a core-closed knowledge base and a set of actions; goals are given in terms of one, or more, MUST/MAY conjunctive query membership assertion; and plans of interest are simple sequences of parameterized actions.

Acknowledgments. This work is supported by the ERC Consolidator grant D-SynMA (No. 772459).

References

1. Ahmetaj, S., Calvanese, D., Ortiz, M., Simkus, M.: Managing change in graph-structured data using description logics. ACM Trans. Comput. Log. **18**(4), 27:1–27:35 (2017)
2. Artale, A., Calvanese, D., Kontchakov, R., Zakharyaschev, M.: The DL-lite family and relations. J. Artif. Intell. Res. **36**, 1–69 (2009)
3. Baader, F., Horrocks, I., Lutz, C., Sattler, U.: An Introduction to Description Logic. Cambridge University Press, Cambridge (2017)
4. Backes, J., et al.: Reachability analysis for AWS-based networks. In: Dillig, I., Tasiran, S. (eds.) CAV 2019. LNCS, vol. 11562, pp. 231–241. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-25543-5_14
5. Backes, J., et al.: Stratified abstraction of access control policies. In: Lahiri, S.K., Wang, C. (eds.) CAV 2020. LNCS, vol. 12224, pp. 165–176. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-53288-8_9

6. Backes, J., et al.: Semantic-based automated reasoning for AWS access policies using SMT. In: Bjørner, N., Gurfinkel, A. (eds.) 2018 Formal Methods in Computer Aided Design, FMCAD 2018, Austin, TX, USA, 30 October–2 November 2018, pp. 1–9. IEEE (2018). https://doi.org/10.23919/FMCAD.2018.8602994
7. Bouchet, M., et al.: Block public access: trust safety verification of access control policies. In: Devanbu, P., Cohen, M.B., Zimmermann, T. (eds.) ESEC/FSE 2020: 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Virtual Event, USA, 8–13 November 2020, pp. 281–291. ACM (2020). https://doi.org/10.1145/3368089.3409728
8. Calvanese, D., Giacomo, G.D., Lembo, D., Lenzerini, M., Rosati, R.: EQL-lite: effective first-order query processing in description logics. In: Veloso, M.M. (ed.) Proceedings of the 20th International Joint Conference on Artificial Intelligence, IJCAI 2007, Hyderabad, India, 6–12 January 2007, pp. 274–279 (2007). http://ijcai.org/Proceedings/07/Papers/042.pdf
9. Calvanese, D., De Giacomo, G., Montali, M., Patrizi, F.: Verification and synthesis in description logic based dynamic systems. In: Faber, W., Lembo, D. (eds.) RR 2013. LNCS, vol. 7994, pp. 50–64. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39666-3_5
10. Calvanese, D., Montali, M., Patrizi, F., Giacomo, G.D.: Description logic based dynamic systems: modeling, verification, and synthesis. In: Yang, Q., Wooldridge, M.J. (eds.) Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, 25–31 July 2015, pp. 4247–4253. AAAI Press (2015). http://ijcai.org/Abstract/15/604
11. Calvanese, D., Montali, M., Patrizi, F., Stawowy, M.: Plan synthesis for knowledge and action bases. In: Kambhampati, S. (ed.) Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9–15 July 2016, pp. 1022–1029. IJCAI/AAAI Press (2016). http://www.ijcai.org/Abstract/16/149
12. Calvanese, D., Ortiz, M., Simkus, M.: Evolving graph databases under description logic constraints. In: Eiter, T., Glimm, B., Kazakov, Y., Krötzsch, M. (eds.) Informal Proceedings of the 26th International Workshop on Description Logics, Ulm, Germany, 23–26 July 2013. CEUR Workshop Proceedings, vol. 1014, pp. 120–131. CEUR-WS.org (2013). http://ceur-ws.org/Vol-1014/paper_82.pdf
13. Calvanese, D., Ortiz, M., Simkus, M.: Verification of evolving graph-structured data under expressive path constraints. In: Martens, W., Zeume, T. (eds.) 19th International Conference on Database Theory, ICDT 2016, Bordeaux, France, 15–18 March 2016. LIPIcs, vol. 48, pp. 15:1–15:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2016). https://doi.org/10.4230/LIPIcs.ICDT.2016.15
14. Cauli, C., Li, M., Piterman, N., Tkachuk, O.: Pre-deployment security assessment for cloud services through semantic reasoning. In: Silva, A., Leino, K.R.M. (eds.) CAV 2021. LNCS, vol. 12759, pp. 767–780. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-81685-8_36
15. Cauli, C., Ortiz, M., Piterman, N.: Closed- and open-world reasoning in dl-lite for cloud infrastructure security. In: Proceedings of the 18th International Conference on Principles of Knowledge Representation and Reasoning, KR 2021, Hanoi, Vietnam (2021)
16. Cauli, C., Ortiz, M., Piterman, N.: Actions over core-closed knowledge bases (2022). https://doi.org/10.48550/ARXIV.2202.12592. https://arxiv.org/abs/2202.12592
17. Chapman, D.: Planning for conjunctive goals. Artif. Intell. **32**(3), 333–377 (1987). https://doi.org/10.1016/0004-3702(87)90092-0

18. Cook, B.: Formal reasoning about the security of amazon web services. In: Chockler, H., Weissenbacher, G. (eds.) CAV 2018. LNCS, vol. 10981, pp. 38–47. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-96145-3_3
19. Erol, K., Nau, D.S., Subrahmanian, V.S.: Complexity, decidability and undecidability results for domain-independent planning. Artif. Intell. **76**(1–2), 75–88 (1995). https://doi.org/10.1016/0004-3702(94)00080-K
20. Giacomo, G.D., Masellis, R.D., Rosati, R.: Verification of conjunctive artifact-centric services. Int. J. Cooperative Inf. Syst. **21**(2), 111–140 (2012). https://doi.org/10.1142/S0218843012500025
21. Hariri, B.B., Calvanese, D., Montali, M., Giacomo, G.D., Masellis, R.D., Felli, P.: Description logic knowledge and action bases. J. Artif. Intell. Res. **46**, 651–686 (2013)
22. Kharlamov, E., Zheleznyakov, D., Calvanese, D.: Capturing model-based ontology evolution at the instance level: the case of dl-lite. J. Comput. Syst. Sci. **79**(6), 835–872 (2013). https://doi.org/10.1016/j.jcss.2013.01.006
23. Liu, H., Lutz, C., Milicic, M., Wolter, F.: Foundations of instance level updates in expressive description logics. Artif. Intell. **175**(18), 2170–2197 (2011). https://doi.org/10.1016/j.artint.2011.08.003
24. Milicic, M.: Planning in action formalisms based on DLS: first results. In: Calvanese, D., et al. (eds.) Proceedings of the 2007 International Workshop on Description Logics (DL2007), Brixen-Bressanone, near Bozen-Bolzano, Italy, 8–10 June 2007. CEUR Workshop Proceedings, vol. 250. CEUR-WS.org (2007). http://ceur-ws.org/Vol-250/paper_59.pdf
25. Pednault, E.P.D.: ADL and the state-transition model of action. J. Logic Comput. **4**(5), 467–512 (1994). https://doi.org/10.1093/logcom/4.5.467
26. Tobies, S.: A NExpTime-complete description logic strictly contained in C^2. In: Flum, J., Rodriguez-Artalejo, M. (eds.) CSL 1999. LNCS, vol. 1683, pp. 292–306. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48168-0_21