Proceedings of the 18th International Conference on Principles of Knowledge Representation and Reasoning
Main Track

Closed- and Open-world Reasoning in DL-Lite for Cloud Infrastructure Security

Claudia Cauli[1] , **Magdalena Ortiz**[2] , **Nir Piterman**[1]
[1]University of Gothenburg
[2]TU Wien

Abstract

Infrastructure in the cloud is deployed through configuration files, which specify the resources to be created, their settings, and their connectivity. We aim to model infrastructure *before deployment* and reason about it so that potential vulnerabilities can be discovered and security best practices enforced. Description logics are a good match for such modeling efforts and allow for a succinct and natural description of cloud infrastructure. Their open-world assumption allows capturing the distributed nature of the cloud, where a newly deployed infrastructure could connect to pre-existing resources not necessarily owned by the same user. However, parts of the infrastructure that are fully known need closed-world reasoning, calling for the usage of expressive formalisms, which increase the computational complexity of reasoning. Here, we suggest an extension of DL-Lite$^{\mathcal{F}}$ that is tailored for capturing such cloud infrastructure. Our logic allows combining a core part that is completely defined (closed-world) and interacts with a partially known environment (open-world). We show that this extension preserves the first-order rewritability of DL-Lite$^{\mathcal{F}}$ for knowledge-base satisfiability and conjunctive query answering. Security properties combine universal and existential reasoning about infrastructure. Thus, we also consider the problem of conjunctive query satisfiability and show that it can be solved in logarithmic space in data complexity.

1 Introduction

Complex cloud infrastructure is managed through code files that are compiled into atomic deployment instructions as part of a process known as Infrastructure as Code, IaC. As of 2021, known IaC frameworks include AWS CloudFormation, Terraform, Microsoft Azure Resource Manager, Google Cloud Deployment Manager, Chef, and Puppet, to name a few. Unfortunately, though, the same features that make IaC a convenient and powerful deployment tool—reusability, modularity, and shareability—also threaten the security of the cloud. IaC files are often recycled and combined, with little consideration of whether the original business context and security requirements apply to the new usage scenario. The security vulnerabilities arising from such a practice are subtle and widespread and need to be detected early, at the level of configuration files, *before* potentially-vulnerable infrastructure is deployed.

For such reasons, we research the application of knowledge representation formalisms to the modeling and reasoning of IaC files and work towards a comprehensive framework that fits into the scene set by existing tools (such as static analysis, linters, and rule-based recommendation systems) to secure cloud infrastructure pre-deployment. Description logics are a good match for such modeling efforts. They allow us to succinctly and unambiguously describe cloud infrastructures, and to leverage decidable reasoning services, often implemented in efficient off-the-shelf engines, when reasoning about their security.

By the distributed nature of the cloud, users can configure their infrastructure to connect to resources that are running elsewhere but not declared in their accounts. This happens frequently; for instance, when users have permission to perform operations on resources that they do not own, such as *write* or *read* permissions on a shared storage instance. As a consequence, IaC files may combine objects for which we have full knowledge, as *declared* in the configuration file, with objects for which we only have partial knowledge, as *referenced by* the configuration file. Although the structural specifications are known for both types of resources, the actual configuration of objects that are not declared in the IaC file is not known. *Is the shared storage encrypted? Is it accessible through a web server? Is it publicly readable or writable?* To answer these questions, we need to combine closed- and open-world reasoning in a way that enables verification and refutation of queries representing potential vulnerabilities. In previous work (Cauli et al. 2021a), we introduced the idea of using DL-based reasoning techniques for cloud infrastructure security, and used the expressive $\mathcal{ALCOIQ}$ to model and reason about AWS CloudFormation, Amazon Web Services proprietary IaC framework. We simulated closed-world reasoning on selected nodes using the rich constructors available in $\mathcal{ALCOIQ}$, such as nominals, universal restrictions, and counting quantifiers. However, reasoning about security using this logic was not efficient, as basic services like satisfiability are NExpTime-complete (Tobies 1999; Baader et al. 2017) and the encoding of vulnerability queries turned out to be non-trivial for users that are not versed in description logic. This work highlighted the need for a formalism that scales to the size of cloud deployments, offers a more transparent and straightforward modeling language, and does not require cumbersome specifications of security properties to catch the desired interpretation.

In this paper, we instead introduce a lightweight description logic that is tailored to model cloud infrastructure, at the same time ensuring tractable reasoning. We extend the popular DL-Lite$^{\mathcal{F}}$ with *specification* predicates whose interpretation is closed over a *core* part of the knowledge base (KB) but open elsewhere. We call such KBs *core-closed* knowledge bases. We show that this specific way of combining open and closed interpretations of the same predicates does not incur complexity penalties. Indeed, we show that satisfiability and query entailment over core-closed KBs are first-order reducible. To reason about *mitigations* and *vulnerabilities* to security threats, and in analogy to the terminology used for 3-valued reasoning in the model-checking community, we introduce MUST and MAY conjunctive queries and devise a simple logical language for the specification of such properties. Technically, properties that *must* hold are resolved via query entailment and properties that *may* hold are resolved via query satisfiability. We show that computing whether a tuple $\vec{t}$ is a *sat-answer* of a given query can be solved in logarithmic space in the core portion of the KB.

The paper is structured as follows. In Sec. 2 we motivate the choices made in the contributions put forth by this paper. In Sec. 3 we review the background on DL-Lite$^{\mathcal{F}}$ and conjunctive queries. In Sec. 4 and 5 we introduce core-closed KBs and study KB satisfiability. In Sec. 6 and 7 we discuss conjunctive query entailment and satisfiability. In Sec. 8 we present our security queries. We then discuss related work (Sec. 9) and conclude in Sec. 10. Results and proofs that are omitted in this paper are found in the full version.

2 Motivation

In this section, we emphasize how the application of description logic to cloud security drives the two main contributions of this paper: *core*-closed KBs and MUST/MAY queries.

IaC Modeling In the Infrastructure as Code paradigm, the creation of resources is managed through *configuration files* that declare types and settings of the resource instances to be created, and are automatically compiled into atomic deployment instructions. Configuration files must validate against *specification files*, supplied by the cloud provider to describe how each type of resource can be declared and configured. In addition to the usual TBox and ABox, we introduce here two dedicated sets of assertions and axioms, denoted as $\mathcal{M}$ and $\mathcal{S}$ respectively, and use them to encode resources configuration and specification according to the IaC paradigm. The following is an example of how these could be used to model the structural specification of the resource type Bucket ($\mathcal{S}$) and the actual configuration of an instance called *"data"* ($\mathcal{M}$); and how these relate to the higher-level concept of Storage ($\mathcal{T}$), which could further have external entities ($\mathcal{A}$).

$$\mathcal{S} = \{\ \exists\mathsf{logsStore} \sqsubseteq \mathsf{Bucket}, \exists\mathsf{logsStore}^- \sqsubseteq \mathsf{Bucket}\ \}$$
$$\mathcal{M} = \{\ \mathsf{Bucket}(\mathit{data}), \mathsf{logsStore}(\mathit{data}, \mathit{logs})\ \}$$
$$\mathcal{T} = \{\ \mathsf{Bucket} \sqsubseteq \mathsf{Storage}\ \}$$
$$\mathcal{A} = \{\ \mathsf{Storage}(\mathit{externalStorage})\ \}$$

Resources that are declared in an IaC configuration file are in the process of being deployed but do not yet exist. We informally call these the *template resources*. These form an infrastructure that can be connected to other external resources—not declared in the current deployment template but already running elsewhere. We call these the *boundary resources*, as they lie at the boundaries of the known *core* infrastructure. In the example above, *data* is a template node and *logs* is a boundary node. Boundary and external nodes are not part of our deployment. We may not own these cloud resources and have no knowledge of their configuration, but still, have permission to use them. However, we do know that *these must have some configuration that conforms to the specifications* too; therefore, we adopt an open-world assumption when it comes to boundary resources configuration w.r.t. the general system specifications. In contrast, we assume to have complete information about the configuration of *our* template resources w.r.t. the specifications and, thus, apply closed-world reasoning over these. In our example, where *logs* is a boundary node, although we do not own its configuration we certainly know that it *must* be a bucket and that it *may* have a logsStore property configured. Regarding the *data* object, which is a template node, we exclude the possibility of it being involved in additional relations (such as being the source or target of a logsStore property). In fact, had there been any further properties they would have been declared, and since this resource instance does not yet exist it cannot be pointed to by any node that is external to the current deployment. We call the pair $\langle \mathcal{S}, \mathcal{M} \rangle$ the *core* of our system, and refer to the richer KBs described above as *core*-closed KBs.

Querying for Vulnerabilities and Mitigations In security, we seek query languages to express that mitigations to security threats *must* be present (vs. may be absent) and vulnerabilities *may* be present (vs. must be absent). Such a requirement calls for efficient decision procedures for *query satisfiability*, in addition to query entailment. In our usage scenario, Boolean combinations of so-called MUST/MAY queries serve that purpose. We define MUST/MAY queries by nesting regular conjunctive queries within the scope of a MUST or MAY operator and resolve these via query entailment and query satisfiability, respectively.

This implementation allows us, for example, to query for potentially vulnerable instances such as *"Buckets that may store their own logs"*, encoded as

$$q_v[x] = \textsc{May}\ \mathsf{logsStore}(x, x),$$

and to query for instances where mitigations to security threats are in place such as *"Buckets that must be server-side encrypted"*, expressed as

$$q_m[x] = \textsc{Must}\ \big(\ \exists y,z.\ \mathsf{encrypt}(x, y) \wedge \mathsf{sseConfig}(y, z)\ \big)\ .$$

In addition, through Boolean combinations of MUST/MAY queries we combine multiple properties into one single check; e.g., the following query witnessing the breach of the mitigation *"Buckets that may store logs must be encrypted"*:

$$q[x] = \textsc{Must}\ \mathsf{Bucket}(x)\ \wedge\ \textsc{May}\ \big(\ \exists y.\ \mathsf{logsStore}(y, x)\ \big)$$
$$\wedge\ \neg\textsc{Must}\ \big(\ \exists y,z.\ \mathsf{encrypt}(x, y) \wedge \mathsf{sseConfig}(y, z)\ \big)\ .$$

We note that the combination of *core* closed-world reasoning and MUST/MAY queries enables a very precise framework for the verification and refutation of security properties. Importantly, such precision allows us to reduce the rate of false-positive results that would clutter the quality of the findings presented to users and security engineers. For instance, the set of answers to the vulnerability query q_v over the sample model introduced in the previous paragraph would contain the *logs* node but would **not** contain the *data* node. The *data* bucket is already known to store its logs in a distinct bucket and is assumed to not have any more properties. The *logs* bucket, instead, belongs to the universe of external underspecified resources, for which it is not known whether it stores any logs (and where), and might actually store logs on itself—a fact that is worth spotlighting while assessing the security of IaC deployments. As can be seen in the extended version of our previous work (Cauli et al. 2021b), the examples discussed here are very close to real IaC deployments' encoding and to the properties that are of interest for a security review.

3 Background

Here, we review DL-Lite$^{\mathcal{F}}$ and CQs, which provide the basis for the contributions made throughout the paper.

Let $\mathbf{C}$, $\mathbf{R}$, and $\mathbf{I}$ be countably infinite sets of concept names, role names, and individual names. A DL-Lite$^{\mathcal{F}}$ concept B is built according to the syntax $\mathsf{B} ::= \bot \mid \mathsf{A} \mid \exists\mathsf{P}$, where A is a concept name from the set $\mathbf{C}$ and P is a role name R, or its inverse R^-, from the set $\mathbf{R}$. A TBox $\mathcal{T}$ is a collection of positive inclusion axioms $\mathsf{B}_1 \sqsubseteq \mathsf{B}_2$, negative inclusion axioms $\mathsf{B}_1 \sqsubseteq \neg\mathsf{B}_2$, and functionality axioms $\mathsf{Funct\ P}$. An ABox $\mathcal{A}$ is a collection of concept and role assertions, both positive and negative, of the form $\mathsf{A}(a)$, $\neg\mathsf{A}(a)$, $\mathsf{R}(a, b)$, and $\neg\mathsf{R}(a, b)$, with a,b individual names from the set $\mathbf{I}$. A DL-Lite$^{\mathcal{F}}$ knowledge base (KB) $\mathcal{K}$ is the pair $\langle\mathcal{T}, \mathcal{A}\rangle$. The semantics of a DL-Lite$^{\mathcal{F}}$ KB is given in terms of interpretations. An interpretation is the tuple $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, where $\Delta^{\mathcal{I}}$ is a non-empty domain and $\cdot^{\mathcal{I}}$ is an interpretation function. The function $\cdot^{\mathcal{I}}$ assigns to every concept name A a set $\mathsf{A}^{\mathcal{I}}$ subset of $\Delta^{\mathcal{I}}$, to every role name R a set $\mathsf{R}^{\mathcal{I}}$ subset of $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$, and to every individual name a a domain element $a^{\mathcal{I}}$ form the set $\Delta^{\mathcal{I}}$. We adopt the unique name assumption (UNA), which requires that $a^{\mathcal{I}} \neq b^{\mathcal{I}}$ for individual names $a \neq b$. The interpretation function is extended to concepts and roles as follows.

$$\bot^{\mathcal{I}} = \emptyset \qquad (\neg\mathsf{B})^{\mathcal{I}} = \Delta^{\mathcal{I}} \smallsetminus \mathsf{B}^{\mathcal{I}}$$
$$(\mathsf{R}^-)^{\mathcal{I}} = \{(a, b) \mid (b, a) \in \mathsf{R}^{\mathcal{I}}\}$$
$$(\exists\mathsf{P})^{\mathcal{I}} = \{a \mid \exists b \in \Delta^{\mathcal{I}}.(a, b) \in \mathsf{P}^{\mathcal{I}}\}$$

An interpretation $\mathcal{I}$ is a model of $\mathcal{K}$ *iff* for all α in $\mathcal{T} \cup \mathcal{A}$ we have $\mathcal{I} \models \alpha$. The KB $\mathcal{K}$ is said to be satisfiable when there exists at least one model. We write $\mathcal{K} \models \alpha$ whenever $\mathcal{I} \models \alpha$ for all models $\mathcal{I}$ of $\mathcal{K}$.

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 $\mathcal{K}$ 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*. In the rest of the paper, we consider inequalities only in the case of query satisfiability and not in the case of query entailment.

4 DL-Lite$^{\mathcal{F}}$ Core-closed KBs

In this section, we introduce the so-called "*core*-closed" knowledge bases, their models, and their unique features.

A DL-Lite$^{\mathcal{F}}$core-closed KB is the tuple $\mathcal{K} = \langle\mathcal{T}, \mathcal{A}, \mathcal{S}, \mathcal{M}\rangle$, built from a standard KB $\langle\mathcal{T}, \mathcal{A}\rangle$ and a *core* $\langle\mathcal{S}, \mathcal{M}\rangle$. As described in section 2, the set $\mathcal{S}$ contains DL-Lite$^{\mathcal{F}}$ axioms representing the core structural specifications and the set $\mathcal{M}$ contains positive concept and role assertions representing the core configuration. Syntactically, $\mathcal{M}$ is similar to an ABox $\mathcal{A}$ but, differently from $\mathcal{A}$, it is assumed to be complete with respect to the specifications $\mathcal{S}$. As usual, $\langle\mathcal{T}, \mathcal{A}\rangle$ encodes the incomplete terminological and assertional knowledge that, in our setting, may refer to both the (closed) core and the surrounding (open) world.

The core-closed KB $\mathcal{K}$ is defined over the sets of concept names $\mathbf{C}$, role names $\mathbf{R}$, and individual names $\mathbf{I}$. The set of concepts is partitioned into specification concepts $\mathbf{C}^{\mathcal{S}}$ and open concepts $\mathbf{C}^{\mathcal{K}}$. The set of roles is partitioned into specification roles $\mathbf{R}^{\mathcal{S}}$ and open roles $\mathbf{R}^{\mathcal{K}}$. The set of individuals is partitioned into the core individuals $\mathbf{I}^{\mathcal{M}}$ and the open individuals $\mathbf{I}^{\mathcal{K}}$. We call $\mathbf{C}^{\mathcal{S}}$ and $\mathbf{R}^{\mathcal{S}}$ *core-closed predicates* as their extension is closed over the core domain and open otherwise. In contrast, we call $\mathbf{C}^{\mathcal{K}}$ and $\mathbf{R}^{\mathcal{K}}$ *open predicates*. From now on, we denote symbols from the alphabet $\mathbf{X}^{\mathcal{X}}$ with the subscript $\mathcal{X}$, and symbols from the alphabet $\mathbf{X}$ with no subscript. We now define which assertions are $\mathcal{M}$-assertions, i.e., fall into the scope of $\mathcal{M}$; and which assertions are $\mathcal{A}$-assertions, i.e., fall into the scope of $\mathcal{A}$.

$$\mathcal{M} \subseteq \{\ \mathsf{A}_{\mathcal{S}}(a_{\mathcal{M}}),\ \mathsf{R}_{\mathcal{S}}(a_{\mathcal{M}}, a_{\mathcal{M}}),\ \mathsf{R}_{\mathcal{S}}(a_{\mathcal{M}}, a_{\mathcal{K}}),\ \mathsf{R}_{\mathcal{S}}(a_{\mathcal{K}}, a_{\mathcal{M}})\ \}$$
$$\mathcal{A} \subseteq \{\ \mathsf{A}_{\mathcal{K}}(a),\ \mathsf{R}_{\mathcal{K}}(a, b),\ \mathsf{A}_{\mathcal{S}}(a_{\mathcal{K}}),\ \mathsf{R}_{\mathcal{S}}(a_{\mathcal{K}}, b_{\mathcal{K}})\ \}$$

We assume $\mathcal{M}$ to be complete and consistent w.r.t. $\mathcal{S}$, and interpret as false all $\mathcal{M}$-assertions missing from $\mathcal{M}$. The usual open-world assumption is made over $\mathcal{A}$-assertions.

For convenience, we sometimes consider the set of open individuals $\mathbf{I}^{\mathcal{K}}$ as further partitioned into a set of *boundary* elements $\mathbf{I}^{B}$, which appear in $\mathcal{M}$, and a set of *free* elements $\mathbf{I}^{\mathcal{K}'}$, which appear only in $\mathcal{A}$. With this notation in mind, we introduce the active domain of constants appearing in $\mathcal{M}$, denoted $adom(\mathcal{M})$ and defined as the set $\mathbf{I}^{\mathcal{M}} \uplus \mathbf{I}^{B}$. We adopt the standard name assumption over individuals in $adom(\mathcal{M})$ and the unique name assumption over individuals in $\mathbf{I}^{\mathcal{K}'}$. In section 7, we will refer to this assumption as *core standard name assumption*. Such an assumption reflects the knowledge that we have of the system that we aim at modeling. According to it, the nodes declared in the (known) *core* part of the infrastructure simply coincide with their interpretation domain; but the nodes belonging to the (unknown) surrounding part of the infrastructure need to be mapped to the domain. All these elements are distinct.

According to the DL-Lite$^{\mathcal{F}}$ syntax, axioms are built from concepts $\mathsf{B} ::= \mathsf{B}^{\mathcal{S}} \mid \mathsf{B}^{\mathcal{K}}$, with $\mathsf{B}^{\mathcal{K}} ::= \bot \mid \mathsf{A}_{\mathcal{K}} \mid \exists \mathsf{P}_{\mathcal{K}}$ and $\mathsf{B}^{\mathcal{S}} ::= \bot \mid \mathsf{A}_{\mathcal{S}} \mid \exists \mathsf{P}_{\mathcal{S}}$, where P, called basic role, is either an atomic role R or its inverse R^- from the set $\mathbf{R}$. Axioms in $\mathcal{S}$ ($\mathcal{S}$-axioms) refer only to core-closed predicates; whereas $\mathcal{T}$-axioms can refer both to core-closed predicates (on the left-hand side of concept inclusions) and to open predicates:

$$\mathcal{S} \subseteq \{\ \mathsf{B}_1^{\mathcal{S}} \sqsubseteq \mathsf{B}_2^{\mathcal{S}},\ \mathsf{B}_1^{\mathcal{S}} \sqsubseteq \neg \mathsf{B}_2^{\mathcal{S}},\ \mathsf{Func}(\mathsf{P}_{\mathcal{S}})\ \}$$
$$\mathcal{T} \subseteq \{\ \mathsf{B}_1 \sqsubseteq \mathsf{B}_2^{\mathcal{K}},\ \mathsf{B}_1 \sqsubseteq \neg \mathsf{B}_2^{\mathcal{K}},\ \mathsf{Func}(\mathsf{P}_{\mathcal{K}})\ \}$$

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 $\Delta^{\mathcal{I}}$. 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 $\mathcal{M}$. If $\mathcal{K}$ has at least one model, then $\mathcal{K}$ is satisfiable.

The notion of FOL-reducibility captures the property that we can reduce satisfiability and query answering over a core-closed KB to evaluating a first-order logic query over $\mathcal{A}$ and $\mathcal{M}$ considered as minimal models. In particular, we consider the following interpretations of $\mathcal{A}$ and $\mathcal{M}$: the *database* interpretation of $\mathcal{A}$, denoted $db(\mathcal{A})$, and the *labeled transition system* interpretation of $\mathcal{M}$, denoted $lts(\mathcal{M})$.

Given an ABox $\mathcal{A}$, with $adom(\mathcal{A})$ its active domain of constants, we denote by $db(\mathcal{A})$ the interpretation $(\Delta^{db(\mathcal{A})}, \cdot^{db(\mathcal{A})})$ that is defined as follows:

$$\begin{aligned}
\Delta^{db(\mathcal{A})} &= adom(\mathcal{A}) \\
a^{db(\mathcal{A})} &= a, \text{ for each constant } a \text{ appearing in } \mathcal{A} \\
\mathsf{A}^{db(\mathcal{A})} &= \{\ a \mid \mathsf{A}(a) \in \mathcal{A}\ \} \text{ for each } \mathsf{A} \in \mathbf{C} \\
\mathsf{R}^{db(\mathcal{A})} &= \{\ (a,b) \mid \mathsf{R}(a,b) \in \mathcal{A}\ \} \text{ for each } \mathsf{R} \in \mathbf{R}.
\end{aligned}$$

For an MBox $\mathcal{M}$, we denote by $lts(\mathcal{M})$ the interpretation $(\Delta^{lts(\mathcal{M})}, \cdot^{lts(\mathcal{M})})$ that is defined similarly as above with one notable exception: the interpretation of concept and role names is computed only for those concepts and roles that fall within the scope of $\mathcal{M}$, that is, core-closed predicates $\mathbf{C}^{\mathcal{S}}$ and $\mathbf{R}^{\mathcal{S}}$. It is easy to see that $db(\mathcal{A}) \models \mathcal{A}$, and, precisely, it is the *minimal* model of $\mathcal{A}$. Similarly, $lts(\mathcal{M}) \models^{\mathsf{CWA}} \mathcal{M}$, and, in particular, it is the *unique* model of $\mathcal{M}$.

We consider various reasoning problems over core-closed KBs and study their combined and data complexity (Vardi 1982). We measure data complexity in terms of the model $\mathcal{M}$, which we expect to be much larger than $\mathcal{A}$.

5 Core-closed KB Satisfiability

As per standard DL-Lite$^{\mathcal{F}}$ results, we now show that satisfiability of core-closed KBs *(i)* can be reduced to consistency of the functionality axioms and of the axioms in the negative closure of $\mathcal{T}$ and $\mathcal{S}$, and *(ii)* it is FOL-reducible. Readers familiar with the work of (Calvanese et al. 2007b) will recognize the analogies between the two presentations.

As defined in the previous section, a DL-Lite$^{\mathcal{F}}$ core-closed KB $\mathcal{K} = \langle \mathcal{T}, \mathcal{A}, \mathcal{S}, \mathcal{M} \rangle$ is satisfiable if and only if there exists at least one interpretation $\mathcal{I}$ such that $\mathcal{I} \models \mathcal{T} \cup \mathcal{A} \cup \mathcal{S}$ and $\mathcal{I} \models^{\mathsf{CWA}} \mathcal{M}$. Let ga be a function that takes as input a basic role P and two individuals a, b and returns a membership assertion in the following way: $\mathsf{ga}(\mathsf{P}, a, b) = \mathsf{R}(a,b)$ if $\mathsf{P} = \mathsf{R}$, and $\mathsf{ga}(\mathsf{P}, a, b) = \mathsf{R}(b,a)$ if $\mathsf{P} = \mathsf{R}^-$.

Canonical Interpretation The canonical interpretation of a core-closed KB $\mathcal{K}$ is constructed according to the notion of boundary chase, or *bchase*. The *bchase* is built by exploiting the *applicable positive inclusion axioms* in the sets $\mathcal{T}$ and $\mathcal{S}$.

Definition 1 (Applicable Axioms)**.** *Let $\mathcal{X}$ be a set of $\mathcal{M}$-assertions, $\mathcal{Y}$ be a set of $\mathcal{A}$-assertions, and $PI_{\mathcal{T}}$ and $PI_{\mathcal{S}}$ be the positive inclusion axioms in $\mathcal{T}$ and $\mathcal{S}$, respectively. Then, an axiom $\alpha \in PI_{\mathcal{T}} \uplus PI_{\mathcal{S}}$ is said to be* applicable in *$\mathcal{Y}$ to an assertion $f \in \mathcal{Y} \uplus \mathcal{X}$ if:*

c1 $\alpha = \mathsf{A} \sqsubseteq \mathsf{A}_{\mathcal{K}}$, $f = \mathsf{A}(a)$, *and* $\mathsf{A}_{\mathcal{K}}(a) \notin \mathcal{Y}$

c2 $\alpha = \exists\mathsf{P} \sqsubseteq \mathsf{A}_{\mathcal{K}}$, $f = \mathsf{ga}(\mathsf{P}, a, b)$, *and* $\mathsf{A}_{\mathcal{K}}(a) \notin \mathcal{Y}$

c3 $\alpha = \mathsf{A} \sqsubseteq \exists\mathsf{P}_{\mathcal{K}}$, $f = \mathsf{A}(a)$, *and there is no* b *such that* $\mathsf{ga}(\mathsf{P}_{\mathcal{K}}, a, b) \in \mathcal{Y}$

c4 $\alpha = \exists\mathsf{P} \sqsubseteq \exists\mathsf{P}_{\mathcal{K}}$, $f = \mathsf{ga}(\mathsf{P}, a, b)$, *and there is no* c *such that* $\mathsf{ga}(\mathsf{P}_{\mathcal{K}}, a, c) \in \mathcal{Y}$

c5 $\alpha = \mathsf{A}_{\mathcal{S}} \sqsubseteq \mathsf{A}'_{\mathcal{S}}$, $f_{\mathcal{A}} = \mathsf{A}_{\mathcal{S}}(a_{\mathcal{K}})$, *and* $\mathsf{A}'_{\mathcal{S}}(a_{\mathcal{K}}) \notin \mathcal{Y}$

c6 $\alpha = \exists\mathsf{P}_{\mathcal{S}} \sqsubseteq \mathsf{A}_{\mathcal{S}}$, $f = \mathsf{ga}(\mathsf{P}_{\mathcal{S}}, a_{\mathcal{K}}, b)$, *and* $\mathsf{A}_{\mathcal{S}}(a_{\mathcal{K}}) \notin \mathcal{Y}$

c7 $\alpha = \mathsf{A}_{\mathcal{S}} \sqsubseteq \exists\mathsf{P}_{\mathcal{S}}$, $f_{\mathcal{A}} = \mathsf{A}_{\mathcal{S}}(a_{\mathcal{K}})$, *and there is no $a_{\mathcal{M}}$ s.t.* $\mathsf{ga}(\mathsf{P}_{\mathcal{S}}, a_{\mathcal{K}}, a_{\mathcal{M}}) \in \mathcal{X}$ *and no $c_{\mathcal{K}}$ s.t.* $\mathsf{ga}(\mathsf{P}_{\mathcal{S}}, a_{\mathcal{K}}, c_{\mathcal{K}}) \in \mathcal{Y}$

c8 $\alpha = \exists\mathsf{P}'_{\mathcal{S}} \sqsubseteq \exists\mathsf{P}_{\mathcal{S}}$, $f = \mathsf{ga}(\mathsf{P}'_{\mathcal{S}}, a_{\mathcal{K}}, b)$, *and for no $a_{\mathcal{M}}$,* $\mathsf{ga}(\mathsf{P}_{\mathcal{S}}, a_{\mathcal{K}}, a_{\mathcal{M}}) \in \mathcal{X}$ *and for no $c_{\mathcal{K}}$,* $\mathsf{ga}(\mathsf{P}_{\mathcal{S}}, a_{\mathcal{K}}, c_{\mathcal{K}}) \in \mathcal{Y}$

Starting with $\mathcal{Y}_0 = \mathcal{A}$ and $\mathcal{X} = \mathcal{M}$ (that is, starting with the contents of $\mathcal{A}$ and $\mathcal{M}$), axioms are incrementally *applied* to assertions. At each i-th step, an axiom α is applied to an assertion f in $\mathcal{Y}_j \cup \mathcal{X}$ and a new membership assertion is added to $\mathcal{Y}_{j+1}$. Following such step, α is not applicable in $\mathcal{Y}_{j+1}$ to the assertion f anymore. Depending on the order of application, syntactically different sets of assertions could be generated. To account for this, from now on we assume the existence of an infinite ordered set of fresh symbols $\mathbf{I}^+$, from which we draw fresh individuals, and *apply* assertions following a preset order.

Definition 2 (Boundary Chase)**.** *Let $\mathcal{K} = \langle \mathcal{T}, \mathcal{A}, \mathcal{S}, \mathcal{M} \rangle$ be a DL-Lite$^{\mathcal{F}}$ core-closed KB, $PI_{\mathcal{T}}$ the positive inclusion axioms in $\mathcal{T}$, $PI_{\mathcal{S}}$ the positive inclusion axioms in $\mathcal{S}$, and $\mathbf{I}^+$ a set of fresh individuals. Then, the* boundary chase *of $\mathcal{K}$, denoted bchase($\mathcal{K}$), is defined as:*

$$bchase(\mathcal{K}, \mathcal{X}) = \bigcup_{j \in \mathbb{N}} \mathcal{Y}_j$$

where $\mathcal{X} = \mathcal{M}$, $\mathcal{Y}_0 = \mathcal{A}$, and $\mathcal{Y}_{j+1} = \mathcal{Y}_j \cup \{f_{new}\}$, where f_{new} depends on the rule being applied:

```
let f be the first assertion s.t. there is $\alpha$ applicable in $\mathcal{Y}_j$ to f
let $\alpha$ be the first applicable axiom
let $a_{new}$ be the next available constant in the ordered set $\mathbf{I}^+$
switch $< f, \alpha >$
  case c1: $f_{new} = \mathsf{A}_{\mathcal{K}}(a)$
  case c2: $f_{new} = \mathsf{A}_{\mathcal{K}}(a)$
  case c3: $f_{new} = \mathsf{ga}(\mathsf{P}_{\mathcal{K}}, a, a_{new})$
  case c4: $f_{new} = \mathsf{ga}(\mathsf{P}_{\mathcal{K}}, a, a_{new})$
  case c5: $f_{new} = \mathsf{A}'_{\mathcal{S}}(a_{\mathcal{K}})$
  case c6: $f_{new} = \mathsf{A}_{\mathcal{S}}(a_{\mathcal{K}})$
  case c7: $f_{new} = \mathsf{ga}(\mathsf{P}_{\mathcal{S}}, a_{\mathcal{K}}, a_{new})$
  case c8: $f_{new} = \mathsf{ga}(\mathsf{P}_{\mathcal{S}}, a_{\mathcal{K}}, a_{new})$
```

As customary, we note that *(i)* negative inclusion and functionality axioms play no role in the construction of the bchase, and that *(ii)* this notion of bchase is *fair*, that is, all applicable axioms will *eventually* be applied, as formalized by the following statements. Let *bchase*$_i$ be the bchase built at the $i-$th rule application. Then, if there is an $i \in \mathbb{N}$ s.t. axiom α is applicable in *bchase*$_i(\mathcal{K},\mathcal{X})$ to an assertion $f \in$ *bchase*$_i(\mathcal{K}, \mathcal{X})$, then there is a $j > i$ s.t. *bchase*$_{j+1}(\mathcal{K}, \mathcal{X}) =$ *bchase*$_j(\mathcal{K}, \mathcal{X}) \cup \{f'\}$, where f' is the result of applying α to f in *bchase*$_j(\mathcal{K}, \mathcal{X})$.

Moreover, as clear from definitions 1 and 2, we have that an axiom is applicable to an $\mathcal{M}$-assertion only when a fresh assertion about a *"boundary"* individual $a_{\mathcal{K}}$ can be added to the chase. However, only $\mathcal{A}$-assertions are included in the bchase itself, and the procedure of adding fresh assertions only generates $\mathcal{A}$-assertions and never generates $\mathcal{M}$-assertions. We formalize this in the following lemma.

Lemma 1. *Let $\mathcal{K} = \langle \mathcal{T}, \mathcal{A}, \mathcal{S}, \mathcal{M} \rangle$ be a DL-Lite$^{\mathcal{F}}$ core-closed KB, let i be an index in $\mathbb{N}$, and let* bchase$_i$*($\mathcal{K}$,$\mathcal{M}$) be $\mathcal{K}$'s i-th boundary chase. Then,* bchase$_i$*($\mathcal{K}$,$\mathcal{M}$) does not contain $\mathcal{M}$-assertions.*

We are now ready to define the notion of *canonical interpretation* of a core-closed KB.

Definition 3 (Canonical Interpretation)**.** *The canonical interpretation of a core-closed KB $\mathcal{K}$, denoted as $can(\mathcal{K})$, is the interpretation $can(\mathcal{K}) = (\Delta^{can(\mathcal{K})}, \cdot^{can(\mathcal{K})})$ where:*

$$\begin{aligned}
\Delta^{can(\mathcal{K})} &= \mathbf{I}^{\mathcal{M}} \uplus \mathbf{I}^{\mathcal{K}} \uplus \mathbf{I}^{+} \\
a^{can(\mathcal{K})} &= a \quad \text{for } a \in adom(\mathcal{M}) \cup bchase(\mathcal{K}, \mathcal{M}) \\
\mathsf{A}_{\mathcal{K}}{}^{can(\mathcal{K})} &= \{a \mid \mathsf{A}_{\mathcal{K}}(a) \in bchase(\mathcal{K}, \mathcal{M})\} \\
\mathsf{R}_{\mathcal{K}}{}^{can(\mathcal{K})} &= \{(a, b) \mid \mathsf{R}_{\mathcal{K}}(a, b) \in bchase(\mathcal{K}, \mathcal{M})\} \\
\mathsf{A}_{\mathcal{S}}{}^{can(\mathcal{K})} &= \mathsf{A}_{\mathcal{S}}{}^{lts(\mathcal{M})} \cup \{a \mid \mathsf{A}_{\mathcal{S}}(a) \in bchase(\mathcal{K}, \mathcal{M})\} \\
\mathsf{R}_{\mathcal{S}}{}^{can(\mathcal{K})} &= \mathsf{R}_{\mathcal{S}}{}^{lts(\mathcal{M})} \cup \{(a, b) \mid \mathsf{R}_{\mathcal{S}}(a, b) \in bchase(\mathcal{K}, \mathcal{M})\}
\end{aligned}$$

We refer to the canonical model built with the i-th bchase as $can_i(\mathcal{K}) = (\Delta^{can(\mathcal{K})}, \cdot^{can_i(\mathcal{K})})$ and note that $\Delta^{lts(\mathcal{M})} \subseteq \Delta^{can(\mathcal{K})}$, $\Delta^{db(\mathcal{A})} \subseteq \Delta^{can(\mathcal{K})}$, and $\cdot^{lts(\mathcal{M})} \cup \cdot^{db(\mathcal{A})} = \cdot^{can_0(\mathcal{K})}$.

Lemma 2. *Let $\mathcal{K} = \langle \mathcal{T}, \mathcal{A}, \mathcal{S}, \mathcal{M} \rangle$ be a DL-Lite$^{\mathcal{F}}$ core-closed KB, and let $can(\mathcal{K})$ be its canonical interpretation. Then, $can(\mathcal{K})$ is a model of $\mathcal{M}$.*

Proof. We show that $can(\mathcal{K})$ models an $\mathcal{M}$-assertion f iff if $f \in \mathcal{M}$. The *'if'* direction follows from the fact that $can(\mathcal{K})$ contains $lts(\mathcal{M})$, which is a model of $\mathcal{M}$ and contains all $\mathcal{M}$-assertions f such that $f \in \mathcal{M}$. The *'only if'* direction follows from Lemma 1: in particular, $can(\mathcal{K})$ is the union of $lts(\mathcal{M})$ and *bchase($\mathcal{K}$,$\mathcal{M}$)*, and since *bchase* does not contain $\mathcal{M}$-assertions, then all $\mathcal{M}$-assertions in $can(\mathcal{K})$ are inside $lts(\mathcal{M})$. Since $lts(\mathcal{M})$ models $\mathcal{M}$, then all $\mathcal{M}$-assertions f in $can(\mathcal{K})$ are also in $\mathcal{M}$. We conclude that $can(\mathcal{K}) \models^{\mathsf{CWA}} \mathcal{M}$. □

Lemma 3. *Let $\mathcal{K} = \langle \mathcal{T}, \mathcal{A}, \mathcal{S}, \mathcal{M} \rangle$ be a DL-Lite$^{\mathcal{F}}$ core-closed KB, let $PI_{\mathcal{T}}$ be the positive inclusion axioms in $\mathcal{T}$, and let $PI_{\mathcal{S}}$ the positive inclusion axioms in $\mathcal{S}$. Then, $can(\mathcal{K})$ is a model of $(PI_{\mathcal{T}}, \mathcal{A}, PI_{\mathcal{S}}, \mathcal{M})$.*

As a consequence, every DL-Lite$^{\mathcal{F}}$ core-closed KB with only positive inclusion axioms in $\mathcal{T}$ and $\mathcal{S}$ (s.t. $PI_{\mathcal{T}} = \mathcal{T}$ and $PI_{\mathcal{S}} = \mathcal{S}$) is always satisfiable, since one can always build a $can(\mathcal{K})$ that is a model of $\mathcal{K}$. Regarding functionality assertions, the following lemma applies.

Lemma 4. *Let $\mathcal{K} = \langle \mathcal{T}, \mathcal{A}, \mathcal{S}, \mathcal{M} \rangle$ be a DL-Lite$^{\mathcal{F}}$ core-closed KB, let $F_{\mathcal{T}}$ be the subset of functionality axioms in $\mathcal{T}$, and let $F_{\mathcal{S}}$ be the subset of functionality axioms in $\mathcal{S}$. Then $can(\mathcal{K})$ is a model of $(F_{\mathcal{T}}, \mathcal{A}, F_{\mathcal{S}}, \mathcal{M})$ if and only if $db(\mathcal{A}) \cup lts(\mathcal{M})$ is a model of $(F_{\mathcal{T}}, \mathcal{A}, F_{\mathcal{S}}, \mathcal{M})$.*

NI-closure Let us now consider negative inclusion axioms. In particular, to establish a satisfaction relation between $db(\mathcal{A})$ and $lts(\mathcal{M})$, on one side, and the NIs in $\mathcal{K}$, on the other side, we need to consider the interaction between the positive and the negative inclusion axioms that are contained in $\mathcal{K}$. In the following, we materialize the interaction between the PIs and NIs contained in $\mathcal{T} \cup \mathcal{S}$ by computing their *negative inclusion closure*, $cln(\mathcal{T} \cup \mathcal{S})$. We then show that $can(\mathcal{K})$ is a model of such closure.

Definition 4. *Let $\mathcal{T}$ be a DL-Lite$^{\mathcal{F}}$ TBox, and let $\mathcal{S}$ be a DL-Lite$^{\mathcal{F}}$ SBox. We call NI-closure of $\mathcal{T} \cup \mathcal{S}$ the set $cln(\mathcal{T} \cup \mathcal{S})$ of inclusion axioms defined inductively as follows:*

1. *All NIs in $\mathcal{T} \cup \mathcal{S}$ are in $cln(\mathcal{T} \cup \mathcal{S})$;*
2. *All Fs in $\mathcal{T} \cup \mathcal{S}$ are in $cln(\mathcal{T} \cup \mathcal{S})$;*
3. *If $\mathsf{B}_1 \sqsubseteq \mathsf{B}_2 \in (\mathcal{T} \cup \mathcal{S})$, and $\mathsf{B}_2 \sqsubseteq \neg\mathsf{B}_3$ (or $\mathsf{B}_3 \sqsubseteq \neg\mathsf{B}_2$) $\in cln(\mathcal{T} \cup \mathcal{S})$, then also $\mathsf{B}_1 \sqsubseteq \neg\mathsf{B}_3 \in cln(\mathcal{T} \cup \mathcal{S})$;*
4. *If either $\exists\mathsf{P} \sqsubseteq \neg\exists\mathsf{P} \in cln(\mathcal{T} \cup \mathcal{S})$ or $\exists\mathsf{P}^- \sqsubseteq \neg\exists\mathsf{P}^- \in cln(\mathcal{T} \cup \mathcal{S})$, then both are in $cln(\mathcal{T} \cup \mathcal{S})$.*

This closure does not add negative inclusion axioms that were not implied already by $\mathcal{T} \cup \mathcal{S}$.

Lemma 5. *Let $\mathcal{T} \cup \mathcal{S}$ be a set of DL-Lite$^{\mathcal{F}}$ inclusion axioms, and let α be either a functionality axiom or a negative inclusion axiom. Then, if $cln(\mathcal{T} \cup \mathcal{S}) \models \alpha$ then $\mathcal{T} \cup \mathcal{S} \models \alpha$.*

We are now ready to show that, provided we have computed the closure $cln(\mathcal{T} \cup \mathcal{S})$, the analogous of Lemma 3 and Lemma 4 hold for NIs.

Lemma 6. *Let $\mathcal{K} = \langle \mathcal{T}, \mathcal{A}, \mathcal{S}, \mathcal{M} \rangle$ be a DL-Lite$^{\mathcal{F}}$ core-closed KB. Then, $can(\mathcal{K})$ is a model of $\mathcal{K}$ if and only if the union $db(\mathcal{A}) \cup lts(\mathcal{M})$ is a model of $cln(\mathcal{T} \cup \mathcal{S})$, $\mathcal{A}$, and $\mathcal{M}$.*

Corollary 1. *Let $\mathcal{T} \cup \mathcal{S}$ be a set of DL-Lite$^{\mathcal{F}}$ inclusion axioms, and α a functionality or negative inclusion axiom. We have that, if $\mathcal{T} \cup \mathcal{S} \models \alpha$ then $cln(\mathcal{T} \cup \mathcal{S}) \models \alpha$.*

FOL-reducibility

Lemma 7. *Let $\mathcal{K} = \langle \mathcal{T}, \mathcal{A}, \mathcal{S}, \mathcal{M} \rangle$ be a DL-Lite$^{\mathcal{F}}$ core-closed KB. Then $can(\mathcal{K})$ is a model of $\mathcal{K}$ if and only if $\mathcal{K}$ is satisfiable.*

Since $can(\mathcal{K})$ could be infinite, its construction is in general neither convenient nor possible. However, the results presented so far, especially Lemmas 6 and 7, allow us to conclude that in order to check satisfiability of a DL-Lite$^{\mathcal{F}}$ core-closed KB $\mathcal{K}$ it is sufficient to compute $cln(\mathcal{T} \cup \mathcal{S})$ and to look at $db(\mathcal{A}) \cup lts(\mathcal{M})$.

Theorem 1. *Let $\mathcal{K} = \langle \mathcal{T}, \mathcal{A}, \mathcal{S}, \mathcal{M} \rangle$ be a DL-Lite$^{\mathcal{F}}$ core-closed KB. Then $\mathcal{K}$ is satisfiable if and only if $db(\mathcal{A}) \cup lts(\mathcal{M})$ is a model of $cln(\mathcal{T} \cup \mathcal{S})$, $\mathcal{A}$, and $\mathcal{M}$.*

Proof. $\Rightarrow$ $\mathcal{K}$ is satisfiable. From Lemma 7, it follows that $can(\mathcal{K})$ is a model of $\mathcal{K}$. From Lemma 6, it follows that $db(\mathcal{A}) \cup lts(\mathcal{M})$ is a model of $cln(\mathcal{T} \cup \mathcal{S})$, $\mathcal{A}$, and $\mathcal{M}$.

$\Leftarrow$ If $db(\mathcal{A}) \cup lts(\mathcal{M})$ is a model of $cln(\mathcal{T} \cup \mathcal{S})$, $\mathcal{A}$, and $\mathcal{M}$, then, from Lemma 6, $can(\mathcal{K})$ is a model of $\mathcal{K}$, and, from Lemma 7, $\mathcal{K}$ is satisfiable. □

Verifying that $db(\mathcal{A}) \cup lts(\mathcal{M})$ models $cln(\mathcal{T} \cup \mathcal{S})$, $\mathcal{A}$, and $\mathcal{M}$, can now be done by writing a Boolean FOL query over $db(\mathcal{A}) \cup lts(\mathcal{M})$ itself. We use the following translation function δ from axioms in $cln(\mathcal{T} \cup \mathcal{S})$ to FOL formulas:

$$\begin{aligned}
\delta(\mathsf{func}\ \mathsf{R}) &= \exists x, y, z.\mathsf{R}(x, y) \wedge \mathsf{R}(x, z) \wedge y \neq z \\
\delta(\mathsf{func}\ \mathsf{R}^-) &= \exists x, y, z.\mathsf{R}(y, x) \wedge \mathsf{R}(z, x) \wedge y \neq z \\
\delta(\mathsf{B}_1 \sqsubseteq \neg \mathsf{B}_2) &= \exists x.\gamma_1(x) \wedge \gamma_2(x)
\end{aligned}$$

where B_i is a DL-Lite$^{\mathcal{F}}$ complex concept, and in the last equation we have: $\gamma_i(x) = \mathsf{A}_i(x)$ if $\mathsf{B}_i = \mathsf{A}_i$; $\gamma_i(x) = \exists y_i.\mathsf{R}_i(x, y_i)$ if $\mathsf{B}_i = \exists \mathsf{R}_i$; and $\gamma_i(x) = \exists y_i.\mathsf{R}_i(y_i, x)$ if $\mathsf{B}_i = \exists \mathsf{R}_i^-$. Intuitively, such formulas detect inconsistencies that would make $db(\mathcal{A}) \cup lts(\mathcal{M})$ not model the axioms in the NI-closure.

To summarize, to decide satisfiability of a DL-Lite$^{\mathcal{F}}$ core-closed KB $\mathcal{K}$ we need to: *(1)* compute $db(\mathcal{A})$ and $lts(\mathcal{M})$; *(2)* compute $cln(\mathcal{T} \cup \mathcal{S})$; and *(3)* compute the Boolean FOL formula q_{unsat} as the union of all Boolean formulas returned by the application of δ to every axiom in $cln(\mathcal{T} \cup \mathcal{S})$. We show how this is done in Algorithm 1.

Algorithm 1: The algorithm Consistent

```
Inputs : $\mathcal{K} = \langle \mathcal{T}, \mathcal{A}, \mathcal{S}, \mathcal{M} \rangle$
Output: true if $\mathcal{K}$ is satisfiable, false otherwise
1 def Consistent ($\mathcal{K}$):
2     $q_{unsat} ::= \bot$;
3     foreach $\alpha \in cln(\mathcal{T} \cup \mathcal{S})$ do
4         $q_{unsat} ::= q_{unsat} \vee \delta(\alpha)$;
5     if $q_{unsat}^{db(\mathcal{A}) \cup lts(\mathcal{M})} = \emptyset$ then
6         return true;
7     return false;
```

Lemma 8. *Let $\mathcal{K} = \langle \mathcal{T}, \mathcal{A}, \mathcal{S}, \mathcal{M} \rangle$ be a DL-Lite$^{\mathcal{F}}$ core-closed KB. Then, the algorithm* $\mathsf{Consistent}(\mathcal{K})$ *terminates, and $\mathcal{K}$ is satisfiable iff* $\mathsf{Consistent}(\mathcal{K})$ *returns true.*

Proof. Termination follows from the fact that $cln(\mathcal{T} \cup \mathcal{S})$ is a finite set. The query q_{unsat} verifies whether there is an axiom α in the NI-closure that is violated in $db(\mathcal{A}) \cup lts(\mathcal{M})$. The algorithm returns true only when such an axiom does not exists, therefore, $db(\mathcal{A}) \cup lts(\mathcal{M})$ is a model of *all* assertions in $cln(\mathcal{T} \cup \mathcal{S})$, and, by Theorem 1, $\mathcal{K}$ is satisfiable. □

As a consequence of Lemma 8, we get:

Corollary 2. *Satisfiability of a DL-Lite$^{\mathcal{F}}$ core-closed KB is FOL reducible.*

6 CQ Entailment

In this section, we discuss entailment of a conjunctive query q over a core-closed KB $\mathcal{K}$ and computation of the certain answers $ans(q, \mathcal{K})$. Let us recall that, for the entailment problem, we are interested in queries that do not contain inequalities. By the construction of $\mathcal{K}$'s canonical model $can(\mathcal{K})$ presented in the previous section, it is easy to see that the preliminary properties that hold for DL-Lite$^{\mathcal{F}}$ KBs (Calvanese et al. 2007b) also hold for DL-Lite$^{\mathcal{F}}$ core-closed KBs. In particular, we have that *(i)* there exists an isomorphism from $\mathcal{K}$'s canonical model to every model of $\mathcal{K}$ and *(ii)* the answers to a CQ over $\mathcal{K}$ correspond to the answers to the query over $can(\mathcal{K})$. Based on these results , we solve entailment of a CQ q over a core-closed KB $\mathcal{K}$ via query reformulation. The query is reformulated based on the *PI* axioms in $\mathcal{T} \cup \mathcal{S}$ and then evaluated over $db(\mathcal{A})$ $\cup$ $lts(\mathcal{M})$. Classically, the algorithm PerfectRef takes in input a CQ q and returns a collection of fresh CQs that reformulate q by internalizing positive inclusion axioms and reducing atoms that can be unified (Calvanese et al. 2007b). We apply PerfectRef as is and, hence, omit its description from the presentation. We report the CAns procedure in algorithm 2 and state its correctness by the following theorem.

Theorem 2. *Let $\mathcal{K} = \langle \mathcal{T}, \mathcal{A}, \mathcal{S}, \mathcal{M} \rangle$ be a DL-Lite$^{\mathcal{F}}$ core-closed KB, let q be a conjunctive query, and $\vec{t}$ a tuple of constants in $\mathcal{K}$. Then $\vec{t} \in ans(q, \mathcal{K})$ iff $\vec{t} \in \mathsf{CAns}(q, \mathcal{K})$.*

As a result of the tight correspondence between the standard and the core-closed setting w.r.t. canonical model construction and query reformulation, we have that $ans(q, \mathcal{K}) = \mathsf{CAns}(\mathsf{q}, \mathcal{K})$ and that, hence, answering conjunctive queries in core-closed DL-Lite$^{\mathcal{F}}$ KBs is *FOL*-reducible. In addition, due to such correspondence, other properties of conjunctive query answering over DL-Lite$^{\mathcal{F}}$ hold as well, e.g., it is also the case that there is a $\mathcal{K}$ with no finite interpretation that answers a CQ, just like usual DL-Lite$^{\mathcal{F}}$ KBs (Calvanese et al. 2007b).

Theorem 3. *Query entailment in DL-Lite$^{\mathcal{F}}$ core-closed KBs is AC^0 in data complexity and* NP*-complete in combined complexity.*

Algorithm 2: The algorithm CAns

```
  Inputs : CQ q, $\mathcal{K} = \langle \mathcal{T}, \mathcal{A}, \mathcal{S}, \mathcal{M} \rangle$
  Output: $ans(q,\mathcal{K})$
1 def CAns ($\mathcal{K}$,q):
2     if not Consistent($\mathcal{K}$) then
3         return AllTup(q,$\mathcal{K}$)
4     return PerfectRef$(q, \mathcal{T} \cup \mathcal{S})^{db(\mathcal{A}) \cup lts(\mathcal{M})}$;
```

7 CQ Satisfiability

We now discuss satisfiability of a conjunctive query with inequalities q w.r.t a core-closed KB $\mathcal{K}$ and computation of the sat answers $sat\text{-}ans(q, \mathcal{K})$. Let q be the conjunctive query $q[\vec{x}] = \exists \vec{y}.conj(\vec{x}, \vec{y})$ where $\vec{x}$ is the set of q's answer variables and $\vec{y}$ are the existentially-quantified variables. We call a *CQ-assertion* a query q where the answer variables $\vec{x}$ have been replaced by an assignment $\vec{c}$ and define the problem of CQ-assertion satisfiability as follows.

Definition 5 (CQ-assertion Satisfiability)**.** *An asserted conjunctive query with inequalities $q[\vec{c}] = \exists \vec{y}.conj(\vec{c}, \vec{y})$ is said to be satisfiable w.r.t. $\mathcal{K} = \langle \mathcal{T}, \mathcal{A}, \mathcal{S}, \mathcal{M} \rangle$ iff there exists an interpretation $\mathcal{I}$ model of $\mathcal{K}$ such that $\mathcal{I}$ satisfies $q[\vec{c}]$.*

To decide CQ-assertion satisfiability we require solving satisfiability of a core-closed KB without the unique name assumption, which we discuss in the following paragraph.

Core-Closed KB Satisfiability w/o UNA Let us drop the unique name assumption on pairs of individuals that are not covered by the *core standard name assumption* (cf. section 4). Intuitively, these include all pairs referring to individuals not in $\mathcal{M}$'s active domain plus all pairs where exclusively *one* element can be a boundary node from $\mathbf{I}^B$. The ABox $\mathcal{A}$ can now contain inequality assertions $a_j \not\approx a_k$ where $a_j \in \mathbf{I}^{\mathcal{K}}$ and $a_k \in \mathbf{I}^{\mathcal{K}'}$. Pairs of individuals not falling in this set definition, that is, pairs s.t. $a_j \in \mathbf{I}^{\mathcal{M}}$ or $a_j, a_k \in \mathbf{I}^B$, will still be assumed to be distinct by the *core SNA*. For instance, a boundary node a_j in $\mathbf{I}^B$ could correspond to the same domain object as an external node a_k in $\mathbf{I}^{\mathcal{K}'}$. We refer to this assumption as $\mathcal{A}$-noUNA.

Lemma 9. *Let $\mathcal{K} = \langle \mathcal{T}, \mathcal{A}, \mathcal{S}, \mathcal{M} \rangle$ be a DL-Lite$^{\mathcal{F}}$ core-closed KB with inequalities in $\mathcal{A}$ interpreted under $\mathcal{A}$-*noUNA*. Then, one can construct in polynomial time in $\mathbf{I}^{\mathcal{K}'}$ and $\mathbf{I}^B$ a core-closed KB $\mathcal{K}' = \langle \mathcal{T}', \mathcal{A}', \mathcal{S}, \mathcal{M} \rangle$ s.t. $\mathcal{A}'$ contains no inequalities and $\mathcal{K}$ is satisfiable iff $\mathcal{K}'$ is satisfiable.*

Proof. We build $\mathcal{T}'$ and $\mathcal{A}'$ by applying the following rules:

- *if* (func P)$\in \mathcal{T} \cup \mathcal{S}$ and $\{\mathsf{ga}(\mathsf{P}, a_i, a_j), \mathsf{ga}(\mathsf{P}, a_i, a_k)\} \subseteq \mathcal{A}$ for $a_j \neq a_k$ s.t. $a_j \in \mathbf{I}^{\mathcal{K}}$ and $a_k \in \mathbf{I}^{\mathcal{K}'}$, *then* replace all occurrences of a_k with a_j in $\mathcal{A}$.
- *if* (func P)$\in \mathcal{T} \cup \mathcal{S}$ and $\{\mathsf{ga}(\mathsf{P}, a_i, a_j), \mathsf{ga}(\mathsf{P}, a_i, a_k)\} \subseteq \mathcal{A}$ for $a_j \neq a_k$ s.t. $a_j \in \mathbf{I}^{\mathcal{M}}$ or $a_j, a_k \in \mathbf{I}^B$, or *if* $\mathcal{A}$ contains $a \not\approx a$ for some a, *then* the KB is not satisfiable and we add $\mathsf{A}^f(a_f)$ to $\mathcal{A}$ and $\mathsf{A}^f \sqsubseteq \bot$ to $\mathcal{T}$ for fresh concept A^f and constant a_f.

Lastly, we remove all inequalities and denote the sets as $\mathcal{A}'$ and $\mathcal{T}'$. For the rest of this proof, see the full version. □

Theorem 4. *Under the $\mathcal{A}$-*noUNA* assumption, satisfiability of DL-Lite$^{\mathcal{F}}$ core-closed KBs with inequalities is AC^0 in data complexity and* P*-complete in combined complexity.*

Solving CQ-assertion Satisfiability Consider a CQ-assertion $\exists \vec{y}.conj(\vec{c}, \vec{y})$. From now on, for simplicity, let us denote it as $conj$, which is treated as the set of atoms that the query comprises. The set $conj$ can be *grounded* by replacing variables $\vec{y}$ with constants $\vec{d}$. The assignment $\vec{d}$ may contain both constants from $\mathbf{I}$ and fresh constants. When $conj$ is grounded in $\vec{d}$, denoted $conj(\vec{d})$, all atoms become assertions. Assertions $\mathsf{C}(c), \mathsf{r}(c, c'), \mathsf{r}(c, a), \mathsf{r}(a, c)$, $c \not\approx c'$, $c \not\approx a$, $a \not\approx c$, and $b \not\approx b'$ where $\mathsf{C} \in \mathbf{C}^{\mathcal{S}}$, $\mathsf{r} \in \mathbf{R}^{\mathcal{S}}$, $c, c' \in \mathbf{I}^{\mathcal{M}}$, $b, b' \in \mathbf{I}^B$, and $a \notin \mathbf{I}^{\mathcal{M}}$ are called $\mathcal{M}$*-assertions*. All other assertions are called $\mathcal{A}$*-assertions*. A grounded CQ-assertion $conj(\vec{d})$ is therefore partitioned into the two sets $conj_{\mathcal{A}}$ and $conj_{\mathcal{M}}$. The set $conj_{\mathcal{M}}$ is the subset of $conj$ containing $\mathcal{M}$-*assertions*. To distinguish the predicate assertions from the inequality assertions we refer to its subsets as $conj^{\star}_{\mathcal{M}}$ and $conj^{\not\approx}_{\mathcal{M}}$, respectively. The set $conj_{\mathcal{A}}$ is the subset of $conj$ containing $\mathcal{A}$-*assertions*. We add to this set the inequality $a \not\approx a'$ for every distinct $a \in \mathbf{I}^{\mathcal{K}}$ and $a' \in \mathbf{I}^{\mathcal{K}'}$. We do this to preserve these objects' distinctness when invoking the satisfiability without UNA, according to the following lemma.

Lemma 10. *An asserted conjunctive query with inequalities $q[\vec{c}] = \exists \vec{y}.conj(\vec{c}, \vec{y})$ is satisfiable w.r.t. $\mathcal{K} = \langle \mathcal{T}, \mathcal{A}, \mathcal{S}, \mathcal{M} \rangle$ iff there exists at least one assignment $\vec{d}$ for the variables in $\vec{y}$ such that $conj(\vec{c}, \vec{d})$ does not include assertion $x \not\approx x$ for every constant x and is grounded in the sets $conj_{\mathcal{A}}$ and $conj_{\mathcal{M}}$ such that $conj^{\star}_{\mathcal{M}} \subseteq \mathcal{M}$ and $\mathcal{K}' = (\mathcal{T}, \mathcal{A} \uplus conj_{\mathcal{A}}, \mathcal{S}, \mathcal{M})$ is satisfiable without the UNA.*

We now show that finding (part of) the assignment $\vec{d}$, which induces the partition to $conj_{\mathcal{M}}$ and $conj_{\mathcal{A}}$ of Lemma 10, can be done in log-space in $\mathcal{M}$. We introduce terminology and notation that will be helpful to understand the reasoning behind Algorithm 3. The algorithm manipulates a set of atoms. We refer to this set as *conj* even though its composition changes between different stages of the running of the algorithm. Each atom in *conj* is either an *assertion*, whose arguments are all constants, or an *unassigned atom*, whose arguments contain some variables. In high-level, *conj* contains five types of atoms that play a different role in determining query satisfiability. These five types are: $\mathcal{A}$-*assertions*, $\mathcal{M}$-*assertions*, $\mathcal{A}$-*atoms*, $\mathcal{M}$-*atoms*, and *atoms*. We have already introduced the first two sets, $conj_{\mathcal{A}}$ and $conj_{\mathcal{M}}$, and assumed that $conj_{\mathcal{A}}$ enforces the unique name assumption on $\mathbf{I}^{\mathcal{K}}$ and $\mathbf{I}^{\mathcal{K}'}$ by explicitly including additional inequalities. The remaining three types of atoms are defined as follows. *(1)* Unassigned atoms that refer to concepts in $\mathbf{C}^{\mathcal{K}}$ and roles in $\mathbf{R}^{\mathcal{K}}$ are called $\mathcal{A}$-*atoms* as they will inevitably be replaced by $\mathcal{A}$-*assertions*. We denote this set as $conj_{\mathcal{A}?}$ and highlight that it does not contain inequalities, but only concept/role atoms. *(2)* Unassigned atoms of the form $\mathsf{r}(c, y)$, $\mathsf{r}(y, c)$, $c \not\approx y$, or $y \not\approx c$ where $\mathsf{r} \in \mathbf{R}^{\mathcal{S}}$ and $c \in \mathbf{I}^{\mathcal{M}}$, are called $\mathcal{M}$-*atoms* as they will inevitably be replaced by $\mathcal{M}$-*assertions*. We denote this set as $conj_{\mathcal{M}?}$. Differently from $conj_{\mathcal{A}?}$, $conj_{\mathcal{M}?}$ may contain inequalities. Hence, we partition it into the subsets $conj^{\star}_{\mathcal{M}?}$ and $conj^{\not\approx}_{\mathcal{M}?}$. *(3)* The remaining elements in *conj* are simply called *atoms* as they might be replaced either by constants from $\mathbf{I}^{\mathcal{M}}$ or $\mathbf{I}^{\mathcal{K}}$, turning them into $\mathcal{A}$-*assertions* or $\mathcal{M}$-*assertions*, respectively. We denote this subset of *conj* as $conj_{?}$ and partition it into $conj^{\star}_{?}$ and $conj^{\not\approx}_{?}$.

Algorithm 3: Sat ($\mathcal{K}$,*conj*)

```
Inputs : Consistent $\mathcal{K} = \langle \mathcal{T}, \mathcal{A}, \mathcal{S}, \mathcal{M} \rangle$, set conj
Output: true if conj is satisfiable w.r.t. $\mathcal{K}$
1  def Sat ($\mathcal{K}$,conj):
2      if ($conj$ contains $x \not\approx x$) or ($conj^{\star}_{\mathcal{M}} \not\subseteq \mathcal{M}$) then
3          return false;
4      $conj ::= conj \smallsetminus conj_{\mathcal{M}}$;
5      if $conj == \emptyset$ then
6          return true;
7      if there is $at \leftarrow conj^{\star}_{\mathcal{M}?}$ with free variable $y$ then
8          for $a$ s.t. $at[a/y] \in \mathcal{M}$ do
9              if $\mathsf{Sat}(\mathcal{K}, conj[a/y])$ then
10                 return true;
11         return false;
12     if there is $at \leftarrow conj^{\star}_{?}$ with free variables $\vec{y}$ then
13         for $\vec{a}$ s.t. $at[\vec{a}/\vec{y}] \in \mathcal{M}$ do
14             if $\mathsf{Sat}(\mathcal{K}, conj[\vec{a}/\vec{y}])$ then
15                 return true;
16         return $\mathsf{Sat}(\mathcal{K}, conj[\vec{a^{y}}/\vec{y}])$;
17     if there is $at \leftarrow conj_{\mathcal{A}?}$ with free variable $y$ then
18         for $a \in \mathbf{I}^{\mathcal{M}}$ do
19             if $\mathsf{Sat}(\mathcal{K}, conj[a/y])$ then
20                 return true;
21         return $\mathsf{Sat}(\mathcal{K}, conj[a^{y}/y])$;
22     if there is $at \leftarrow conj^{\not\approx}_{?}$ then
23         return $\mathsf{Sat}(\mathcal{K}, conj[\vec{a^{y}}/\vec{y}])$;
24     $conj ::= conj \smallsetminus conj^{\not\approx}_{\mathcal{M}?}$;
25     $conj_{\mathcal{A}} ::= conj_{\mathcal{A}} \cup \{a \not\approx b \mid a, b \in \mathbf{I}^{B} \cup \mathbf{I}^{\mathcal{K}'} \wedge a \neq b\}$;
26     return $\mathsf{sat}^{\not\approx}_{\mathsf{noUNA}}(\mathcal{T}, \mathcal{A} \cup conj_{\mathcal{A}}, \mathcal{S}, \mathcal{M})$
```

Algorithm Description The algorithm searches for an assignment that partitions *conj* into the sets $conj_{\mathcal{M}}$ and $conj_{\mathcal{A}}$ such that $conj_{\mathcal{M}}$ is consistent with (i.e., included in) $\mathcal{M}$ and $conj_{\mathcal{A}}$ is satisfiable w.r.t. $\mathcal{K}$ when dropping the UNA. Assignments are found by replacing variables with constants in $\mathcal{A}$-*atoms*, $\mathcal{M}$-*atoms*, and *atoms*, which become $\mathcal{A}$-*assertions* or $\mathcal{M}$-*assertions*; thus, populating the sets $conj_{\mathcal{A}}$ and $conj_{\mathcal{M}}$. The algorithm starts with a set *conj* that may contain all types of assertions (i.e., $conj \subseteq conj_{\mathcal{M}} \cup conj^{\star}_{\mathcal{M}?} \cup conj^{\not\approx}_{\mathcal{M}?} \cup conj^{\star}_{?} \cup conj^{\not\approx}_{?} \cup conj_{\mathcal{A}?} \cup conj_{\mathcal{A}}$). At each recursive invocation of Algorithm 3 with the currently handled set of atoms *conj*, we have that *conj* is certainly unsatisfiable if: *(i)* it contains any inequality assertions referring to the same pair of symbols *or (2)* it contains any concept/role $\mathcal{M}$-*assertions* that are not in $\mathcal{M}$ (lines 2-3). Hence, when the code execution continues to line 4, all the $\mathcal{M}$-*assertions* $conj^{\star}_{\mathcal{M}}$ do not affect satisfiability and all the $\mathcal{M}$-*inequalities* $conj^{\not\approx}_{\mathcal{M}}$ are simply true by the underlying core SNA. These assertions can then be disregarded (line 4). If, after removing these, *conj* is empty, then it is surely satisfiable (lines 5-6). Otherwise, *conj* may still contain $\mathcal{M}$-*atoms*, $\mathcal{A}$-*atoms*, *atoms*, and $\mathcal{A}$-*assertions* (i.e., $conj \subseteq conj^{\star}_{\mathcal{M}?} \cup conj^{\not\approx}_{\mathcal{M}?} \cup conj^{\star}_{?} \cup conj^{\not\approx}_{?} \cup conj_{\mathcal{A}?} \cup conj_{\mathcal{A}}$). We prioritize the replacement of atoms that must be mapped to assertions in $\mathcal{M}$, and try all of these during replacement of both $\mathcal{M}$-*atoms* and *atoms* (lines 8-10 and 13-15). For concept/role $\mathcal{M}$-*atoms* in $conj^{\star}_{\mathcal{M}?}$ we try *all* replacements from $\mathcal{M}$. If we find an atom from $conj^{\star}_{\mathcal{M}?}$ that cannot be instantiated with an assertion in $\mathcal{M}$ leading to satisfiability of the replaced query, then *conj* is surely unsatisfiable (line 11). Otherwise, it is satisfiable (line 10). Similarly, for concept/role *atoms* in $conj^{\star}_{?}$, we *first* try *all* the replacements in $\mathcal{M}$ (lines 13-15); if none of these replacements makes *conj* satisfiable, then we try by replacing variables with fresh constants (line 16), turning the generic *atom* into an $\mathcal{A}$-*assertion*. For concept/role $\mathcal{A}$-*atoms* in $conj_{\mathcal{A}?}$, we *first* try *all* the replacements in $\mathbf{I}^{\mathcal{M}}$ (lines 18-20); if none of these replacements makes *conj* satisfiable, then we try by replacing variables with fresh constants (line 21) Notice that in both cases the $\mathcal{A}$-*atom* becomes an $\mathcal{A}$-*assertion*. When the algorithm progresses to line 22, *conj* may still contain inequality *atoms*, inequality $\mathcal{M}$-*atoms*, and $\mathcal{A}$-*assertions* (i.e., $conj \subseteq conj^{\not\approx}_{?} \cup conj^{\not\approx}_{\mathcal{M}?} \cup conj_{\mathcal{A}}$). We assign the inequality atoms by replacing variables with fresh constants (lines 22-23) and disregard the inequalities in the set $conj^{\not\approx}_{\mathcal{M}?}$, as they must be true by the core SNA (line 24). If a recursive call reaches line 25, then only $\mathcal{A}$-*assertions* are left in *conj* (i.e., $conj = conj_{\mathcal{A}}$). In line 25, we enforce the uniqueness of the pre-existing nodes in $\mathbf{I}^{B}$ and $\mathbf{I}^{\mathcal{K}'}$ and, finally, invoke the $\mathsf{sat}^{\not\approx}_{\mathsf{noUNA}}$ algorithm (line 26).

Correctness

Theorem 5. *Let $\mathcal{K} = \langle \mathcal{T}, \mathcal{A}, \mathcal{S}, \mathcal{M} \rangle$ be a DL-Lite$^{\mathcal{F}}$ core-closed KB, q a conjunctive query, and $\vec{c}$ a tuple of constants in $\mathcal{K}$. Then, $q[\vec{c}]$ is satisfiable w.r.t to $\mathcal{K}$ if and only if* $\mathsf{Sat}(\mathcal{K}, q[\vec{c}])$ *returns true.*

Corollary 3. *Query satisfiability in DL-Lite$^{\mathcal{F}}$ core-closed KBs is decided in* LOGSPACE *in data complexity and is* P*-complete in combined complexity.*

We leave open the question of whether query satisfiability is in AC^0 in data complexity. In algorithm 4, we report the procedure that given a query q computes the set $sat\text{-}ans(q, \mathcal{K})$ w.r.t. a core-closed KB $\mathcal{K}$. We now state the correctness of the algorithm.

Theorem 6. *Let $\mathcal{K} = \langle \mathcal{T}, \mathcal{A}, \mathcal{S}, \mathcal{M} \rangle$ be a DL-Lite$^{\mathcal{F}}$ core-closed KB, q a CQ with inequalities over $\mathcal{K}$, and $\vec{t}$ a tuple of constants in $\mathcal{K}$. Then, $\vec{t} \in sat\text{-}ans(q, \mathcal{K})$ iff $\vec{t} \in \mathsf{SAns}(q, \mathcal{K})$.*

Algorithm 4: The algorithm SAns

```
Inputs : CQ q, $\mathcal{K} = \langle \mathcal{T}, \mathcal{A}, \mathcal{S}, \mathcal{M} \rangle$
Output: $sat\text{-}ans$(q,$\mathcal{K}$)
1 def SAns ($\mathcal{K}$,q):
2     if not Consistent($\mathcal{K}$) then
3         return $\emptyset$
4     return
        $\{ \vec{c} \mid \vec{c} \in AllTup(q, \mathcal{K}) \wedge \mathsf{Sat}(\mathcal{K}, q[\vec{c}]) = tt \}$;
```

8 Must/May Queries

As introduced in section 2, we are interested in Boolean combinations of Must/May queries. Such a Boolean combination is a query ψ that combines nested UCQs in the scope of a Must or a May operator as follows:

$$\psi ::= \neg\psi \mid \psi_1 \wedge \psi_2 \mid \psi_1 \vee \psi_2 \mid \textsc{Must}\ \varphi \mid \textsc{May}\ \varphi_{\not\approx}$$

where φ,$\varphi_{\not\approx}$ are unions of conjunctive queries potentially containing inequalities. Note that we do not allow nesting of Must/May atoms within ψ as we believe it would not increase its expressive power. Intuitively, the reasoning needed for answering the nested queries can be decoupled from the reasoning needed to answer the higher-level Boolean combination. In particular, the set of answers to the query ψ over a core-closed KB $\mathcal{K} = \langle \mathcal{T}, \mathcal{A}, \mathcal{S}, \mathcal{M} \rangle$ with individual names $\mathbf{I}$, denoted as $\mathsf{ANS}(\psi, \mathcal{K})$, is computed as follows. Each nested query Must $q[\vec{x}]$ with q the union of conjunctive queries $\bigvee_i q_i$ is resolved by computing the set $\bigcup_i ans(q_i, \mathcal{K})$ as $\bigcup_i \mathsf{CAns}(q, \mathcal{K})$. Each nested query May $q[\vec{x}]$ with q the union of conjunctive queries with inequalities $\bigvee_i q_i$ is resolved by computing the set $\bigcup_i sat{-}ans(q_i, \mathcal{K})$ as $\bigcup_i \mathsf{SAns}(q, \mathcal{K})$. Connectives $\neg, \wedge, \vee$ are resolved by set complementation w.r.t. $\mathbf{I}$, intersection, and union, respectively.

Theorem 7. *Answering whether a given tuple $\vec{t}$ satisfies a* Must/May *query over a core-closed DL-Lite$^{\mathcal{F}}$ KBs can be decided in* LogSpace *in data complexity and is* NP*-complete in combined complexity.*

9 Related Work

Many authors have advocated for combining open- and closed-world reasoning in description logics, and proposed a variety of ways to achieve it, e.g., (Baader and Hollunder 1995; Borgwardt and Forkel 2019; Franconi, Ibáñez-García, and Seylan 2011; Gaggl, Rudolph, and Schweizer 2016). One of the most prominent approaches is to extend DLs with *closed predicates* (Franconi, Ibáñez-García, and Seylan 2011), that is, with a set of concepts and roles that are viewed as complete and their extensions fixed in all models. Our combination of open- and closed-world reasoning was tailored specifically for this application domain, and it is not obvious whether it can be easily expressed using the usual closed predicates, due to the presence of predicates that are closed over part of the domain but open on the rest.

One of the major challenges of extending DLs with closed predicates is to keep the complexity in check. Closed predicates can be simulated in expressive DLs with nominals (like $\mathcal{ALCO}$ and its extensions), but for such logics satisfiability is at least ExpTime-hard (Baader et al. 2017) and conjunctive query entailment 2ExpTime-hard (Ngo, Ortiz, and Šimkus 2016). Moreover, such an encoding is not useful for obtaining improved bounds for the data complexity. Unfortunately, query answering with closed predicates is also intractable in data complexity, and the coNP lower bound applies already to very restricted classes of *conjunctive queries (CQs)* and very weak DLs like DL-Lite$_{\text{core}}$ or $\mathcal{EL}$ (Lutz, Seylan, and Wolter 2019). Lutz, Seylan, and Wolter (2013) showed that for most lightweight DLs conjunctive query answering is FOL rewritable only under some *safety restrictions* that make the presence of closed predicates irrelevant. Our core-closed KBs resemble their safe KBs and are FOL rewritable, but the partial closed-world assumption plays an important role, particularly in the query satisfiability problem that arises from the May queries.

Semantic approaches to security are being studied (Hendre and Joshi 2015) and will soon lead to publicly available, community-maintained, threat modeling ontologies. As an example, we refer the reader to the *"Ontology-driven Threat Modeling"* incubator project by OWASP (https://github.com/OWASP/OdTM) and reflect on how this will impact the adoption of DL-based semantic reasoning techniques in threat modeling and security. We believe that our Must/May queries could be used within a first-order logic of knowledge/belief (Reiter 1992), as done in (Calvanese et al. 2007a), but this was not in the scope of the application presented in this paper.

10 Conclusion and Future Work

We introduce a variant of DL-Lite$^{\mathcal{F}}$ that combines closed- and open-world reasoning within the same predicates. Our variant is tailored for the modeling of cloud infrastructure and allows to reason about security issues that might arise in such applications. We avoid the complexity price usually involved in reasoning with closed predicates and show that we keep the convenient complexity of DL-Lite$^{\mathcal{F}}$ for KB satisfiability and conjunctive query answering, and that conjunctive query satisfiability is also tractable. We combine query answering and satisfiability in a logic that includes must and may queries over KBs, as required for testing security issues.

As future work, we are interested in including more complex knowledge in the $\mathcal{T}$-box while still keeping (data) complexity tractable. For example, complex role inclusions would be required to reason about dataflow, which is a central aspect of security. Also, to be able to reason about permissions, we would have to consider non-monotone extensions. Practically, we are interested in logical languages that would allow security engineers to pose security queries in an intuitive and easy-to-use way.

References

Baader, F., and Hollunder, B. 1995. Embedding defaults into terminological knowledge representation formalisms. *J. of Automated Reasoning* 14(1):149–180.

Baader, F.; Horrocks, I.; Lutz, C.; and Sattler, U. 2017. *An Introduction to Description Logic*. Cambridge University Press.

Borgwardt, S., and Forkel, W. 2019. Closed-world semantics for conjunctive queries with negation over *ELH*_\bot ontologies. In *JELIA*, volume 11468 of *Lecture Notes in Computer Science*, 371–386. Springer.

Calvanese, D.; Giacomo, G. D.; Lembo, D.; Lenzerini, M.; and Rosati, R. 2007a. Eql-lite: Effective first-order query processing in description logics. In *IJCAI*, 274–279.

Calvanese, D.; Giacomo, G. D.; Lembo, D.; Lenzerini, M.; and Rosati, R. 2007b. Tractable reasoning and efficient query answering in description logics: The *DL-Lite* family. *J. Autom. Reason.* 39(3):385–429.

Cauli, C.; Li, M.; Piterman, N.; and Tkachuk, O. 2021a. Pre-deployment security assessment for cloud services through semantic reasoning. In *Computer Aided Verification - 33rd International Conference, CAV 2021, Proceedings*, Lecture Notes in Computer Science.

Cauli, C.; Li, M.; Piterman, N.; and Tkachuk, O. 2021b. Pre-deployment security assessment for cloud services through semantic reasoning. Full version https://gup.ub.gu.se/publication/304989, Last accessed on 2021-07-14.

Franconi, E.; Ibáñez-García, Y. A.; and Seylan, I. 2011. Query answering with dboxes is hard. *Electr. Notes Theor. Comput. Sci.* 278:71–84.

Gaggl, S. A.; Rudolph, S.; and Schweizer, L. 2016. Fixed-domain reasoning for description logics. In Kaminka, G. A.; Fox, M.; Bouquet, P.; Hüllermeier, E.; Dignum, V.; Dignum, F.; and van Harmelen, F., eds., *Proc. of the 22nd Eur. Conf. on Artificial Intelligence (ECAI 2016)*, volume 285 of *Frontiers in Artificial Intelligence and Applications*, 819–827. IOS Press.

Hendre, A., and Joshi, K. P. 2015. A semantic approach to cloud security and compliance. In *CLOUD*, 1081–1084. IEEE Computer Society.

Lutz, C.; Seylan, I.; and Wolter, F. 2013. Ontology-based data access with closed predicates is inherently intractable(sometimes). In *Proc. Int. Joint Conf. on Artificial Intelligence (IJCAI'2013)*, 1024–1030. IJCAI/AAAI.

Lutz, C.; Seylan, I.; and Wolter, F. 2019. The data complexity of ontology-mediated queries with closed predicates. *Logical Methods in Computer Science* 15(3).

Ngo, N.; Ortiz, M.; and Šimkus, M. 2016. Closed predicates in description logics: Results on combined complexity. In *Proc. Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR 2016)*, 237–246. AAAI Press.

Reiter, R. 1992. What should a database know? *J. Log. Program.* 14(1&2):127–153.

Tobies, S. 1999. A nexptime-complete description logic strictly contained in c^2. In *CSL*, volume 1683 of *Lecture Notes in Computer Science*, 292–306. Springer.

Vardi, M. Y. 1982. The complexity of relational query languages (extended abstract). In *STOC*, 137–146. ACM.