Thesis for The Degree of Doctor of Philosophy

Pre-deployment Description Logic-based Reasoning for Cloud Infrastructure Security

Claudia Cauli

Division of Computing Science
Department of Computer Science & Engineering
University of Gothenburg
Gothenburg, Sweden, 2022

Pre-deployment Description Logic-based Reasoning for Cloud Infrastructure Security

Claudia Cauli

ISBN 978-91-8009-839-7 (PRINT)
ISBN 978-91-8009-840-3 (PDF)
http://hdl.handle.net/2077/71634

Technical Report No 219D
Department of Computer Science & Engineering
Division of Computing Science
University of Gothenburg
Gothenburg, Sweden

Cover Image *"Reasoning about Actions"*
Illustration inspired by Example 9, p.14 of the Introduction.

This thesis has been prepared using LaTeX.
Printed by Chalmers Reproservice,
Gothenburg, Sweden 2022.

Abstract

Ensuring the security of a cloud application is exceptionally challenging. Not only is cloud infrastructure inherently complex, but also a precise definition of *what is secure* is hard to give. Business context, regulatory compliance, use cases, intent, and human interpretation influence this definition, and what is considered secure in one setting may not be in another. This thesis aims to improve the extent to which automated techniques support manual security reviews and, by doing so, to aid users of all levels in designing infrastructure compliant with their security standards. To achieve this objective, we investigate the application of provable techniques to security analyses amenable to early design phases. In particular, we study description logic-based semantic reasoning for the pre-deployment modeling and verification of cloud infrastructure.

The body of this thesis is based on three published papers. In the first paper, we encode Amazon Web Services CloudFormation deployment language into the expressive description logic $\mathcal{ALCOIQ}$. We verify configuration checks with ad-hoc reasoners and sketch an axiomatization of security knowledge to reason about system-level properties. We find that expressive logics can simulate partial closed-world reasoning, vulnerabilities, and mitigations to threats but trigger high complexity of the reasoning tasks and require cumbersome modeling. To overcome these, in the second paper, we define a novel lightweight logic and a query language for security threats. The logic mixes open-world and closed-world assumptions to succinctly encode complete and incomplete knowledge. The query language embeds optimistic and pessimistic reasoning to express *vulnerabilities that may be present* versus *mitigations that must be in place*. Lightweight logics enable tractability: knowledge base satisfiability and query answering become decidable in AC^0 and LogSpace data complexity, respectively. Lastly, in the third paper, we build on this new formalism by introducing a language to encode *mutating actions* (that create, delete, or modify cloud resources) and defining the transition system generated from an initial configuration when all possible actions are applied. In the transition system, states represent alternative configurations, and transitions represent changes induced by the actions. By focusing on the planning problem, we search for sequences of actions that mitigate the potential vulnerabilities of the initial configuration. Due to the practical decision procedures of the underlying formalism, we do so in PTime data complexity.

Keywords: Description Logic, Security, Cloud, Automated Reasoning, Formal Methods, Verification

Abstrakt

Att garantera säkerheten för en molnapplikation är exceptionellt svårt. Molninfrastruktur är inte bara i sig komplex, utan dessutom är det svårt att ge en exakt definition av *vad som är säkert*. Kontext, regelefterlevnad, användningsfall, avsikt, och mänsklig tolkning påverkar denna definition, och vad som anses säkert i en miljö kanske inte finns i en annan. Denna avhandling syftar till att förbättra i vilken utsträckning automatiserade tekniker stödjer manuella säkerhetsgranskningar och, som en konsekvens, att hjälpa användare på alla nivåer att designa infrastruktur som är säker enligt deras standarder. För att uppnå detta mål undersöker vi tillämpningen av verifierbara tekniker för säkerhetsanalyser som är lämplig för tidiga designfaser. Vi studerar speciellt *"description logic"*-baserade semantiska resonemang för modellering och verifiering av molninfrastruktur innan den implementeras.

Denna avhandling bygger på tre publicerade artiklar. I den första artikeln kodar vi Amazon Web Service CloudFormation språk till den uttrycksfulla logiken $\mathcal{ALCOIQ}$. Vi verifierar konfigurations kontroller med ad-hoc-verktyg och ger en axiomatisering av säkerhetskunskap för att resonera om systemnivåegenskaper. Vi finner att uttrycksfull logik kan simulera partiell slutenvärld resonemang, säkerhetssårbarhet, och säkerhetsbegränsningar men orsaka en hög komplexitet i resonemanget och kräver besvärlig modellering. För att övervinna dessa hinder, definierar vi i den andra artikeln en ny lättvikts logik och ett frågespråk för sårbarheter och begränsningar av säkerhetshot. Logiken kombinerar sluten och öppen världsantagande för att kortfattat koda fullständig respektive ofullständig kunskap. Frågespråket bäddar in optimistiska och pessimistiska resonemang för att uttrycka *sårbarheter som kan finnas* kontra *begränsningar som måste finnas på plats*. Lättviktslogik möjliggör praktiska algoritmer: kunskapsbas tillfredsställelse och frågesvar blir avgörande i AC^0 respektive LogSpace data komplexitet. Slutligen, i den tredje artikeln bygger vi vidare på denna nya formalism genom att introducera ett språk för att koda *muterande åtgärder* (som skapar, tar bort, eller modifierar nuvarande resurser) och definierar övergångssystemet som genereras från en initial konfiguration när alla möjliga åtgärder tillämpas. I övergångssystemet representerar stater alternativa konfigurationer, och övergångar representerar de förändringar som induceras av åtgärderna. Genom att fokusera på planeringsproblemet, söker vi efter sekvenser av åtgärder som mildrar sårbarheterna i startkonfigurationen. På grund av de praktiska algoritmerna för den använda logiken kan vi göra det i PTime data komplexitet.

Acknowledgments

During these Ph.D. years, I met many people who, directly or indirectly, supported me through this time. I want to thank them all.

Most importantly, I am very grateful to my advisor Nir Piterman for his continuous support, great deal of patience, kindness, and positivity. Without him, this thesis would not have been possible. Thanks for giving me the opportunity to do a Ph.D. with you and for investigating this research area with me. I hope you are proud of the results.

This work would not have been the same without Oksana Tkachuk, who hosted me during my internships at Amazon Web Services, believed in this research idea, and dedicated her time to our collaboration. I also want to express my gratitude to Byron Cook and Neha Rungta, who encouraged and enabled this joint research effort; and to my past and present colleagues in the Automated Reasoning Group for the insightful scientific discussions.

Many thanks also to my second supervisor, Gerardo Schneider, for his support during my time at the University of Gothenburg; and my internal committee, Dave Sands, Nir, Gerardo, and Agneta Nilsson, for the supportive environment during our regular follow-up meetings. I want to thank Agneta, Clara Oders, Nir, and Wolfgang Ahrendt for taking care of the graduate students at the CSE department. Your role is important. Please keep up the good work. A big thank you also goes to the co-authors of the papers included in this thesis, Magdalena Ortiz, Meng Li, and Oksana, who shared their wisdom on research theory and practice, helping me grow professionally. Thanks also to the thesis defense opponent, Piero Bonatti, and committee members Ana Ozaki, Meghyn Bienvenu, and Sebastian Rudolph for reviewing this thesis, providing feedback, and offering their availability for the defense.

I am also grateful to Rustan Leino and David Tedenstedt for their kind help with the Swedish version of the abstract; Andres Nötzli for assisting with the formatting; Norine Coenen, Nafi Diallo, and Pamela Andrade Sevillano for proofreading parts of the thesis; Nir and Shaun Azzopardi for handling the final bureaucracy. Importantly, I want to thank the many colleagues in the dept. of Informatics at the University of Leicester and the many colleagues in the Formal Methods and Information Security units at Chalmers and GU. I am lucky to have found friends in Ph.D. students from other universities: Hadar Frenkel, Norine, and Andres provided a sometimes much-needed new perspective.

A big thank you to all my friends across Italy, the UK, the US, and Sweden who have accompanied me through the bad and the good times. Last but not least, I owe my gratitude to my parents and family for being an example of unwavering integrity, resilience, and hard work.

List of Publications

Appended publications

This thesis is based on the following publications:

[A] C. Cauli, M. Li, N. Piterman, and O. Tkachuk.
"Pre-deployment Security Assessment for Cloud Services Through Semantic Reasoning"
Proceedings of the 33rd International Conference on Computer-Aided Verification (CAV), 2021.

[B] C. Cauli, M. Ortiz, and N. Piterman.
"Closed- and Open-world Reasoning in DL-Lite for Cloud Infrastructure Security"
Proceedings of the 18th International Conference on Principles of Knowledge Representation and Reasoning (KR), 2021.

[C] C. Cauli, M. Ortiz, and N. Piterman.
"Actions over Core-closed Knowledge Bases"
To appear in *Proceedings of the 11th International Joint Conference on Automated Reasoning (IJCAR), 2022.*

Other publications

The following work was published during my PhD studies, but is not appended due to unrelated content to the thesis.

(a) C. Cauli, N. Piterman.
"Equivalence of Probabilistic μ-Calculus and p-Automata"
Proceedings of the 22nd International Conference on Implementation and Application of Automata (CIAA), 2017.

Statement of Contribution

The following statement summarizes my personal contribution to the papers included in this thesis.

[A] Pre-Deployment Security Assessment for Cloud Services Through Semantic Reasoning

This paper was co-authored with Meng Li, Nir Piterman, and Oksana Tkachuk. The research idea of applying description logic-based reasoning to CloudFormation is mine and was conceived during an internship at Amazon Web Services. The formalization, encoding, and prototype implementation were carried out by myself. Writing is almost exclusively mine.

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

This paper was co-authored with Magdalena Ortiz and Nir Piterman. The idea behind core-closed knowledge bases and security queries is mine and motivated by the previous work. Results and proofs regarding core-closed knowledge bases are mostly mine, the remaining results and proofs are developed jointly. Writing is almost exclusively mine.

[C] Actions over Core-closed Knowledge Bases

This paper was co-authored with Magdalena Ortiz and Nir Piterman. The decision to incorporate reasoning about actions into the previously developed formalism is mine and motivated by practical considerations. Results and proofs regarding the static verification problem are developed jointly. Results and proofs regarding the planning problem are mostly mine. Writing is exclusively mine.

Contents

Chapter 1

Introduction

This thesis investigates description logic-based semantic reasoning for the pre-deployment modeling and verification of cloud infrastructure. The overarching goal is to improve the extent to which provable automated techniques support manual security reviews and, by doing so, to aid users of all levels in designing infrastructure that is guaranteed to be secure.

1.1 The Cloud

The term *cloud* is a general term to denote the on-demand availability of computer services and resources. Although initially limited to services such as data storage and computing power, over the last two decades, the cloud offering has grown far beyond and now includes a multitude of computing solutions. Amazon Web Services (AWS) began serving customers in 2006 with the introduction of the Simple Storage Service (S3) and the Elastic Compute Cloud (EC2). Companies such as Microsoft Azure, IBM Cloud, Oracle, and Google Cloud began their operations in the following years. Today, these companies offer services spanning quantum, satellite, robotics, virtual reality, and gaming solutions, with surveys reporting that 94% of all enterprises worldwide use some cloud service [1].

Cloud computing is made possible by *virtualization*. Virtualization is a software abstraction that allows dividing single physical computers into multiple virtual resources. The global physical infrastructure (computing resources and network) is geographically partitioned into *regions*. Regions are further organized in *availability zones*. Depending on the region and customer needs, availability zones can be further partitioned into smaller units, e.g., local zones and outposts. Such a hierarchical topology allows for high availability, fault tolerance, high bandwidth, low latency, redundancy, data residency, and scalability.

Users purchase virtual resources depending on their needs and efficiently scale as these grow, following a *pay-as-you-use* model. The process of purchasing and configuring instances of virtual cloud resources is called *deployment*. When users compose multiple resource instances into larger infrastructure, we say that they are "deploying a cloud infrastructure." For example, users could deploy a simple infrastructure constituted by a single storage instance to store

their data; or a more complex infrastructure with several storage instances, an API interface, a computing unit, and a database to allow for a more articulated set of features. Customers first subscribe to the cloud services by creating an *account* and configuring *users* to access it. The configuration of users requires the establishment of access credentials, which are needed for *authentication*. When a user—human or machine—logs into an account or service using its access credentials, we say that it assumes (or becomes) an *identity*. Identities are allowed (or denied) to perform operations within accounts in a process broadly referred to as *authorization*. Although there are multiple alternative ways to manage authorization, it is often recommended to use policies (either role-based or tag-based access control policies). These allow for fine-grained control over who has access to what and under what conditions. Actions grant different access *levels*, such as read, write, list, or management. Actions granting access to read, write, and list resources entail a *data flow* from one node to another within a cloud configuration. Actions granting write and management access—in addition to potentially allowing data to flow—allow users to modify the current cloud configuration and are thus called *mutating actions*.

1.1.1 Cloud Security

Designing a *secure* cloud-based application is exceptionally challenging. The challenges arise from multiple contributing factors, technical and non-technical. Not only is cloud infrastructure deployed at a large scale, complex, highly distributed, and communicating with an open (potentially malicious) environment, but also a precise definition of secure infrastructure is hard to characterize [2]. What is considered secure varies depending on business context, legislation, use case, human interpretation, and intent. These aspects are less understood and defined, almost fuzzy, than the more technical ones; and are hard to grasp with formal definitions. In addition to security, similar qualities like *privacy* and *regulatory compliance* must also be considered when designing and monitoring a cloud application. With the advent of the cloud, users of any expertise level can effortlessly build very complex and robust infrastructure, often having minimal understanding and awareness of all the factors described here that contribute to its security, privacy, and compliance.

According to the latest findings published by the *Open Web Application Security Project foundation* (commonly known as the OWASP foundation) in their "OWASP Top 10" 2021 report [3], the most severe web application security risks are attributable to missing, broken, or insufficient control mechanisms. These threats are prevalent and could be very simple to fix but are tremendously difficult to find. In particular, broken access control occupies the first position, followed (in order) by faulty encryption, injection attacks, insecure design, security misconfiguration, use of vulnerable and outdated components, broken authentication, data integrity failures, and faulty logging and monitoring. Especially relevant to this thesis is the *insecure design* category, added in 2021 and motivated by the need for a more substantial adoption of *threat modeling*, secure design patterns and principles, and reference architecture.

Threat Modeling Threat modeling helps identify, understand, and address potential exposure to security threats [4]. Threat modeling is advisable since the early stages of software development, best during the design phase, and requires a graph representation of the infrastructure, including diagrams of its data flows, control flows, authentication and authorization flows, and architectural components. A threat model usually includes *(1)* a model of the infrastructure, *(2)* a list of actors interacting with the system, *(3)* a list of the assumptions based on the system's functionality, *(4)* a list of all the valuable assets, and, most importantly, *(5)* a list of applicable threats. By working on this latter list, expert security engineers, architects, and developers address potential exposure to security risks before they can manifest. Although most vulnerabilities have a fix, some of them do not. Often, vulnerabilities cannot be completely eliminated, and the only available strategy is to reduce the probability of an exploit by applying mitigations. Sometimes, however, no mitigation is possible, and the probability of an exploit cannot be reduced. When this happens, the final decision is to accept the vulnerability, dynamically monitoring the system at runtime. Threat modeling is a time-consuming and error-prone activity, mainly done by manually analyzing the system architecture details.

Formal Methods for Security Cloud providers have successfully employed formal methods to verify their systems' functional correctness and reliability [5–8], and new research directions have been proposed to integrate automated threat modeling techniques in agile software development [9, 10]. There is certainly no technique to completely replace expert security engineers and users performing security reviews of cloud-based applications. However, integrating *formal automated threat modeling* techniques in the process could help. Such techniques need to be based on a precise language suitable to model and reason about the configuration of a cloud deployment, the open world surrounding it, common misconfigurations and known vulnerabilities, data handling, and critical components involved in authentication and authorization. In addition, it needs to be equipped with inference procedures to investigate and discover potential exposures (resp. existing mitigations) to threats and reason about access control permissions, flows, and use cases. Overall, such a technique should enable comprehensive exploration and semantic reasoning of the system's security.

1.1.2 Infrastructure as Code

The practice of purchasing remote, virtualized, and distributed infrastructure discussed in the previous subsections is generally known as *Infrastructure as a Service* (IaaS). Users and enterprises can provision, manage, and update their cloud infrastructure in several alternative ways. They can do it through a command-line interface, a web-based dashboard, programmatic API calls, or intermediate high-level declarative languages that hide low-level deployment instructions. The last method (i.e., managing infrastructure via high-level declaration files) is called *Infrastructure as Code* (IaC). All cloud providers offer tools to deploy resources on the cloud, including a (usually declarative) deployment language. Amazon Web Services offers CloudFormation, Microsoft Azure offers Resource Manager, IBM Cloud offers Cloud Schematics, and

Google Cloud offers Deployment Manager. In addition to these provider-specific frameworks, provider-independent IaC tools exist, which abstract out the vendors' specifics. Examples of provider-independent deployment tools are Puppet, Chef, Terraform, and open standards, such as Tosca [11]. Source code files are usually written in a standard data exchange format, such as XML, JSON, or YAML. The research on infrastructure as code deployments is growing (e.g., see [12, 13]), with particular focus on its security, for instance, code smells [14–16], testing [17, 18], and intra-update vulnerabilities [19].

1.1.2.1 Services and Resource Types

Cloud products are usually structured into services and resource types. A *service* is like an abstraction for the product to be deployed. Examples of services include storage, database, computing, and networking components. Services encompass a collection of *resource types* providing the structure for the actual objects that users create and manage. Resource types correspond to more granular definitions of the products, such as a single storage unit, an object of data in the storage, a table in a database, an item in a database, or a network interface. Resource types are accompanied by a list of *property types*. Intuitively, these correspond to specific settings that customers can configure for their resource instances.

Example 1 (Services and Resource Types)

The *Amazon Web Services* cloud provider offers a storage service called *Simple Storage Service*, abbreviated as S3. Within the S3 product category, users can deploy and manipulate several resource types, each being responsible for different functionality in the scope of the overall storage service. Examples of resource types that are available in S3 are Bucket, Object, BucketPolicy, and AccessPoint, to name a few. A *"bucket"* is a virtual storage unit, while an *"object"* represents an actual blob of data contained in the storage. Since resource type names can be used more than once across different services, to avoid ambiguity, the names are prefixed by the service they belong to; for example, by writing S3::Bucket or S3::Object.

1.1.2.2 Resource Types' IaC Specifications

Infrastructure as code frameworks come with *specification files* that define the resource types (and property types) supported by each service. These specification files are structured definitions of the particular resources deployable, written in a data exchange format such as JSON. The definition usually includes a list of properties that users can configure with their names, possibly nested subproperties, types, and allowed values for a given resource type. The following example gives an intuition of this.

Example 2 (Infrastructure as Code Specifications)

We consider Amazon Web Services infrastructure as code framework, *CloudFormation*, and the S3::Bucket resource type introduced in the previous example. Since the bucket is a unit of storage, the settings that users can configure for it include data encryption, replication, versioning, predefined access control, and settings related to logging and notifications.[1] In the interest of simplicity, let us assume that the CloudFormation service specification for the S3::Bucket resource type allowed for the configuration of only two properties: `AccessControl`, to be used to specify whether the resource is, e.g., *Public* or *Private*; and `LogDestination`, to be used as a pointer to another bucket instance where logs are stored. The code snippet below shows a simplified syntax of this example specification.

```
"ResourceType":
"AWS::S3::Bucket": {
  "Properties":{
    "AccessControl":  {
      "Type":  "String",
      "Required":  false }
    "LogDestination" : AWS::S3::Bucket,
    ...
    }
}
```

1.1.2.3 Deployments' IaC Configurations

Customers use specification files as part of their automated workflows and, in general, as a guide to writing their *configuration files*. Configuration files are code files representing the actual infrastructure that will be deployed. In these declaration files, users specify which resource *instances* they wish to deploy in their architecture and how these should be configured and composed together. The resource instances declared in a configuration must validate against the resource types defined in the specifications.

Example 3 (Infrastructure as Code Configuration)

Building on the previous example, let us now consider a user configuration with only one bucket instance. The instance is called `DataBucket` and is configured to push its logs onto another bucket, called `LogBucket`. `LogBucket` is not part of what is currently being deployed but is a resource already existing. This usage scenario is quite common, as users can refer to resources already deployed elsewhere, either in the same or in a different account, provided that access permissions are granted. No `AccessControl` attribute is given, which is allowed by the property being optional. This

[1] For a complete list of the S3::Bucket properties in CloudFormation, readers are invited to consult https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-properties-s3-bucket.html (Link valid as of April 2022).

configuration is summarized in the code snippet below.

```
"DataBucket": {
  "Type":  "AWS::S3::Bucket",
  "Properties":
    "LogDestination": "LogBucket"
}
```

What we have described is the essence of the Infrastructure as Code paradigm. Code deployments facilitate automation, modularity, management, and reusability. Cloud providers publish code snippets to guide customers through the services' functionalities; specific deployment patterns can be easily reused; many similar instances can be quickly created, either programmatically or by just copy-pasting their settings. Furthermore, users can share their solutions. A quick search on platforms such as GitHub, Stack Overflow, and Reddit for keywords (and tags) such as *"CloudFormation," "Azure Resource Manager,"* or *"Infrastructure as Code"* yields thousands of results. Unfortunately, though, these practical advantages undermine the system's security. What is considered an adequate level of security in a given usage scenario might not be in another. Keeping track of the security-related settings of hundreds of resources and their interactions could be challenging. New security concerns can arise, and established security properties can break once different modules are composed. These security concerns are real, and both inexperienced and advanced users are equally exposed.

1.2 Description Logics

The term description logics (DLs) denotes a *family* of logics suitable to describe knowledge [20]. Most of the logics in the description logic family are a subset of *first-order logic* (FOL) and are carefully crafted to provide good expressivity while at the same time maintaining decidability and offering practical decision procedures. Knowledge in DLs is encoded with respect to *individual*, *concept*, and *role* names. Despite the different terminology, these are equivalent notions to first-order logic constants, unary predicates, and binary predicates, respectively. Depending on the specific description logic considered, a different set of operators is used to combine atoms into more complex *expressions*. Expressions are then related through top-level *axioms* and *assertions*. Axioms are universally quantified statements, written in the form of inclusion axioms, used to encode constraints that are valid for all the objects in the domain of interpretation. Assertions are existential facts known about the domain of interpretation and are usually written in the form of concept and role assertions and equality and inequality assertions. Sets of axioms and assertions constitute the so-called description logic *knowledge base*.

Besides expressivity and modularity, another prominent feature of description logics is their well-defined logical semantics. Description logics are *modal logics*, therefore interpreted over models that are relational Kripke structures (labeled transition systems). Axioms and assertions in DL knowledge bases *constrain* all possible models. Thus description logic knowledge bases should be

seen as an *abstraction for a set of models* satisfying the axioms and assertions included in the knowledge base. To understand how this interpretation is made possible, we need first to understand assumptions such as the open-world assumption and the unique name assumption, and properties of the logic derivation system such as monotonicity [21]. Description logics adopt the *open-world assumption* (OWA), which states that all information that is not known to be true (or cannot be proven true) is unknown. The open-world assumption is widely adopted in semantic web techniques, as it better aligns with real-world reasoning where knowledge tends to be incomplete [22]. This is in contrast to the *closed-world assumption* (CWA) that interprets as false every statement that is not true and is usually the standard adopted in database reasoning. In some modeling scenarios, the two assumptions are needed simultaneously, as both complete and incomplete information coexists, and using either one may lead to unintuitive or unintended results [23]. By default, description logics do *not* make the *unique name assumption*. Two individuals with different names a and b are not assumed to be different elements within the underlying domain of interpretation. To make these different, one must explicitly add an inequality assertion $a \not\approx b$. Like the OWA case, dropping the unique name assumption could lead to unintuitive results. However, it can be a powerful tool when modeling uncertainty. Description logics belong to the so-called *monotonic logics*. Monotonicity is defined with respect to the entailment relation: introducing new facts or axioms does not falsify previously inferred conclusions, and, generally, valid arguments cannot be made invalid. Logics that do not have this feature are called *non-monotonic* logics (NMLs) or *defeasible* logics [24, 25]. Examples of logical operators making a logic non-monotonic are exceptions, default values, and introducing a priority of evaluation among axioms.

1.2.1 Expressive Description Logics

We now introduce the reader to description logics' syntax, semantics, and main decision procedures. We mainly concentrate on a simple DL called $\mathcal{ALC}$ (*Attributive Language with Complement*) and discuss additional, more expressive features as we progress.

1.2.1.1 Syntax

Let us start by fixing the basic alphabets from which complex concept expressions, axioms, and assertions are constructed in this logic. We define three sets: the set $\mathbf{C}$ of atomic concept names, the set $\mathbf{R}$ of atomic role names, and the set $\mathbf{I}$ of individual names. Valid $\mathcal{ALC}$ concept expressions include the empty concept $\bot$, representing a set containing no individuals; the universe concept $\top$, representing a set containing all individuals; a basic concept A from the alphabet $\mathbf{C}$; the negation of a complex concept expression $\neg C$; the disjunction of two complex concept expressions $C \sqcup D$; the conjunction of two complex concept expressions $C \sqcap D$; the existential restriction $\exists r.C$; and the universal restriction $\forall r.C$. These operators are summarized in the grammar below:

$$C, D ::= \bot \mid \top \mid A \mid \neg C \mid C \sqcup D \mid C \sqcap D \mid \exists r.C \mid \forall r.C$$

where A is an atomic concept in the set $\mathbf{C}$ and r is a role from the set $\mathbf{R}$. Complex concept expressions C, D are used to write *general concept inclusion*

axioms (GCIs) of the form $C \sqsubseteq D$. These state that the concept expression C is subsumed by the concept expression D and are used to establish subsumption hierarchies between concepts. General concept inclusion axioms are collected in the set $\mathcal{T}$, typically referred to as *TBox*, and constitute the *terminological knowledge* of the domain being modeled.

Example 4 (Specifications Modeling)

Encoding infrastructure as code specifications into terminological knowledge enables resource type inference and serves as a harness for semantic extensions of the resource specifications with domain knowledge (e.g., ontologies of security, environment, and data-flow terms). The sample S3::Bucket specification of Example 2 can be modeled through concept inclusion axioms as the TBox $\mathcal{T}_{\mathsf{S3}}$. To capture the connection that a bucket instance can establish with another bucket instance, we encode the range of the `LogDestination` property as follows:

$$\mathcal{T}_{\mathsf{S3}} = \{\ \top \sqsubseteq \forall \mathsf{logDestination.S3Bucket}\ \}$$

The axiom in $\mathcal{T}_{\mathsf{S3}}$ states that all nodes that are a target of the property `LogDestination` are inferred to be instances of the type S3::Bucket.

Facts about the individuals of the modeled domain are encoded using *concept assertions* of the form $C(a)$ and *role assertions* of the form $r(a, b)$. The former states that the individual a is an instance of concept C. The latter states that the relation r links the individuals' pair (a, b). Concept and role assertions are collected in the set $\mathcal{A}$, typically referred to as *ABox*, and constitute the *assertional knowledge* of the modeled domain. Finally, an $\mathcal{ALC}$ knowledge base $\mathcal{K}$ arises from the union of the terminological and assertional knowledge; that is, $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$.

Example 5 (Configurations Modeling)

We now discuss how the cloud configuration of Example 3 could be modeled in DL. Intuitively, the S3::Bucket instance `DataBucket` is encoded through the concept assertion $\mathsf{S3Bucket}(\mathit{DataBucket})$ and its `LogDestination` property is encoded through the role assertion $\mathsf{logDestination}(\mathit{DataBucket}, \mathit{LogBucket})$. These two assertions are added to the ABox $\mathcal{A}_{\mathsf{t}}$:

$$\mathcal{A}_{\mathsf{t}} = \{\ \mathsf{S3Bucket}(\mathit{DataBucket}),\ \mathsf{logDestination}(\mathit{DataBucket}, \mathit{LogBucket})\ \}$$

However, since we assume the information in the user configuration file to be complete and no `AccessControl` property has been declared, we may want to make the absence of this property explicit by adding the inclusion axiom $\{\mathit{DataBucket}\} \sqsubseteq \neg\exists \mathsf{accessControl}.\top$. This axiom states that the individual *DataBucket* certainly does not have *any* outgoing edge labeled by the property accessControl directed to *any* other domain node. It is needed to enforce a closed-world interpretation over the missing property, resolving an *unknown* to a *false*. Therefore, the encoding of this cloud

configuration also produces the TBox:

$$\mathcal{T}_{\mathsf{t}} = \{\ \{\mathit{DataBucket}\} \sqsubseteq \neg\exists \mathsf{accessControl}.\top\ \}$$

The assertions and axioms presented here and in the previous example constitute the knowledge base $\mathcal{K} = \langle \mathcal{T}_{\mathsf{S3}} \cup \mathcal{T}_{\mathsf{t}},\ \mathcal{A}_{\mathsf{t}} \rangle$, which encompasses all the known information about this sample system.

By adding operators to the logic $\mathcal{ALC}$, one can obtain more expressive description logics. The logic $\mathcal{ALCOIQ}$ is obtained from $\mathcal{ALC}$ by adding three main features: *nominals*, *inverse roles*, and *quantified restrictions*, symbolized by the letters $\mathcal{O}$, $\mathcal{I}$, and $\mathcal{Q}$, respectively, in the logic name signature. Nominals are denoted with the syntax $\{a\}$ and represent special singleton sets containing only one individual, the individual a. Inverse roles are denoted with the syntax r^- and represent the binary relation obtained by inverting the position of the individuals a and b in a pair (a, b) belonging to relation r. Quantified restrictions are denoted with the syntax $_{\geq n}r.C$ and $_{\leq n}r.C$ and represent the existence of a number of r-successors in set C that is either greater than n or less than n. In even more expressive description logics, subsumption hierarchies can be constructed starting from *complex role expressions*, in addition to complex concept expressions. Examples of complex role expressions are the inverse role r^-, which we already discussed, and *role chains* of the form $r \circ r'$ where two binary relations r and r' are composed in sequence forming a new relation. Using role expressions within inclusion axioms allows one to increase the expressivity of the logic enormously, for example, allowing one to encode transitivity as the axiom $r \circ r \sqsubseteq r$ and therefore adding the capability to reason about reachability within the graph entailed by the models of a knowledge base. A notable example of such a description logic is the *irresistible* $\mathcal{SROIQ}$ [26].

1.2.1.2 Semantics

Description logics have a model-theoretic semantic that precisely specifies the implications of the axioms and assertions of a knowledge base $\mathcal{K}$ [27]. The semantic is defined over an interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, consisting of a set $\Delta^{\mathcal{I}}$ and an interpretation function $\cdot^{\mathcal{I}}$. The set $\Delta^{\mathcal{I}}$ is the domain of the interpretation. The interpretation function $\cdot^{\mathcal{I}}$ maps each atomic concept A to a set $A^{\mathcal{I}}$ subset of $\Delta^{\mathcal{I}}$; each atomic role r to a binary relation $r^{\mathcal{I}}$ subset of $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$; and each individual name a to an element $a^{\mathcal{I}}$ in the set $\Delta^{\mathcal{I}}$. The interpretation is extended to complex concept expressions and axioms in a predictable way. In particular, we note that the interpretation of inverse roles and nominals is given by the following set expressions: $(r^-)^{\mathcal{I}} = \{(x, y) \mid (y, x) \in r^{\mathcal{I}}\}$ and $\{a\}^{\mathcal{I}} = \{a^{\mathcal{I}}\}$. The semantics of top-level inclusion axioms of the form $C \sqsubseteq D$ is given by $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$, and that of concept assertions $C(a)$ and role assertions $r(a, b)$ is given by the two statements $a^{\mathcal{I}} \in C^{\mathcal{I}}$ and $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in r^{\mathcal{I}}$. When all axioms and assertions in a knowledge base $\mathcal{K}$ hold in an interpretation $\mathcal{I}$, then we say that $\mathcal{I}$ *is a model of* $\mathcal{K}$. A knowledge base has multiple potential models, all satisfying all the axioms and assertions that it contains.

Example 6 (Interpretation)

We now reflect on the possible interpretations that model the knowledge base $\mathcal{K}$ from Example 5. In particular, the following logical statements are true for all interpretations $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ that model $\mathcal{K}$:

(1) $DataBucket^{\mathcal{I}} \in \mathsf{S3Bucket}^{\mathcal{I}}$

(2) $LogBucket^{\mathcal{I}} \in \mathsf{S3Bucket}^{\mathcal{I}}$

(3) $(DataBucket^{\mathcal{I}}, LogBucket^{\mathcal{I}}) \in \mathsf{logDestination}^{\mathcal{I}}$

(4) $\forall y \in \Delta^{\mathcal{I}} \,.\, (DataBucket^{\mathcal{I}}, y) \notin \mathsf{accessControl}^{\mathcal{I}}$

Where *(2)* and *(4)* are inferred from TBox axioms and not introduced as ABox assertions. More precisely, *(2)* is entailed by the encoding of the service specification—the inclusion axiom in $\mathcal{T}_{\mathsf{S3}}$, which states that the property `LogDestination` points to an object of type S3::Bucket. Whereas, *(4)* is a consequence of the inclusion axiom in $\mathcal{T}_{\mathsf{t}}$, which allows us to model that the bucket `DataBucket` certainly does not have any value under the property `AccessControl`.

1.2.1.3 Decision Problems

Some relevant decision problems when doing description logic reasoning are consistency, satisfiability, and instance checking [28]. A TBox $\mathcal{T}$ (resp. ABox $\mathcal{A}$) is said to be *consistent* iff there exists a model satisfying all axioms in $\mathcal{T}$ (resp. all assertions in $\mathcal{A}$), written $\mathcal{I} \models \mathcal{T}$ (resp. $\mathcal{I} \models \mathcal{A}$). A knowledge base $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ is *consistent* iff it has a model $\mathcal{I}$ such that $\mathcal{I} \models \mathcal{T}$ *and* $\mathcal{I} \models \mathcal{A}$. A concept C is *satisfiable* with respect to $\mathcal{K}$ iff there exists a model $\mathcal{I}$ of $\mathcal{K}$ with $C^{\mathcal{I}} \neq \emptyset$. An individual a is an *instance* of concept C with respect to $\mathcal{K}$ iff $a^{\mathcal{I}} \in C^{\mathcal{I}}$ holds for all models $\mathcal{I}$ of $\mathcal{K}$.

Adding to Example 6, it is easy to see that the ABox $\mathcal{A}_{\mathsf{t}}$ is consistent with respect to $\mathcal{T}_{\mathsf{t}}$ *and* $\mathcal{T}_{\mathsf{S3}}$; that the concept $\mathsf{S3Bucket} \sqcap \exists \mathsf{accessControl}.\top$ is satisfiable with respect to $\mathcal{K}$; and that *LogBucket* is an instance of S3Bucket.

1.2.2 Lightweight Description Logics

Lightweight description logics is a term used to denote smaller fragments of the description logic family, characterized by the low complexity of the standard decision procedures. Notable examples of lightweight DLs are the logics $\mathcal{EL}$++ [29] and the family of logics under the DL-*Lite* umbrella [30, 31].

The logic $\mathcal{EL}$++ retains useful modeling features such as existential quantifiers, conjunction, disjoint concepts, general concept inclusion axioms, complex role chains, and transitive relations while at the same time admitting sound and complete reasoning in polynomial time. The DL-*Lite* family of logics is also particularly appealing due to its low computational complexity and high expressive power as a conceptual modeling formalism. DL-*Lite* forms the basis of OWL 2 QL,[2] one of the three profiles of OWL 2,[3] which is the logic of choice when modeling conceptual diagrams, UML class diagrams, and ER diagrams.

[2] https://www.w3.org/TR/owl2-profiles/#OWL_2_QL
[3] https://www.w3.org/TR/owl2-overview/

In Paper B, we provide an alternative definition of DL-*Lite* description logic knowledge bases mixing open- and closed-world reasoning. We then reuse this definition in Paper C to introduce temporal and action reasoning. Using a lightweight DL allows us to keep reasoning tractable. In particular, satisfiability becomes decidable in AC^0 in data complexity (the size of the actual, variable, data) and optimistic/pessimistic query answering is done in LogSpace in data complexity.

1.3 Automated Reasoning for Cloud Security

This section aims to emphasize the automated reasoning techniques that have been instrumental to this thesis research. Verifying the security properties of cloud deployments requires reasoning about uncertain environments and incomplete models, the semantic meaning of a given configuration, exposure to (and mitigation of) possible threats, and potential changes that could damage or improve the deployments' overall security.

1.3.1 Incomplete Knowledge and the Open-World

Secure design of cloud deployments requires reasoning about potentially incomplete models, which may interact with an unknown open environment. Especially at the early stages of development, models can be patchy, inconsistent, or non-homogeneously defined, and their API interfaces may still be unclear. When reasoning about infrastructure as code deployments, one must simultaneously reason about nodes that are declared *inside* a given template and nodes that exist *outside* of it (let us recall the difference between the *DataBucket* and the *LogBucket* objects from Example 5). In particular, we assume that the knowledge we have for nodes in the scope of a given configuration file is *complete* and that the knowledge we have for nodes outside is *incomplete*. Also, we could consider the case where the deployment template is unfinished, with some of its resource instances still unspecified. In both cases, we need to reason about *known resources with unknown properties* and about *generally unknown resources*, as both are important when evaluating the security of an infrastructure and its potentially adversarial users. Within the area of description logic reasoning, most of the approaches that combine DLs' intrinsic open-world semantics with local closed-world reasoning (e.g., [23, 32]) adopt a *database* view. According to the usual database view, some predicates (concepts or roles) are interpreted with the closed semantics; that is, their interpretation coincides with the assertions in the ABox. In Paper B, we introduced a distinction between *configuration* and *external* nodes. Whether an interpretation follows an open- or closed-world semantics is decided more granularly over constants (individuals) rather than bulk predicates. A convenient alternative way to achieve a combination of open- and closed-world reasoning would be through epistemic logics and epistemic operators [22, 33, 34]. Epistemic concepts are interpreted as the set of individuals *known* to belong to the concept in all possible worlds.

1.3.2 Semantic Reasoning

Extending the knowledge formalized in the models of the cloud specifications could enable a more complex semantic reasoning on the security of a cloud configuration. More expressive description logics, e.g., $\mathcal{SROIQ}$, could be used to reason about *subsumption hierarchies*, *chains* of relations composed together, and *transitivity* of relations. This expressive power is needed to reason about the complex reachability properties of the underlying infrastructure graph. For example, one could write an ontology of terms that relate general higher-level concepts of security, logging, monitoring, and alerting to the specific services, resource types, and property types of the IaC specifications, effectively adding meaning to configuration files. Such an ontology could become part of a more extensive semantic model of threat modeling and security, which could be integrated into a tool to reason about the security of a cloud configuration automatically and potentially block deployment if predetermined security requirements are not met. In Paper A, we include a proof-of-concept example of the advantages that such an approach could bring in terms of reasoning about system-level security properties.

1.3.3 Querying for Vulnerabilities and Mitigations

In description logic, the outcome of querying a knowledge base with a formula representing a concept expression is the set of its *certain answers*, i.e., the set of individuals that are proven to be instances of that complex concept in all the interpretations that are model of the knowledge base. This semantics is well-suited for expressing facts that *must necessarily* be true in all worlds. Trivially, these include those that are asserted, that is, the knowledge that we explicitly included in the model. The following example shows how querying for certain answers could be helpful to express mitigations to threats that are certainly in place.

Example 7 (Mitigations that *must* be in place)

Let us imagine that we aim to identify all the *"buckets that certainly keep logs"* since maintaining logs is the most fundamental mitigation to threats in the *repudiation* family. Such a check could be done by querying the knowledge base $\mathcal{K}$ encoding of the deployment configuration with the complex concept expression $\mathsf{S3Bucket} \sqcap \exists \mathsf{logDestination}.\top$, and retrieving the set of individuals that are instances of it. By looking at the interpretation discussed in Example 6, it is easy to see that the set of instances known to *certainly* satisfy this concept is $\{DataBucket\}$.

Sometimes, in contrast with what was discussed above, we need to express facts that *may* be true in some worlds. These include facts that are not asserted and cannot be inferred but are consistent with respect to the terminological knowledge of the system. Using the usual open-world semantics adopted in description logics, however, there is no direct way to express similar properties and identify the individuals that are instances of these. As shown in the next example, to achieve such an outcome, one could choose between two approaches:

either adding a layer of additional logic on top of the query answering process, or modifying the internal interpretation of the logic in a way that enables some form of pessimistic and optimistic reasoning.

Example 8 (Vulnerabilities that *may* be present)

We now aim to identify all the *"buckets that may be public"*, as that could represent a threat to the *confidentiality* of the system. One way to obtain this information is to query all the buckets that are known to be *not* public and complement such set. As a consequence of the axioms introduced in Example 5, and as highlighted in Example 6, we infer that $DataBucket$ is the only individual known to be not public, and that the set of known buckets is $\{DataBucket, LogBucket\}$. Therefore, the only bucket that *may* be public is $LogBucket$, which is pointed at by the infrastructure being created, but is not part of it.

In Paper A of this thesis, we tackle this issue by taking the first approach: we classify properties in four types, each with its dedicated interpretation of the outcome of the satisfiability and instance retrieval checks and with a different pass or fail logic. In Paper B of this thesis, we tackle this issue by taking the second approach: we define a new type of knowledge bases, which we call *core-closed knowledge bases*. Their notion of *full satisfiability*, introduced in Paper C, generalizes satisfiability and model checking. Core-closed knowledge bases can be queried with a language for vulnerabilities and mitigations capable of alternating between certain and possible answers within the scope of one query.

1.3.4 Reasoning about Actions and Updates

As discussed at the beginning of this introductory chapter, access control policies allow or deny identities' permissions to perform actions over given resources. Actions can either entail a data flow between resources or mutate the structural configuration of a cloud deployment. Extending the reasoning to include these two types of actions has practical applications. First, reasoning about actions that enable data flow (read and write actions such as those starting with `get*`, `put*`, and `list*`) aids information-flow analyses that guarantee system-level properties such as confidentiality and integrity. Second, reasoning about mutating actions (management actions such as those starting with `create*`, `delete*`, `modify*`) helps prevent changes that would introduce vulnerabilities or drive complex structural repairs to mitigate exposure to security threats. In Paper C, we incorporate action and temporal reasoning into the core-closed knowledge bases previously introduced in Paper B. Reasoning now becomes two-dimensional. Our models are now transition systems. Each state is a different core-closed knowledge base, and the transitions between states are labeled by mutating actions. Inside each state, we perform semantic reasoning about a snapshot in time of the cloud deployment and query it for mitigations and vulnerabilities to security threats. By moving along transitions, we reason about potential changes caused by the execution of a mutating action. This framework enables us to find sequences of actions that, when applied on the

cloud deployment, would lead it to a state satisfying specific security properties, therefore driving structural repair, or to a state removing security controls, therefore to be avoided.

Example 9 (Action Reasoning)

Let us consider a knowledge base with the following assertions:

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

and the set of mutating action labels Act defined as

$\{\ \mathsf{deleteBucket}, \mathsf{createBucket}, \mathsf{deleteKey}, \mathsf{createKey},$
$\mathsf{enableKeyRotation}, \mathsf{putBucketEncryption}, \mathsf{deleteBucketEncryption}\ \}.$

Let us assume that we are interested in verifying the existence of a sequence of actions that when executed would lead the system to a state where the storage bucket node b is encrypted with a *rotating* key. Formally, this is equivalent to checking the existence of a *plan* (a minimal sequence of actions) that, when executed on the *transition system* arising from the combination of the knowledge base and the action set, leads to a state where the query *"bucket must have a bucket encryption configuration with a rotating key"* is satisfied on the bucket object b. It is easy to see that the following three sequences of actions are valid plans to reach the goal property.

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

The example is borrowed from Paper C, where we combine our novel definition of core-closed knowledge bases and a language for mutating actions into an overall *generated transition system*. The resulting transition system enables reasoning about actions and temporal logic formulas.

1.4 Summary of the Contributions

This thesis consists of three papers. Papers A and B have been published in peer-reviewed conferences, CAV 2021 and KR 2021, respectively. Paper C has been accepted to IJCAR 2022, also a peer-reviewed conference, and will be published in the proceedings soon. The purpose of this section is to outline the contributions of each paper.

1.4.1 Paper A

Pre-Deployment Security Assessment for Cloud Services Through Semantic Reasoning

Claudia Cauli, Meng Li, Nir Piterman, and Oksana Tkachuk

This paper presents a general methodology for the formal modeling and security analysis of cloud deployment configuration files. The focus is on the Amazon Web Services deployment framework, AWS CloudFormation. The paper proposes an automated technique to map CloudFormation templates and security domain knowledge into description logic. The conceptual contribution is that the open-world assumption makes description logic a good candidate to model the underlying uncertainty of this scenario. The global service specifications and the user deployment configurations are modeled separately, as terminological and assertional knowledge, respectively. The first contribution is the formalization of *(i)* the resources' type system enforced by the specifications and *(ii)* the graph structure entailed by the configuration files. These formal objects are then encoded into logical assertions and axioms in the expressive description logic $\mathcal{ALCOIQ}$. The second contribution is a prototype tool implementation. The tool embeds a collection of widely-adopted security best practices, encoded as DL-concept queries, and evaluates them against real-world CloudFormation templates publicly available on GitHub. The experimental evaluation shows that the approach is feasible and detects numerous violations of security best practices. As an additional proof-of-concept of the value delivered by description logic-based semantic reasoning approaches, the models are then extended with ontologies of more general cloud and dataflow domain knowledge. By reasoning on these extended models, we check system-level properties related to the higher-level business logic and reachability within the cloud deployments.

The novelty of this work is applying description logic-based semantic reasoning to the verification of the security of cloud configurations before deployment. However, this study also highlights the need for a logical formalism that is better tailored to encode cloud deployment specifications and configurations, a query language to identify security issues and their mitigations, and efficient decision procedures for reasoning and query answering.

Statement of Contribution This paper was co-authored with Meng Li, Nir Piterman, and Oksana Tkachuk. The research idea of applying description logic-based reasoning to CloudFormation is mine and was conceived during an internship at Amazon Web Services. The formalization, encoding, and prototype implementation were carried out by myself. Writing is almost exclusively mine.

Appeared in: Proceedings of the 33rd International Conference on Computer-Aided Verification (CAV), Virtual Event, July 20-23, 2021.

1.4.2 Paper B

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

Claudia Cauli, Magdalena Ortiz, and Nir Piterman

This paper addresses the theoretical challenge of finding a suitable description logic formalism to efficiently encode, reason, and answer security queries on cloud deployment configuration files. The two main contributions introduced are: first, a richer definition of knowledge base that we now call *core-closed* knowledge base, and second, a query language for vulnerabilities and mitigations to security threats. Core-closed knowledge bases combine a core system (to be interpreted under the closed-world assumption) and a standard knowledge base (to be interpreted under the usual open-world assumption). Predicates and objects are classified as either open or partially closed. The core is used to model the system being formalized as a collection of description logic axioms and assertions. The peculiarity is that assertions referring to closed objects and partially closed predicates are fixed and not influenced by inference reasoning. Core-closed knowledge bases written in a lightweight description logic, such as DL-$Lite^{\mathcal{F}}$, inherit the nice computational properties of standard knowledge bases written in lightweight DLs. Satisfiability and conjunctive query answering are in AC^0 in data complexity. Satisfiability of asserted conjunctive queries is in LogSpace in data complexity. The query language for vulnerabilities and mitigations to security threats allows for the boolean combination of brave and cautious reasoning applied to standard conjunctive queries, which are then reduced to query entailment and satisfiability, respectively.

The novelty of this work lies in the unique combination of closed- and open-world reasoning—through predicates that are closed only over a subset of the domain but open otherwise—while at the same time ensuring tractability in data complexity of the main decision procedures.

Statement of Contribution This paper was co-authored with Magdalena Ortiz and Nir Piterman. The idea behind core-closed knowledge bases and security queries is mine and motivated by previous work. Results and proofs regarding core-closed knowledge bases are mostly mine, the remaining results and proofs are developed jointly. Writing is almost exclusively mine.

Appeared in: Proceedings of the 18th International Conference on Principles of Knowledge Representation and Reasoning (KR), Virtual Event, November 3-12, 2021.

1.4.3 Paper C

Actions over Core-closed Knowledge Bases

Claudia Cauli, Magdalena Ortiz, Nir Piterman

This paper addresses the problem of incorporating action reasoning and temporal reasoning into core-closed knowledge bases. The main contribution is the definition of an action language to encode *mutating* actions. Mutating actions are actions that modify the structural configuration of a deployment by adding, modifying, or deleting existing resources. Once an action language with formal syntax and semantics is defined, we concentrate on two distinct problems: static verification and planning. The *static verification* problem, defined in the context of core-closed knowledge bases and mutating actions, answers whether the application of an action, no matter the parameters passed to it, always preserves the given properties of the system. The *planning* problem, which we instantiate in our setting for both the plan existence and the plan synthesis formulations, answers whether there exists a sequence of actions leading the configuration to a state satisfying a preset property; and, if so, computes it. Both the static verification and the planning problems are relevant in practice, as they could help predict the impact of potential changes and aid complex structural repairs.

The main novelty of this work lies in the definition of theoretical tools to reason about cloud deployments' change and repair, and how these impact their security posture. Specifically in defining the syntax and semantics for a formal action language to encode mutating actions; and providing decision procedures for the two problems of static verification and planning in the context of core-closed knowledge bases and the above-mentioned action formalism.

Statement of Contribution This paper was co-authored with Magdalena Ortiz and Nir Piterman. The decision to incorporate reasoning about actions into the previously developed formalism is mine and motivated by practical considerations. Results and proofs regarding the static verification problem are developed jointly. Results and proofs regarding the planning problem are mostly mine. Writing is exclusively mine.

To Appear in: Proceedings of the 11th International Joint Conference on Automated Reasoning (IJCAR), Haifa, August 7-12, 2022.

1.5 Discussion

In this section, we compare and contrast description logic-based approaches to other logic-based formalisms and tools.

Knowledge Representation Formalisms Description logics originally developed from frameworks such as semantic networks, concept languages, and frame-based systems; and later matured into languages with precise formal semantics. From this perspective, DLs can be viewed as a merging of two trends: *knowledge representation* systems, on the one hand, and *formal logic*, on the other hand.

First-Order and Modal Logic Most description logics are a subset of *first-order logic*, specifically the *two-variable guarded fragment* of first-order logic [35], denoted as GFO^2, occasionally extended with more expressive features such as

counting quantifiers and transitive closure. Description logics are also related to *modal logic*. The logic $\mathcal{ALC}$, presented in this introduction, has been proven to be a notational variant of the modal logic **K** [36]. The logic $\mathcal{ALC}$ extended with transitive closure, union, and composition of binary relations (denoted as $\mathcal{ALC}_{reg}$) has been proven to be a notational variant of PDL [37], and when extended with inverse relations ($\mathcal{ALCI}_{reg}$) becomes a syntactical variant of *converse*-PDL. Differently from full FOL, reasoning in description logic *is decidable*. All logics included in the DL family, from the more lightweight ones to the more expressive ones, have been carefully crafted to maintain decidability while ensuring good practical expressive power and, often, tractability of the main decision procedures. By pushing the boundaries of the operators included in known description logics, new insights are generally gained on the *decidable fragments* of *first-order logic* [38].

Expressivity and Scalability The fundamental idea behind the description logic machinery and its numerous fragments is to give users the possibility to choose a trade-off between expressivity and scalability that suits their needs. As a consequence of their nice computational properties (in terms of decidability, expressivity, scalability, and availability of several different tractable fragments), description logics have been chosen as the logical underpinning for the semantic Web; in particular, for the *Web Ontology Language* (OWL). In its current version, the Web Ontology Language is based on the expressive description logic $\mathcal{SROIQ}(D)$ and includes three main profiles, roughly corresponding to $\mathcal{EL}$++, DL-*Lite*, and DLs extended with rule-based reasoning, respectively.

Datalog Datalog [39] is a declarative logic programming language whose programs consist of facts and rules. In standard Datalog, the latter are restricted *Horn clauses*, allowing conjunction in the rules' body and disallowing negation in the rules' head. Some level of semantic reasoning analysis, similar—but not equivalent—to what was done in Paper A, could be performed by encoding designs' specifications and higher-level security knowledge as Datalog programs. However, such an encoding would not have enabled us to reason about incomplete models and unknown nodes out-of-the-box (without a bespoke axiomatization). More importantly, it would not have enabled us to obtain the results that we achieved in Papers B and C, where we define a decidable formalism mixing open- and closed-world reasoning, optimistic and pessimistic querying, based on lightweight DLs, with low computational complexity, and equipped with decision procedures for action reasoning, verification, and planning. With this respect, the main reason why Datalog differs from description logic lies in its underlying closed-world assumption and its inability to reason about anonymous objects in the domain of interpretation [40].

Alloy Alloy [41] is a declarative object-oriented specification language based on first-order logic supporting n-ary relations, quantifiers, facts, predicates, functions, and subtyping. Models are expressed in *relational logic*, and then encoded into boolean formulas to be solved by a SAT solver. Relational logic reasons about sets of objects and uses n-ary relations to model general facts about these sets (without explicitly having to enumerate their elements) and to refer to the existence of objects satisfying a given condition (without saying which objects these are). From this perspective, Alloy would allow us

to overcome some of the limitations of Datalog related to the assumptions made on the openness of the domain of interpretation. In addition, the latest release (Alloy 6) now supports full model checking, backed by NuSVM [42] and nuXmv [43], and temporal logic keywords, thus being well-suited for modeling both *structural* and *behavioral* aspects. These features could make Alloy a good candidate for all the automated reasoning tasks that we discussed in the previous section; however, we have not yet investigated its usage in the context of the application analyzed in this thesis.

Three-Valued Logic and Model Checking Three-valued models, such as partial Kripke structures [44], Modal Transition Systems [45], and Kripke Modal Transition Systems (KMTS) [46], provide different degrees of abstractions for the set of all interpretations that satisfy the constraints set by the models. In KMTSs, this is allowed by a state labeling function that ranges over the three values *true*, *false*, and *unknown*, and by the existence of two transition relations: *may* and *must*. Generalized model checking (GMC) checks whether there exists a concrete implementation satisfying the abstraction represented by the model and a temporal formula. When the models are fully defined, GMC reduces to model checking. When the models are fully undefined, GMC reduces to satisfiability. Our core-closed knowledge bases, from Paper B, work in a similar fashion. We conjecture that it would be possible to encode cloud deployment and security domain knowledge into such formalisms and reason about vulnerabilities and mitigations to security threats with three-valued temporal logics.

1.6 Conclusion

In this thesis, we have explored description logic-based semantic techniques to reason about the security posture of cloud deployment configurations. We concentrated on methods enabling a combination of open- and closed-world reasoning as tools to model unknown and incomplete aspects of the designs.

We started by leveraging existing tools to encode cloud deployments and security domain knowledge into expressive description logic as our first step. We found this to be effective as a fully automated method to assess the security posture of cloud services pre-deployment and decided to pursue it further. However, we also concluded that the modeling effort required to achieve the desired mixed open- and closed-world interpretation was excessive and computationally disadvantageous. Then, we decided to limit the expressive power of the logic used to that of lightweight description logic, and embed the mixed interpretation into their syntax and semantics instead. We ended up with a formalism that more elegantly achieves the intended interpretation of our first study, with much lower computational overhead. Finally, we extended this formalism to support, in addition to reasoning about configuration queries, reasoning about the impact of updates on the system's overall security. To this end, we developed a formal language for mutating actions and decision procedures for static verification and planning over the transition system induced by the actions' execution.

The big absent in our final picture is *data flow*. We did not investigate

the integration of actions that entail a flow of data between nodes in a cloud deployment. Reasoning about actions entailing a flow of data is paramount to establishing confidentiality, integrity, and trust boundaries. The invocation of these actions impacts security either within a snapshot in time of the deployment or across several consecutive snapshots that are reachable as a consequence of changes to the initial configuration. Differently from mutating actions, these would need to be included inside the knowledge base and not at the level of transitions. We leave this for future work.

Chapter 2

Paper A

This chapter is based on

Pre-deployment Security Assessment for Cloud Services Through Semantic Reasoning,

written by

Claudia Cauli, Meng Li, Nir Piterman, and Oksana Tkachuk,

published in

Proceedings of the 33rd International Conference on Computer-Aided Verification (CAV), 2021.

Abstract

Over the past ten years, the adoption of cloud services has grown rapidly, leading to the introduction of automated deployment tools to address the scale and complexity of the infrastructure companies and users deploy. Without the aid of automation, ensuring the security of an ever-increasing number of deployments becomes more and more challenging. To the best of our knowledge, no formal automated technique currently exists to verify cloud deployments during the design phase. In this case study, we show that Description Logic modeling and inference capabilities can be used to improve the safety of cloud configurations. We focus on the Amazon Web Services (AWS) proprietary declarative language, CloudFormation, and develop a tool to encode template files into logic. We query the resulting models with properties related to security posture and report on our findings. By extending the models with dataflow-specific knowledge, we use more comprehensive semantic reasoning to further support security reviews. When applying the developed toolchain to publicly available deployment files, we find numerous violations of widely-recognized security best practices, which suggests that streamlining the methodologies developed for this case study would be beneficial.

2.1 Introduction

The term *Infrastructure as Code* (*IaC*) refers to the practice of configuring, provisioning, and updating systems resources from source code files, which are compiled into atomic instructions and then executed to deploy the desired architecture [47]. The advantage of handling code, instead of manually provisioning resources, lies in the capability to use version control systems, orchestration frameworks, and automated testing tools as part of the deployment process. In addition to instructions relevant for resource creation, dependencies, and updates, IaC configuration files contain information about settings, dataflow, and access control. In a time when cloud companies provide customers with simple-to-launch, albeit extremely powerful infrastructure, it is crucial to automatically and provably verify the security of such systems.

In this study, we investigate IaC deployment frameworks and how these are formally modeled and reasoned upon. We explore the usage of description logics (DLs) as a conceptual-modeling formalism that is expressive, decidable, and equipped with mature tooling. We argue that formal reasoning techniques applied to deployment templates are an immensely valuable tool for developers and security engineers by substantially aiding the automation of time-consuming security reviews; helping them to detect complex logical errors at earlier stages; and, containing the costs that finding and fixing security issues at later stages would cause. As the prevalence of cloud infrastructure increases, in addition to experts, automated reasoning tools could benefit inexperienced users as well.

System Studied. We focus on the Amazon Web Services proprietary IaC tool, CloudFormation, the first to be introduced at a large scale, over ten years ago. AWS, cloud provider within Amazon, serves millions of customers worldwide. These include private businesses as well as government, education, nonprofit, and healthcare organizations. While the cloud provider is responsible for the faithful deployment of the customers' desired configurations, it is the customer's duty to make sure that these comply with the security requirements of their business context. Few management tools of this scale exist. Notable mentions are Terraform [48], Microsoft Azure's *Resource Manager* [49], Google Cloud's *Deployment Manager* [50], and the recently introduced OASIS standard TOSCA [11].

Goal of Study. Our goal is to improve the quality of the security analyses that are performed over IaC configurations pre-deployment; and by doing so, their overall security. With this study, we investigate the application of description logics to the formalization and reasoning over IaC deployments. In particular, we are interested in three aspects: *(i)* whether proposed cloud configurations comply with security best practices, *(ii)* how to aid customers in building more secure infrastructure *before* deploying it, and *(iii)* to what extent formal automated techniques can support manual pre-deployment security reviews.

Challenges. Little research has been done so far on the possibility to formalize IaC languages, and no research has been done to devise a logic that is well-suited to reason about cloud infrastructure. By nature, cloud infrastructure interacts with an open environment that is, at best, only partially known. In particular, external-facing APIs and users participate in these interactions. By design, cloud services allow for the composition of smaller components into

large infrastructure, the complexity of which creates a challenge with respect to security. Our models should capture the connectivity of resources, the flow of information that spans across multiple paths, and the rich security-related data available in IaC configuration files. This is further complicated by the need for a query language for verification and falsification, able to express that mitigations must be present (vs. may be absent), and security issues must be absent (vs. may be present). Importantly, we need practical tools that support the implementation of all these parts and that can scale to real-world IaC configurations.

Our Contribution. We provide a framework to encode IaC into description logic, and investigate its effectiveness in answering configuration queries and reasoning about dataflow, trust boundaries, and potential issues within the system. Specifically, we test DLs reasoning capabilities to infer new facts about underspecified resources (such as those not included in a given deployment but used by it) and leverage DLs *open-world assumption* to perform verification and refutation, depending on the property being checked. We formalize additional security knowledge that allows for checking system-level semantic properties; i.e., properties that consider the nature of the cloud environment and more complex reachability over an *inferred* graph representation of the infrastructure.

Throughout the study, we make four novel contributions: (*i*) the formalization and logical encoding of AWS CloudFormation (Section 2.3); (*ii*) a technique to express security properties (Section 2.4); (*iii*) the experimental evaluation of encoding and query times, accounting for the most common security issues that we found over publicly available IaC templates (Section 2.5); and (*iv*) an extension that enables semantic dataflow reasoning (Section 2.6). Our tool is implemented in Scala and available online [51]. We include preliminaries in Section 2.2; discuss related work in Section 2.7; and conclude in Section 2.8.

2.2 Preliminaries

2.2.1 Description Logics

DLs are a family of logics well suited to model *relationships between entities*. They provide the logical foundation of the well-known *Web Ontology Language* [52–54], for which extensive tool support exists (e.g., the Protégé editor and off-the-shelf reasoners such as FaCT, HermiT, and Pellet [55–58]). We introduce the description logic $\mathcal{ALC}$ [27, 28, 59], *Attributive Logic with Complement*, and two additional features that are relevant for our study. $\mathcal{ALC}$ formulae are built from symbols from the alphabets $\mathbf{C}$, of atomic concept names; $\mathbf{R}$, of role names; and $\mathbf{I}$, of individual names. These are the DL equivalents of FOL unary predicates, binary predicates, and constants, respectively. $\mathcal{ALC}$ concept expressions are built according to the grammar:

$$C, D ::= \bot \mid \top \mid \mathsf{A} \mid \neg C \mid C \sqcup D \mid C \sqcap D \mid \exists \mathsf{r}.C \mid \forall \mathsf{r}.C$$

where A is an atomic concept from the set $\mathbf{C}$; C, D are possibly complex concepts; and r is a role from the alphabet $\mathbf{R}$. Terminological knowledge is represented via general concept inclusion axioms $C \sqsubseteq D$. As an example, in the remainder of this paper we will refer to two standard axioms that

enforce the domain and range of binary relations: $\mathsf{dom}(r, C) \equiv \exists r.\top \sqsubseteq C$ and $\mathsf{ran}(r, C) \equiv \exists r^{-}.\top \sqsubseteq C$. Assertional knowledge is represented via concept assertions $\mathsf{C}(a)$ and role assertions $\mathsf{r}(a, b)$. In this paper, we will use three additional operators: *inverse roles*, *functionality constraints*, and *complex role inclusions*. The first, denoted r^{-}, encodes the converse of the binary relationship r. The second enforces binary relationships to be functional. The third, written $r \circ s \sqsubseteq t$, establishes that the chaining of the two relationships r and s implies the relationship t, and can be used to implement transitivity (when $r = s = t$). A model of a DL knowledge base is an interpretation $\mathcal{I}$, over a domain Δ, that satisfies all the axioms and assertions contained and implied by the knowledge base. For the purpose of our application, we leverage two classical inference problems: *satisfiability* and *instance retrieval*, whose full definitions are found in standard textbooks [20, 60].

2.2.2 AWS CloudFormation

AWS CloudFormation, cfn, provides users with a declarative programming language and a framework to provision and manage over 500 *resources* spread across 70 *services* [61].[1] Services are products such as storage, databases, and processors, and their interface is implemented through resources, which are the actual modules that users declare and deploy. Their declaration is done by writing one or more so-called *CloudFormation Templates* (JSON-formatted configuration files). Within a template, users configure settings and communication of the desired resource instances. As an example, let us consider one of the most widely known storage products within AWS: the Simple Storage Service S3 (also illustrated in Listings 2.1 and 2.2). The CloudFormation interface for S3 consists of two resources: S3::Bucket and S3::BucketPolicy. A Bucket is a single unit of storage whose properties include encryption, replication, and logging settings, which can be viewed as the bucket's own configuration parameters. They could also be *references* to other resources that are connected to the current one, e.g., the unique ID of another bucket where logs are stored. A BucketPolicy is a resource that links an access control policy to a bucket. All the properties that can be instantiated and the structure of resource-types such as S3::Bucket and S3::BucketPolicy are given in the *CloudFormation Resource Specification* [61]. The *resource specification* is a collection of files that prescribe resource properties and their allowed values. Provided that a *configuration file* is valid with respect to the specifications, an IaC deployment environment compiles it into instructions that are then executed to provision the requested resources in the correct dependency order and with the desired settings.

2.3 Formalization and Encoding of IaC Deployments

While setting up this case study, we found it convenient to come up with a formalization, of both IaC resource specifications and IaC configuration files, to use as an intermediate representation during the encoding process.

[1] As of August 2020, exact number is Region-dependent.

This was also needed since we could not find suitable research in the area (although some preliminary research on IaC formalization does exist: e.g. the PhD thesis in [62]). As mentioned in Section 2.2.2, users consult the *resource specifications* to find out what fields and values are allowed when declaring a resource. Intuitively, these provide a sort of type-system, or JSON schema, against which *configuration files* must validate. Configuration files contain the resource declarations of the instances that the user wishes to deploy. Let us illustrate this with some examples.

```
"ResourceType":
"S3::Bucket": {
  "Properties":{
    "BucketName" : "String",
    "LoggingConfiguration":  {
      "Type":  "LoggingConfiguration",
      "Required":  false } ...}},

"PropertyTypes":  ...,
"S3::Bucket.LoggingConfiguration":{
  "Properties":  {
    "DestinationBucketName":{
      "Type":  "String",
      "Required":  false },
    "LogFilePrefix":{
      "Type":  "String",
      "Required":  false }}}
```

Listing 2.1. S3::Bucket specification

Listing 2.1 shows a snippet of the S3::Bucket resource-type specification. In addition to the main resource type, the specification includes definitions for its subproperties, their types, and whether these are required. Although the example only shows string properties, in general, allowed properties values range over objects, arrays, and primitive types such as integers, doubles, longs, strings, and booleans.

```
"ConfigS3Bucket": {
  "Type":  "AWS::S3::Bucket",
  "Properties":
   "BucketName": "ConfigStore",
   "LoggingConfiguration": {
     "DestinationBucketName":"ConfigStore",
     "LogFilePrefix":"config-bucket-logs/"}}
```

Listing 2.2. S3::Bucket instance declaration

Listing 2.2, on the other hand, shows a common usage scenario of the S3 storage service, where a bucket with basic configuration is used to store the desired data. The instance has logical ID `ConfigS3Bucket`, is of type S3::Bucket, and specifies two top-level properties, `BucketName` and `LoggingConfiguration`. It is easy to see that this instance declaration validates against the resource specification of Listing 2.1. This snippet is taken from one of the benchmark deployments evaluated in Section 2.5 (StackSet 15) and, incidentally, it violates a security best practice: "no bucket should store its own logs." Such formalization has been instrumental to capture infrastructure configurations, resources settings and inter-connections, and to precisely and automatically encode it into DL.

Encoding We translate IaC specifications into DL terminological knowledge, and IaC configurations into assertional knowledge. The conceptual modeling features needed to model the former include axioms to define *domain* and

range of properties, *requiredness*, and *functionality*. These give us enough expressivity to infer qualities of nodes that are underspecified, such as those that are referenced by a template but not declared in it (e.g., already deployed and running elsewhere), whose configuration is unknown. To give readers an intuition of the encoding procedure, let us look at the equation below, which contains some of the axioms and assertions generated by the translation of the code in Listings 2.1 and 2.2.

$$
\begin{aligned}
\mathit{Spec}_{\mathsf{S3::Bucket}} = \{\ & \mathsf{dom}(\mathsf{bucketName}, \mathsf{BUCKET}),\ \mathsf{ran}(\mathsf{bucketName}, \mathit{String}), ..., \\
& (\mathsf{Funct\ bucketName}), \\
& \mathsf{dom}(\mathsf{destinationBucket}, \mathsf{LOGCONFIG}), \\
& \mathsf{ran}(\mathsf{destinationBucket}, \mathsf{BUCKET}),\ \ldots\ \}
\end{aligned}
$$

$$
\begin{aligned}
\mathit{Config} = \{\ & \mathsf{BUCKET}(\mathit{ConfigS3Bucket}), \\
& \mathsf{bucketName}(\mathit{ConfigS3Bucket}, \text{“}\mathit{ConfigStore}\text{”}), \\
& \mathsf{loggingConfig}(\mathit{ConfigS3Bucket}, x), \\
& \mathsf{destinationBucket}(x, \mathit{ConfigS3Bucket}), \\
& \mathsf{logFilePrefix}(x, \text{“}\mathit{config\text{-}bucket\text{-}logs}\text{”})\ \}
\end{aligned}
$$

2.4 Security Properties Specification

We group properties into three categories that reflect their high-level meaning: *security issues*, *mitigations*, and *global protections* to security concerns. We view these in analogy to *must* and *may* specifications, which one would use to express that an issue may be present (vs. must be absent) or that a protection must be in place (vs. may be missing). Each property type is matched to a corresponding query structure, which aids the translation of security requirements into formal specifications and implements different fail/pass logics. Queries are written as description logic expressions whose outcome can be one of UNSAT, SAT with no instance found (SAT/0), and SAT with instances (SAT/+). These are achieved by running a satisfiability check, possibly followed by an instance retrieval call.

Mitigations are configurations of single resources that reduce the likelihood of a security event. In order to pass, these checks must be verified. Examples are:

M1 *“All buckets must keep logs,”*

M2 *“Only buckets that host websites can have a public preset ACL,”* and

M3 *“Data stores must have backup or versioning enabled.”*

Security Issues are configurations that potentially increase exposure to security concerns. In order to pass, these checks must be falsified. Examples are:

I1 *“There may be a bucket that is not encrypted,”*

I2 *"Encrypted bucket that sends events to a not-encrypted queue,"* and

I3 *"There may be a networking component that opens all ports to all."*

Global Protections are more general mitigations, applied on single resources or as configuration patterns, whose presence and proper configuration ensures protection over the system as a whole. Examples are:

P1 *"There is an alarm configured to perform an action when triggered,"* and

P2 *"There is a configuration recorder logging changes to the infrastructure."*

We refer the reader to the repository in [51] for the properties specification files.[2]

2.5 Application to Existing Infrastructure

We now discuss the application of our approach to real-world IaC deployments. We analyze AWS CloudFormation specification and configuration files, showing that the approach is practical, scalable, and identifies potential security issues.

Operation of the Tool We develop a tool that performs three main tasks. First, the encoding of the cfn resource specifications into formal models (*Resource Terminologies*).[3] Second, the encoding of the actual cfn configuration files, also called *StackSet*, into formal models (*Infrastructure Model*). Third, inference and query answering for a set of predefined queries. We use the OWLApi [63] for the encoding phase, and JFact [56] as the inference engine.

Experimental Setup We run our tool on 15 CloudFormation StackSets openly available on GitHub. Regarding metrics, we define the infrastructure size as the numbers of both declared resources (N) and their types (N_{RT}). The latter determines which *resource terminologies* are imported into the final encoded model and thus influences its size, measured in number of logical axioms (N_α). The smallest StackSet has 6 resources and 6 resource types, the largest has 508 resources and 21 resource types. We implement 50 properties from the ScoutSuite collection [64] that are applicable at design time and, thus, over IaC deployment files. Of the 50 properties, 29 are *mitigations*, 18 are *security issues*, and 3 are *global protections*. We conduct our evaluation on an Intel Core i5 with 16GB RAM and perform warmup runs and clear the heap before each measurement. This tuning helps to minimize the impact of just-in-time compilation and to reduce the likelihood of garbage collection during the measured benchmark runs.

Results Evaluation The average compilation time of the entire cfn resource specifications (542 files) was 940ms. Table 2.1 reports the results of our experimental evaluation. StackSets are sorted by number of resources. For

[2] https://tiny.cc/PropertiesSpecifications

[3] Available here: https://tiny.cc/ResourceTerminologies

Table 2.1. Evaluation results (mean times in millisec).

ID	N	N_{RT}	ENC	N_α	INF	UNSAT	SAT/0	SAT/+
05	6	6	44.53	814	30.64	0.67	–	2.46
11	8	8	79.22	917	37.09	0.72	–	2.86
03	10	7	59.94	886	35.65	0.64	2.23	1.56
09	10	9	76.33	940	38.66	0.68	5.03	2.96
02	11	8	76.73	1194	49.99	0.85	2.66	2.02
01	16	7	94.95	1007	43.38	0.66	3.96	1.83
08	19	8	87.66	1051	50.93	0.78	5.40	3.23
10	30	9	89.07	1177	71.23	0.86	2.62	2.08
06	30	12	102.00	1666	108.30	1.05	–	4.91
12	31	21	185.06	2798	301.61	4.99	24.93	36.43
13	51	32	241.17	3835	608.09	7.16	38.56	47.93
14	73	31	264.56	4143	847.36	2.83	51.36	19.20
15	79	21	313.40	4596	901.18	2.86	–	17.55
04	132	33	363.58	4834	2100.85	2.94	162.95	23.21
07	508	21	1005.46	10161	15834.14	7.34	40.86	13.52

each, we measure the time taken by the stackset encoding (ENC), inference (INF), and query answering task (grouped by outcome: UNSAT, SAT with no instances, and SAT with instances). As we can see from the table, the encoding time increases with the infrastructure's size, producing larger models that require longer inference times. Average query answering times increase accordingly. UNSAT queries have shorter average answering times than those evaluating to SAT/0 or SAT/+ (UNSAT proofs are found before a SAT outcome can be deduced). In addition, once a query is proved SAT, we invoke a procedure for *instances retrieval* to determine whether satisfying instances are present or not. The specific infrastructure configuration and its size are the main influencing factors of query answering times. Considering that the average template has about 50-100 resources, and templates having 100-500 resources are rare, the results suggest that our approach scales to real-world IaC templates. For example, StackSet **04** has 132 resources, is encoded in 363ms, classified in 2.1s, and has a max average per-query time of 162ms. Assuming a pool of 100 checks to be run, the automated modeling and verification of such an infrastructure would take, in the worst-case, around 18s.

2.5.1 Found Security Issues

Across all 15 deployments, we run 15x50 = 750 checks: 608 pass and 142 fail. Of the 142 failing checks, 73 do not return any instance and 69 return one or more instances (i.e., they fail with a SAT/+ outcome). Such a difference is due to the nature of the single check and its definition of failure. A *global protection* check fails when no instance implementing the protection is found; a *security issue* check fails whenever is possible (SAT/0 or SAT/+); and a *mitigation* check fails when no instance is found. We consider SAT/+ findings particularly important, as they do not only witness a potential security issue but also an actual misconfiguration. In particular, the 69 SAT/+-failing checks

fail on 239 resource instances, with the most found issues being:

Missing or misconfigured encryption	131
Missing or misconfigured logging	46
Missing or misconfigured versioning/backup/replication	44
Missing User password reset requirement	12
Misconfigured authorization	3
Misconfigured networking configuration	3

The 73 findings returning no instances fall into two groups: the absence of any monitoring or alarming system is very frequent, as is the dependency on external resources whose security posture cannot be assessed.

Absent global monitoring/alarming/logging protection	41
Usage of external resources with unknown configuration	32

2.6 Semantic Reasoning about Dataflows

To conclude our study, we manually craft two proof-of-concept models of terms related to cloud security (ontologies). We use these to extend the formalization of the CloudFormation IaC specification that was automatically generated by our tool. Such domain-specific ontologies formalize several common cloud terms, such as account, deployment, authenticated and unauthenticated users; generic dataflow terms, such as storage, process, nodes, and flows of different kind; and service-specific dataflow terms. By adding these on top of the underlying IaC formal specification, we can reason about the higher-level business logic and reachability of the infrastructure, and we can abstract it and visualize it in a more convenient way. This is where the full inference power of description logics comes into play. Such an inference power would be hard to achieve with an alternative encoding (e.g. using a modal logic). Let us illustrate how this technique is applied to system-level analyses of interest for a security review: *dataflow* and *trust boundary* analyses. A trust boundary is a portion of a system whose components trust each other and where data can securely flow. Multiple trust boundaries may exist within one system. Dataflows that travel across boundaries may introduce security issues and should be carefully reviewed.

```
"CustomerData": {
  "Type": "AWS::S3::Bucket",
  "Properties": {
    "LoggingConfig": {
      "DestinationBucket": "
          AccessLog" }}},
"TopicSubscription":{
  "Type": "AWS::SNS::Subscription",
  "Properties": {
    "Endpoint": "devs@mail",
    "Protocol": "email",
    "TopicArn": "AccessTopic" }}
```

```
"TestData": {
  "Type": "AWS::S3::Bucket",
  "Properties": {
    "LoggingConfig": {
      "DestinationBucket": "
          AccessLog" }}},
"AccessLog": {
  "Type": "AWS::S3::Bucket",
  "Properties": {
    "NotificationConfig" : {
      "TopicConfig" : {
        "Topic":"AccessTopic" }}}},
"AccessTopic": {
  "Type": "AWS::SNS::Topic" ... }
```

Figure 2.1. Sample template: accounts *prod* (left) and *test* (right).

In Fig. 2.1, we see an example of such a situation, where the infrastructure is deployed across two accounts, *prod* and *test*, sharing resources `AccessLog` and `AccessTopic`. In our encoding, we use the so-called DLs *inclusion* axioms to rewrite properties that (when chained) imply the existence of a more general relation and to infer additional characteristics of nodes. For example, in the following list axioms 2-7 formalize the relationships of *"logging to"* and *"sending notifications to"* a resource, which imply the existence of a *transitive dataflow* between nodes; and axioms 8-9 allow to infer that the node *devs@mail* is an external node.

$$\mathsf{LoggingConfig} \circ \mathsf{DestinationBucket} \sqsubseteq \mathsf{logsTo} \tag{2.1}$$

$$\mathsf{TopicArn}^{-} \circ \mathsf{Endpoint} \sqsubseteq \mathsf{sendsNotifications} \tag{2.2}$$

$$\mathsf{NotificationConfig} \circ \mathsf{TopicConfig} \circ \mathsf{Topic} \sqsubseteq \mathsf{sendsNotifications} \tag{2.3}$$

$$\mathsf{logsTo} \sqsubseteq \mathsf{dataflow} \tag{2.4}$$

$$\mathsf{sendsNotifications} \sqsubseteq \mathsf{dataflow} \tag{2.5}$$

$$\mathsf{dataflow} \circ \mathsf{dataflow} \sqsubseteq \mathsf{dataflow} \tag{2.6}$$

$$\exists \mathsf{Protocol}.\{\text{“}\texttt{email}\text{”}\} \sqsubseteq \forall \mathsf{Endpoint}.\mathsf{EmailAddress} \tag{2.7}$$

$$\mathsf{EmailAddress} \sqsubseteq \mathsf{ExternalNode} \tag{2.8}$$

This encoding enables us to compute a succinct dataflow diagram from the reasoned IaC configuration (see Fig. 2.2), and to formally verify properties that usually require a manual analysis of the infrastructure and its underlying graph representation.

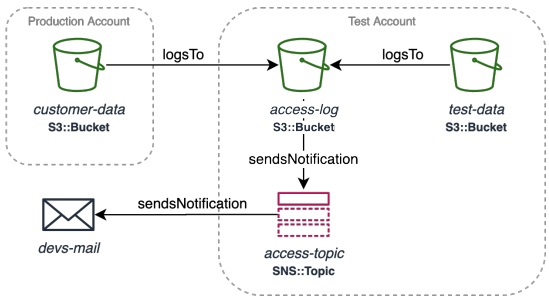

Figure 2.2. Dataflow extracted from Fig. 2.1

E.g., the question, *"can data flow from the customer-data bucket to the outside?"* can now be formalized as a DL formula and, using a reasoning engine, the existence of a dataflow that starts on the *customer-data* bucket and reaches the *devs@mail* node can now be inferred. We note that, due to the structure of the `TopicSubscription` resource, this dataflow could not have been detected with simple reachability analysis on a graph built without the aid of semantic reasoning. Moreover, the dataflow diagram highlights another potential source of information leakage: testers being exposed to customer access information. This needs to be mitigated by enforcing the proper trust boundaries, in particular, by adding a dedicated access log storage for *customer-data* bucket in the *prod* account.

2.7 Related Work

To the best of our knowledge, the problem of formally verifying the design of a cloud infrastructure in its entirety has not been addressed before. Formal reasoning techniques have been successfully applied to different aspects of the cloud, e.g. networks and access policies [6,8,65,66]. Non-formal tools exist that recommend and run checks against *already deployed* resources [64,67], or scan IaC templates [68–70] for syntactical patterns violating security best practices. These checks overlap considerably and can be expressed in our framework as well. The disadvantages of such tools are that checks are local to single components, can be performed only post-deployment, need complex configurations, access permissions, or even manual interaction. The CFn-Linter [69] has a rule-based component that users can extend with custom syntax checks, but none of the rules currently available focus on security. The CFn-nag linting tool [68] checks compliance to best practices only locally to the single resources; e.g., it cannot detect issues such as *"there is an events queue, receiving from a bucket with critical functionality, that may not be encrypted"* or *"there might be a user that is shared by multiple policies"* (which would go against the *least privilege* principle); as well as including in its analysis external resources that are referenced by the template being linted.

Regarding our choice of logic, large-scale configuration problems have been tackled with description logic before [71,72]. Simpler first-order logic formulas with operators to represent object-oriented interface relationships could be used to model IaC specifications. However, such an encoding would only partially solve our problem, which is more complex because our overall goal is to do formal semantic analyses (e.g. dataflow and threat modeling). Semantic-based approaches, even DL-based, are being used to do conceptual modeling of security engineers' expertise with the provable and explainable inference capabilities of logics. As an example, we refer the reader to the OWASP *"Ontology-driven Threat Modeling"* project [73] that aims at the formalization of security-related knowledge in the context of different types of computer systems by means of description logic ontologies. In contrast to logic programming languages, such as Datalog, DLs inherently support functionality axioms and the existence of anonymous individuals within a domain that is assumed to be open. These are supported out-of-the-box without the need for an additional, more complex, axiomatization or encoding. In particular, we took advantage of DL's open-world assumption to implement, in our properties encoding, verification and falsification. Another alternative to DLs as a modeling language would be to use 3-valued models with labels on states and transitions and apply model checking [74,75]. However, expressive branching-time logics [76,77] have not been studied in the context of 3-valued models and we are also not aware of tool support at the level available for DLs (cf. [78,79]).

2.8 Conclusion and Future Work

Throughout this case study, we investigated the usage of description logics-based semantic reasoning to evaluate the security of cloud infrastructure pre-deployment. We encoded Amazon Web Services' Infrastructure as Code specifi-

cations and configurations into description logic models and verified the presence and absence of potential security issues. We showed how this approach enables deeper system-level analyses such as dataflow analysis. All results can be generalized to other existing IaC tools. While working on this project, we interacted with developers on two occasions. First, for the benchmark templates used in our experimental evaluation, we contacted the owners, told them about the misconfigurations, and discussed potential security implications. Second, within AWS, security engineers use a technique based on this paper for security reviews of AWS products before they are launched, helping developers fix real issues pre-deployment. In the process, we received valuable feedback that we used for improving precision and reducing the number of false-positive results. We plan to continue researching for an even better-fitting description logic formalism, query language, three-valued semantics, and decision procedures for verification and falsification of properties relevant to security analyses, such as dataflows, trust boundaries, and threat modeling.

Acknowledgements

This research is supported by the ERC consolidator grant D-SynMA under the European Union's Horizon 2020 research and innovation programme (grant agreement No 772459) and by Amazon Web Services.

A Appendix

A.1 AWS CloudFormation Formalization

As a preliminary step in the process of representing CloudFormation templates as formal models, we identify the key components that are responsible for both the resources' settings and their interactions, and map them into a precise mathematical notation. This notation will serve as an intermediate representation, aiding the translation of cfn templates into description logic knowledge bases. The formalization, and the successive translation, are straightforward; however, we report it here in appendix to give a more rigorous definition of the contribution presented in the paper.

As introduced in the paper, the two main components of IaC tools, and therefore of CloudFormation, are *resource specifications* and configuration (or *template*) files.

Resource Specifications For each "resource type" declarable in a template, the specification defines its properties and their allowed values. We call such a structure the *Resource-Type*. Similarly to a JSON Schema, a specification file is a JSON document itself that supports validation,[4] meaning that every *instance* of the given type must validate against it. More precisely, each resource specification provides the schema of the main resource-type and of possibly many (sub)property-types. This is done by using *keywords* to shape and restrict the set of valid documents.

Template Files Configuration files contain the resource declarations of the instances that the user wishes to deploy. This is done by writing JSON-formatted text files containing a list of resource instances to be deployed together with their settings (conform to the definitions given in the resource's type specification). Under each resource instance's name and type, users specify properties as a list of attribute-value pairs. In line with the general definition of JSON documents, in cfn templates attributes have values that range over objects, arrays, or primitive types. When referring to a property's value type, we say that the values allowed for that property range over subproperties, arrays, strings, integers, longs, doubles, booleans, and the constant null. Overall, the values that are allowed in cfn templates coincide with the standard JSON values: integers, longs, and doubles are JSON numbers; subproperties are JSON objects.

Formalizing Specifications Syntax. Both resource- and property-types have a name and a collection of property definitions. Each property definition is given by a name, a type, and a flag indicating whether the property is required or not. Let RT_i denote the name of the principal resource-type within the i-th resource specification, and the labels $PT_{i.1} \dots PT_{i.\ell^i}$ denote the names of all its (sub)property-types (collectively referred to as the set PT_i). We assume the existence of a renaming function that unambiguously maps specification names to unique identifiers and, when talking about properties and types, we mean their unique renaming. We call the properties appearing under the specification of a resource- or property-type its *defining properties*, or simply *its* properties.

[4] Strictly speaking, according to the technical standard of *"JSON Schema Language"* in IETF Internet-Draft, this specification is not the JSON Schema of a template.

The type of a property from the i-th specification ($\texttt{T}_\texttt{i}$) is one of:

$$\begin{aligned} \texttt{T}_i &::= T \mid \mathit{List}(T) \mid \mathit{PT}_i \mid \mathit{List}(\mathit{PT}_i) \\ T &::= \mathit{String} \mid \mathit{Integer} \mid \mathit{Long} \mid \mathit{Double} \mid \mathit{Boolean} \end{aligned} \tag{2.9}$$

Properties within a given RT_i always range over types of the i-th specification, never referencing a set $PT_{i'}$ s.t. $i' \neq i$ or RT_i itself. In other words, resources do not *explicitly* point to each other. However, as explained in the *Inter-Resource References* paragraph below, there can be *implicit* references.

Formalizing Templates Syntax. Every resource declared in a cfn template can be modeled as a tree-shaped structure whose root represents the resource itself and the branching follows from its properties. Accordingly, the whole template is a forest of resource trees. Formally, each tree t is the tuple (N, E, A, v), where N is the set of nodes; E and A are sets of edges; and v is a value function. The set N is the tree domain, which contains the root and a node for each object, array, and value appearing within a resource declaration's `Properties` block. It is, therefore, partitioned into the disjoint sets $\{r\}$, *Obj*, *Arr*, and *Val*. The root node r carries extra information: the pair $\langle id, type \rangle$ of the resource's logical ID and type. The edges of the tree derive from two nesting relations, found in properties and arrays, respectively. The first is the *property-value* relation E, connecting a root or object node to the nodes encoding their properties values. E-edges are labeled by property names from the set $\mathbf{R}$ and are contained in the set $(\{r\} \cup \mathit{Obj}) \times \mathbf{R} \times (N \setminus \{r\})$. The second relation is called *array* relation A, and links every array node to its elements by introducing indexed edges. It is contained in $\mathit{Arr} \times \mathbb{N} \times (\mathit{Obj} \cup \mathit{Val})$. Nodes in N do not carry information about the actual content of a property value. This is particularly relevant for nodes in the set *Val* that are leaf nodes and need to be assigned a concrete element from their respective domains. An exception is made only for *Val* nodes corresponding to the JSON null, which do not have an assignment. Let *String*, *Int*, *Long*, *Double*, and *Bool* be the sets of all concrete elements that can be associated with the content of a value node. The partial function $v : \mathit{Val} \to (\mathit{String} \cup \mathit{Int} \cup \mathit{Long} \cup \mathit{Double} \cup \mathit{Bool})$ maps the leaf nodes to concrete elements and it is undefined for null nodes.

Inter-Resource References. In Listing 2.1, the type of property `DestinationBucketName` is declared as *String*. However, further documentation specifies that the string must contain a reference to a resource of type S3::Bucket. Usually, such reference takes the form of an *Amazon Resource Name* (aka ARN, the instance's unique address) or, if the resource is in the same template, of its name. The snippet in Listing 2.2 is a case in point – `"ConfigStorage"` is the *name* of the referenced resource of type S3::Bucket. To cover for such cases, we extend our formalization to include the definition and the resolution of these references, which are needed when encoding the cfn resource specifications and the cfn templates, respectively. Let $\mathbf{C}$ be the alphabet of distinct type names, $\mathbf{R}$ the alphabet of property names, and $\mathbf{I}$ a set of individual names. To *define* the references as part of the specification, we introduce a function $\mathsf{range} : \mathbf{R} \to \mathbf{C}$ whose domain is the set of *all* properties and range matches the grammar of Equation (2.9) extended with resource-types. If a property $p \in \mathbf{R}$ is of type string and intended to reference a resource, $\mathsf{range}(p)$ is set to the correct resource-type RT_j (for some j given in the documentation); otherwise,

$\mathsf{range}(p)$ is set to the original type prescribed in the specification file. To *resolve* the references in actual templates, we introduce a $\mathsf{resolve}$ procedure that given a node x returns the corresponding DL individual i_x; i.e., $\mathsf{resolve}: N \to \mathbf{I}$. The function is undefined over nodes not referencing any resource; returns a pre-existing individual over those nodes that point to a resource declared in the same template; returns a fresh individual in $\mathbf{I}$ for those nodes referencing a resource that is external to the template. From now on, we assume range to capture the *real* type of a given property and $\mathsf{resolve}$ to return the correct individual. We rely on these for the encoding of the next section.

A.2 AWS CloudFormation Description Logic Encoding

We now describe the translation implemented in our tool from the mathematical notation formalized in the previous appendix to DL. The tool creates a *"Terminological Box"* (also called *TBox* in description logic) from the resource specifications and an *"Assertional Box"* (also called *ABox* in description logic) from the cfn templates.

Resource Specifications Translated to Terminological Boxes. We construct a global cfn DL specification $\mathcal{T}^{\mathsf{cfn}}$ as the union of TBoxes $\mathcal{T}_{RT_i}$ that we create for each individual resource specification RT_i. We introduce concept predicates matching resource-, property-, and primitive types, and role predicates matching property names. The structure of the cfn resource specifications is preserved by introducing axioms that simulate the characteristics of the properties declared therein: domain, range, requiredness, and functionality. Let X be a property- or resource-type in RT, and p one of its defining properties. The collection of TBox axioms enforce (1) the connectivity of a resource to its properties, (2) the type application of the properties, (3) required properties, and (4) connection to individuals rather than lists. The first and the second are enforced using dom and ran axioms (see Sec. 2.2). For the third we use axioms of the form $\mathsf{req}(p, X) \equiv X \sqsubseteq \exists p.\top$, which enforce X-nodes to have at least one p-successor. For the fourth we use axioms of the form $\mathsf{fun}(p) \equiv {\geq_2} p.\top \sqsubseteq \bot$, which enforce a single p successor when p is not a list. Overall, the tool builds $\mathcal{T}_{RT}$ as:

$$\mathcal{T}_{RT} = \bigcup_{(X,p)\in RT} \{\ \mathsf{dom}(p, X), \mathsf{ran}(p, \mathsf{range}(p)), \mathsf{req}(p, X), \mathsf{fun}(p)\ \}$$

We note that we have not created any individuals or assertions. We created axioms that constrain the valid interpretations and enable inference on the characteristics of underspecified nodes, such as those that are *pointed by* a cfn template but not declared in it.

Templates Translated to Assertional Boxes. For a template t, valid against the cfn specifications, our tool creates ABox assertions $\mathcal{A}^{\mathsf{t}}$. The construction ensures that $\mathcal{A}^{\mathsf{t}}$ is consistent w.r.t. the DL encoding of the specifications, $\mathcal{T}^{\mathsf{cfn}}$. For resource declarations we add individuals, and model their configuration by means of assertions. Because DLs are interpreted under an *open-world assumption*, which treats missing information as unknown, and since we assume a template to be complete, we *"close the world"* around the given properties' assertions by using additional concept assertions. Let us

recall, from the formalization, that a template is a forest of resource trees, each having its own root, nodes, edges, and value function; and that trees may be implicitly connected via inter-resource references. In the following, we give an overview of how: *i*) our tool creates individuals for nodes, and *ii*) creates concept and role assertions from node types and properties.

Create Individuals for Nodes We construct the alphabet of individual names $\mathbb{I}$ and a map ind from nodes into individuals, which is used for the construction of role assertions. For x a root node with identifier id, our tool adds the symbol id to $\mathbb{I}$ and sets $\mathsf{ind}(x)=\{id\}$. For x a node in *Obj*, our tool adds individual i_x to $\mathbb{I}$ and sets $\mathsf{ind}(x)=\{i_x\}$. For x a node in *Val* s.t. $\mathsf{resolve}(x)$ is defined, our tool does not add any new individual names (the $\mathsf{resolve}$ invocation takes care of that, see Sec. 2.3) and sets $\mathsf{ind}(x)=\{\mathsf{resolve}(x)\}$. For x a node in *Val* s.t. $\mathsf{resolve}(x)$ is undefined and $v(x)=y$, our tool adds the symbol y to $\mathbb{I}$ and returns $\mathsf{ind}(x)=\{y\}$. For x a node in *Arr* s.t. $A(x)=\{(1,v_1),\ldots,(k,v_k)\}$, our tool does not add any fresh names to $\mathbb{I}$ but sets $\mathsf{ind}(x)=\bigcup_{i\in[k]}\mathsf{ind}(v_i)$. The result is that we do not preserve the encoding of the array parent node x but rather relate the parent of x to each encoded element of the array. We note that ind is set to a singleton set in all cases except the latter. When clear from context that such set is a singleton, we denote by i_x the individual encoding node x, omitting the statement $i_x\in\mathsf{ind}(x)$.

Create Assertions from Configurations The newly introduced individuals are related through assertions, which encode the structure of the resource trees and their settings. The encoding of root and object nodes is straightforward. However, since DLs are interpreted under an *open-world assumption* that treats missing information as unknown, we seek a special treatment for the explicit encoding of properties that are *not* given in the template and for the encoding of bounded arrays. The assertions created from nodes and properties are as follows:

$$\begin{aligned}
\mathsf{root}(j) &= \{\ \mathsf{RT}(id), \mathsf{p}(id,i) \mid r_j=\langle id,RT\rangle,\ (r_j,p,c)\in E^j,\ i\in\mathsf{ind}(c)\ \}\\
\mathsf{obj}(j,x) &= \{\ \mathsf{p}(i_x,i) \mid (x,p,c)\in E^j,\ i\in\mathsf{ind}(c)\ \}\\
\mathsf{abs}(j,x) &= \{\ \neg\exists\mathsf{p}.\top(i_x) \mid (x,p,c)\notin E^j,\ p \text{ is a property of } x\text{'s type}\ \}\\
\mathsf{arr}(j,x) &= \{\ {\leq}k\mathsf{p}.\top(i_x) \mid (x,p,a)\in E^j,\ a\in Arr^j,\ |A^j(a)|=k\ \}
\end{aligned}$$

For every root node r_j, with identifier id and type RT, our tool adds the concept assertion $\mathsf{RT}(id)$ (see root). We add the role assertion $\mathsf{p}(i_x,i)$ to encode the properties of either a root or object node x for every triplet (x,p,c) in the set E and every i arising from the mapping of node c (see root, obj). For a node x and property p, which could be set (according to the specification) under a node of x's type but is absent in the template, our tool adds the axioms $\mathsf{abs}(\cdot)$. These axioms enforce individuals i_x to have no p-successors. For a node x linking to an array node a, our tool adds the axioms $\mathsf{arr}(\cdot)$, which enforce the exact array cardinality. In addition, we remark that we do not need any further assertions to encode value and array nodes, as their mapping into individuals is enough. Lastly, no axioms are added for the individuals introduced by the $\mathsf{resolve}$ function (and external to the template), since their information is incomplete. The tool thus creates the following ABox $\mathcal{A}^{\mathsf{t}}$:

$$\mathcal{A}^{\mathsf{t}}=\bigcup_{t_j\in\mathsf{t}}\Big(\ \mathsf{root}(j)\ \cup\ \bigcup_{x\in Obj^j}\mathsf{obj}(j,x)\ \cup\ \bigcup_{x\in\{r_j\}\cup Obj^j}\big(\ \mathsf{abs}(j,x)\ \cup\ \mathsf{arr}(j,x)\ \big)\Big)$$

A.3 Formal Encoding of the Infrastructure in Listing 2.2

By following the encoding procedure presented, the S3::Bucket resource specification (partly reported in Listing 2.1) and the deployment template snippet of Listing 2.2 are encoded into the knowledge base $\mathcal{K} = (\mathcal{T}_{\mathsf{S3::Bucket}}, \mathcal{A})$ that is defined as follows:

$$
\begin{aligned}
\mathcal{T}_{\mathsf{S3::Bucket}} = \{\ & \exists \mathsf{bucketName}.\top \sqsubseteq \mathsf{BUCKET},\\
& \exists \mathsf{bucketName}^{-}.\top \sqsubseteq \mathsf{STRING},\\
& {\geq_2}\mathsf{bucketName}.\top \sqsubseteq \bot,\\
& \exists \mathsf{loggingConfig}.\top \sqsubseteq \mathsf{BUCKET},\\
& \exists \mathsf{loggingConfig}^{-}.\top \sqsubseteq \mathsf{LOGGINGCONFIG},\\
& {\geq_2}\mathsf{loggingConfig}.\top \sqsubseteq \bot,\\
& \exists \mathsf{destinationBucket}.\top \sqsubseteq \mathsf{LOGGINGCONFIG},\\
& \exists \mathsf{destinationBucket}^{-}.\top \sqsubseteq \mathsf{BUCKET},\\
& {\geq_2}\mathsf{destinationBucket}.\top \sqsubseteq \bot,\\
& \exists \mathsf{logFilePrefix}.\top \sqsubseteq \mathsf{LOGGINGCONFIG},\\
& \exists \mathsf{logFilePrefix}^{-}.\top \sqsubseteq \mathsf{STRING},\\
& {\geq_2}\mathsf{logFilePrefix}.\top \sqsubseteq \bot,\\
& \exists \mathsf{notifConfig}.\top \sqsubseteq \mathsf{BUCKET},\\
& \exists \mathsf{notifConfig}^{-}.\top \sqsubseteq \mathsf{NOTIFCONFIG},\\
& {\geq_2}\mathsf{notifConfig}.\top \sqsubseteq \bot,\\
& \exists \mathsf{topicConfigs}.\top \sqsubseteq \mathsf{NOTIFCONFIG},\\
& \exists \mathsf{topicConfigs}^{-}.\top \sqsubseteq \mathsf{TOPICCONFIG},\\
& \exists \mathsf{topic}.\top \equiv \mathsf{TOPICCONFIG},\\
& \exists .\mathsf{topic}^{-}.\top \sqsubseteq \mathsf{TOPIC}\ \}\\
\mathcal{A} = \{\ & \mathsf{BUCKET}(\mathit{ConfigS3Bucket}),\\
& \mathsf{bucketName}(\mathit{ConfigS3Bucket}, \text{“}\mathit{ConfigStore}\text{”}),\\
& \mathsf{loggingConfig}(\mathit{ConfigS3Bucket}, x),\\
& \mathsf{destinationBucket}(x, \mathit{ConfigS3Bucket}),\\
& \mathsf{logFilePrefix}(x, \text{“}\mathit{config\text{-}bucket\text{-}logs}\text{”})\ \}
\end{aligned}
$$

The assertions introduced in the ABox $\mathcal{A}$ only specify the nature of the root node *ConfigS3Bucket*. No concept assertion is introduced for the nested nodes x, *ConfigStore*, and *config-bucket-logs*. However, by combining the axioms in $\mathcal{T}_{\mathsf{S3::Bucket}}$ with the knowledge asserted in $\mathcal{A}$, the reasoning engine will infer that LOGGINGCONFIG(x), STRING(*ConfigStore*), and STRING(*config-bucket-logs*).

A.4 Formal Encoding of the Infrastructure in Fig. 2.1

The encoding procedure applied to the template of Fig. 2.1 and to the resource specifications of the S3 and SNS resource types (for the sake of brevity not reported here) results in the knowledge base $\mathcal{K} = (\ \mathcal{T}_{\mathsf{S3::Bucket}} \cup \mathcal{T}_{\mathsf{SNS::Topic}} \cup \mathcal{T}_{\mathsf{SNS::Subscription}}, \mathcal{A}\ )$, where $\mathcal{T}_{\mathsf{S3::Bucket}}$ is the same as in the previous appendix,

and the remaining components are as follows:

$$
\begin{aligned}
\mathcal{T}_{\mathsf{SNS::Topic}} = \{\ & \exists \mathsf{subscription}.\top \sqsubseteq \mathsf{TOPIC},\\
& \exists \mathsf{subscription}^{-}.\top \sqsubseteq \mathsf{TOPICSUBSCR},\\
& \exists \mathsf{endpoint}.\top \equiv \mathsf{TOPICSUBSCR},\\
& \exists \mathsf{endpoint}^{-}.\top,\\
& {\geq_2}\mathsf{endpoint}.\top \sqsubseteq \bot,\\
& \exists \mathsf{protocol}.\top \equiv \mathsf{TOPICSUBSCR},\\
& \exists \mathsf{protocol}^{-}.\mathsf{STRING},\\
& {\geq_2}\mathsf{protocol}.\top \sqsubseteq \bot, \dots \}\\
\mathcal{T}_{\mathsf{SNS::Subscription}} = \{\ & \exists \mathsf{endpoint1}.\top \sqsubseteq \mathsf{SUBSCRIPTION},\\
& \exists \mathsf{endpoint1}^{-}.\top,\\
& {\geq_2}\mathsf{endpoint1}.\top \sqsubseteq \bot,\\
& \exists \mathsf{protocol1}.\top \equiv \mathsf{SUBSCRIPTION},\\
& \exists \mathsf{protocol1}^{-}.\mathsf{STRING},\\
& {\geq_2}\mathsf{protocol1}.\top \sqsubseteq \bot,\\
& \exists \mathsf{topicArn}.\top \equiv \mathsf{SUBSCRIPTION},\\
& \exists \mathsf{topicArn}^{-}.\mathsf{TOPIC},\\
& {\geq_2}\mathsf{topicArn}.\top \sqsubseteq \bot, \dots \}\\
\mathcal{A} = \{\ & \mathsf{BUCKET}(\mathit{CustomerData}),\\
& \mathsf{loggingConfig}(\mathit{CustomerData}, x),\\
& \mathsf{destinationBucket}(x, \mathit{AccessLog}),\\
& \mathsf{BUCKET}(\mathit{AccessLog}),\\
& \mathsf{notifConfig}(\mathit{AccessLog}, x'),\\
& \mathsf{topicConfig}(x', x''),\\
& \mathsf{topic}(x'', \mathit{AccessTopic}),\\
& \mathsf{SUBSCRIPTION}(\mathit{TopicSubscription}),\\
& \mathsf{endpoint1}(\mathit{TopicSubscription}, \mathit{devs@mail}),\\
& \mathsf{protocol1}(\mathit{TopicSubscription}, \text{“}\mathit{email}\text{”}),\\
& \mathsf{topicArn}(\mathit{TopicSubscription}, \mathit{AccessTopic}),\\
& \mathsf{TOPIC}(\mathit{AccessTopic}),\\
& \mathsf{BUCKET}(\mathit{TestData}),\\
& \mathsf{loggingConfig}(\mathit{TestData}, y),\\
& \mathsf{destinationBucket}(y, \mathit{AccessLog})\}
\end{aligned}
$$

A.5 Sample of Security Properties

A.5.1 Mitigations

Also called *All-known-must* queries, meaning that a property φ *certainly* holds on all the *known* objects of interest, formulated as $(\sqcup_{x \in inst(\mathsf{X})} \{x\}) \sqcap \neg\varphi$ and interpreted as *tt-ff-ff* over the three outcomes UNSAT, SAT/0, and SAT/+.

These properties are used to express that *all known instances of a given concept* X *certainly satisfy a property* φ_{X}. They are suitable for expressing *mitigations* to security threats that must be in place in order for the property to be satisfied and for the check to pass. We adopt an epistemic approach, where by *known instances* we mean all the individuals that satisfy the concept X in all models. We simulate epistemic queries. To the best of our knowledge, epistemic queries (corresponding to the language *EQL-Lite(UCQ)* from [80]) are implemented in the *SparSQL* query language [81], used in the MASTRO reasoner over DL-Lite$_A$ ontologies; but are not supported by more expressive description logic reasoners. We simulate the epistemic operator by building a composite query that, first, fetches the set of all individuals certainly satisfying X (which we denote as $inst(\mathsf{X})$) and, then, builds the corresponding nominal concept set to be included in the DL query. The DL queries have the following structure: $\sqcup_{x \in inst(\mathsf{X})} \{x\} \sqcap \neg\varphi_{\mathsf{X}}$ and their interpretation over the three outcomes (UNSAT, SAT/0, and SAT/+) is of True, False, and False, respectively.

Example 1. *"Only* S3::Bucket*s that host a website can be public"*

$$\sqcup_{x \in inst(\mathsf{Bucket})} \{x\} \sqcap \exists \mathsf{accessControl}.\{Public\} \sqcap \neg \exists \mathsf{websiteConfig}.\top$$

Example 2. *"All* S3::Bucket*s with a critical functionality must be encrypted and log to an encrypted bucket."*

$$\sqcup_{x \in inst(\mathsf{CriticalBucket})} \{x\} \sqcap \neg \mathsf{EncrLoggBucket}$$

where

$$\begin{aligned}
\mathsf{CriticalBucket} \equiv\ & \exists \mathsf{s3bucketName}^{-}.\mathsf{DeliveryChannel} \sqcup \\
& \exists \mathsf{s3bucket}^{-}.(\exists \mathsf{code}^{-}.\mathsf{Function}) \sqcup \\
& \exists \mathsf{s3bucket}^{-}.\mathsf{AccessLoggingPolicy} \sqcup \\
& \exists \mathsf{bucket}^{-}.(\exists \mathsf{bodyS3location}^{-}.\mathsf{RestApi}) \\
\mathsf{EncrBucket} \equiv\ & \exists \mathsf{encryption}.(\ \exists \mathsf{sseConfiguration}.(\ \exists \mathsf{kmsMasterKeyId}.\mathsf{Key}\ )) \\
\mathsf{EncrLoggBucket} \equiv\ & \mathsf{EncrBucket} \sqcap \\
& \exists\, \mathsf{loggingConfig}.(\ \exists \mathsf{destinationBucket}.\mathsf{EncrBucket}\ ))
\end{aligned}$$

Example 3. *"All known* KMS::Key*s must be enabled"*

$$\sqcup_{x \in inst(\mathsf{Key})} \{x\} \sqcap \neg \exists \mathsf{enabled}.\{tt\}$$

Example 4. *"No* S3::Bucket *should store its own logs"*

$$\sqcup_{x \in inst(\mathsf{LogBucket})} (\ \{x\} \sqcap \exists \mathsf{loggingConfig}.\exists \mathsf{destinationBucket}.\{x\}\ )$$

where

$$\mathsf{LogBucket} \equiv \exists \mathsf{destinationBucket}^{-}.\exists \mathsf{loggingConfig}^{-}.\mathsf{Bucket}$$

A.5.2 Security Issues

Also called *Exists-known-may* queries, meaning that a property φ *may* hold on some *known* instances, formulated as $(\sqcup_{x \in inst(\mathsf{X})} \{x\}) \sqcap \varphi$ with an interpretation of *ff-tt-tt* over the three outcomes UNSAT, SAT/0, and SAT/+.

These properties are used to express that *there exist a known instance of a concept* X *that may satisfy a property* φ_{X}. They are suitable for expressing *security issues* that might be present and could cause the check to fail. Similarly to the previous type of property, we adopt an epistemic approach and simulate the epistemic operator by building a composite query that, first, fetches the set of all individuals certainly satisfying X (which we denote as $inst(\mathsf{X})$) and, then, builds the corresponding nominal concept set to be included in the DL query. The DL queries have the following structure: $\sqcup_{x \in inst(\mathsf{X})} \{x\} \sqcap \varphi_{\mathsf{X}}$ and their interpretation over the three outcomes (UNSAT, SAT/0, and SAT/+) is of False, True, and True, respectively.

Example 5. *"There is a* SQS::Queue *known to receive from a bucket with critical functionality that might not be encrypted"*

$$\sqcup_{x \in inst(\mathsf{CriticalQueue})} \{x\} \sqcap \neg \exists \mathsf{kmsMasterKeyId.Key}$$

where

$$\mathsf{CriticalQueue} \equiv \mathsf{Queue} \sqcap \exists \mathsf{queue}^{-}.\exists \mathsf{queueConfig}^{-}.\exists \mathsf{notificationConfig}^{-}.\mathsf{Bucket}$$

Example 6. *"There might be an* IAM::User *that is shared by two or more policies"*

$$\sqcup_{x \in inst(\mathsf{User})} \{x\} \sqcap \geq 2 \mathsf{users}^{-}.\mathsf{Policy}$$

Example 7. *"There might be an* EC2::SecurityGroup *that opens all ports to all"*

$$\sqcup_{x \in inst(\mathsf{SecurityGroup})} \{x\} \sqcap \exists \mathsf{ingress}.$$
$$(\exists \mathsf{cidrIp}.\{0.0.0.0/0\} \sqcap \exists \mathsf{fromPort}.\{0\} \sqcap \exists \mathsf{toPort}.\{65535\})$$

A.5.3 Global Protections

Also called *Exists-must* queries, more general than the previous, formulated as $\mathsf{X} \sqcap \varphi_{\mathsf{X}}$ with an interpretation of *ff-ff-tt* over the three outcomes UNSAT, SAT/0, and SAT/+ (corresponding to the natural semantics of DL simple concepts). These properties coincide with simple DL concept queries and are used to express that *there exists an instance of a concept* X *that certainly satisfies a property* φ_{X}. They are suitable for expressing global mitigations and configuration queries of simple facts not involving potentially underspecified resources. The DL queries are simply written as $\mathsf{X} \sqcap \varphi_{\mathsf{X}}$ and their interpretation over the three outcomes (UNSAT, SAT/0, SAT/+) is of False, False, and True, respectively.

Example 8. *"There is a* CloudWatch::Alarm *that must perform an action when triggered"*

$$\mathsf{Alarm} \sqcap \exists \mathsf{alarmActions}.\top$$

Example 9. *"There is a* CloudTrail::Trail *that logs global service events"*

$$\mathsf{Trail} \sqcap \exists \mathsf{isLogging}.\{tt\} \sqcap \exists \mathsf{includeGlobalEvents}.\{tt\}$$

A.6 StackSet 15 - Automated Checks Report

P/F	Property ID	T/F	Outc.	Outcome Description	Instances
PASS	01_AKM_BUCKETS_SHOULD_LOG	T	USAT	Either there are no S3::Buckets or there are some and they certainly keep logs	N/A
FAIL	02_AKM_NO_BUCKETS_STORING_OWN_LOGS	F	SAT+	There are S3::Buckets declared in this template that store their own logs	(configs3bucket)
PASS	03_AKM_BUCKET_STORING_LOGS_NOT_PUBLIC	T	USAT	No bucket that store logs and is public can be found	N/A
FAIL	04_AKM_BUCKETS_ENCRYPTED	F	SAT+	There are S3::Buckets - declared in this template - that are not encrypted	(configs3bucket - datas3bucket)
PASS	05_AKM_BUCKET_NOT_PUBL_UNL_WEB_CORS	T	USAT	No bucket that does not host a website or allow CORS and is public can be found	N/A
PASS	06_AKM_CRIT_LAMBDA_BUCKETS_ENCR_ROTAT	T	USAT	All S3::Bucket declared in the template are encrypted and keep logs whenever they store lambda code	N/A
FAIL	07_AKM_LOGS_STORED_ON_ENCR_BUCKETS	F	SAT+	There are S3::Buckets in this template that store logs and are not encrypted	(configs3bucket)
PASS	08_EM_ALARM_ACTION	T	SAT+	There are CloudWatch::Alarms that certainly perform an action when triggered	(cpureservationtoohighalarm memoryreservationtoohighalarm tracebackindartlogalarm subscriptionqueuedepthhigh subscriptionqueuedepthlow cpualarmhigh cpualarmlow)
FAIL	09_EM_CONFIG_RECORDER	F	USAT	There is no Config::ConfigurationRecorder that is recording changes of all resource types	N/A
FAIL	10_EM_GLOBAL_SERVICES_TRAIL	F	USAT	There is no CloudTrail::Trail that logs events from global services	N/A
PASS	11_EKM_SECURITYGROUP_ALL_PORTS_TO_ALL	F	USAT	Either there are no EC2::SecurityGroups or there are some and they certainly do not open all ports to all	N/A
PASS	12_EKM_SHARED_BY_MULTIPLE_POLICIES	F	USAT	Either there are no IAM::Users or there are some and they are certainly not shared by two or more policies	N/A
PASS	13_EKM_QUEUE_CRITICAL_NOT_ENCRYPTED	F	USAT	If there are SQS::Queues receiving notifications from a bucket - they certainly are encrypted	N/A
PASS	14_AKM_LAMBDA_FAILED_INPUT_TO_ENCRYPTED	T	USAT	Either there are no Lambda::Functions or they all send failed input to an encrypted resource	N/A
PASS	15_AKM_IAM_POLICIES_MUST_LINKED_TO_SMTH	T	USAT	Either there are no IAM::Policys or they are all attached to something	N/A
PASS	16_AKM_VPC_NO_DEFAULT_SECURITYGROUP	T	USAT	Either there are not VPC::EC2 or there are some and they are all associated with a security group	N/A
PASS	17_AKM_KEYS_ENABLED	T	USAT	All KMS::Keys are enabled	N/A
PASS	18_AKM_REPLICAS_ENCRYPTED	T	USAT	Either there are no bucket replicas or they are all encrypted	N/A
PASS	19_AKM_LAMBDAS_ENCRYPTED	T	USAT	Either there are no Lambda::Functions or they are all encrypted	N/A
PASS	20_EKM_SECURITYGROUP_OPENTOALL	F	USAT	Either there are no EC2::SecurityGroups or there are some and they are certainly not open to all IPs	N/A
PASS	21_EKM_TRAIL_CARRYING_NOTHING	F	USAT	Either there are no CloudTrail::Trails or there are some and they all certainly carry logs about data events	N/A
PASS	22_EKM_TRAIL_NO_LOGFILE_VALIDATION	F	USAT	Either there are no CloudTrail::Trails or there are some and they all certainly have log file validation enabled	N/A
PASS	23_AKM_TRAIL_LOG	T	USAT	Either there are no CloudTrail::Trails or there are some and they all certainly keep logs	N/A
PASS	24_EKM_TRAIL_NOT_ALL_REGIONS	F	USAT	Either there are no CloudTrail::Trails or there are some and they all certainly log events from all regions	N/A
PASS	25_EKM_VOLUME_NOT_ENCRYPTED	F	USAT	Either there are no EC2::Volumes or there are some and they are all certainly encrypted	N/A
PASS	26_EKM_INSTANCE_PUBLIC	F	USAT	Either there are not EC2::Instances or there are some and they certainly do not allow public IPs	N/A
PASS	27_EKM_INSTANCE_USERDATA	F	USAT	Either there are no EC2::Instances or they don't have user data on template	N/A
PASS	28_EKM_SECURITY_GROUP_OPEN_PORTS_SELF	F	USAT	Either there are no EC2::SecurityGroups or there are some and they certainly do not open all ports to self	N/A
PASS	29_EKM_SECURITYGROUP_UNUSED	F	USAT	Either there are no EC2::SecurityGroups or there are some and they are all linked to a VPC	N/A
FAIL	30_AKM_LOADBALANCER_ACCESSLOGGINGPOLICY	F	SAT+	There is at least one ElasticLoadBalancing::LoadBalancer that certainly has Access Logging Policy disabled	(elasticloadbalancer - elasticloadbalancer)
PASS	31_AKM_LOADBALANCER_S3ACCESSLOGS	T	USAT	Either there is no ElasticLoadBalancingv2::LoadBalancer or they all have S3 Access Logs enabled	N/A
PASS	32_AKM_LOADBALANCER_DELETION_PROTECTED	T	USAT	All ElasticLoadBalancingv2::LoadBalancers - if any - certainly have deletion protection enabled	N/A
PASS	33_AKM_LISTENER_NO_OLD_POLICIES	T	USAT	All LoadBalancingV2::Listeners - if any - certainly do not use old SSL policies	N/A
PASS	34_AKM_POLICIES_ATTACHED_TO_GROUPS_ONLY	T	USAT	Either there are no IAM::Policies or they are all attached to groups	N/A
PASS	35_AKM_GROUP_NO_USERS	T	USAT	All IAM::Groups - if any - are linked to some user	N/A
PASS	36_AKM_USERS_PWDRESET	T	USAT	All known IAM::Users have password reset required	N/A
PASS	37_EKM_USER_WITH_LOGINPROFILE	F	USAT	Either there are no IAM::Users or they all have password in template	N/A
PASS	38_AKM_ACCESSKEY_ROTATING	T	USAT	Either there are no IAM::Users or they are all linked to a key that is rotating	N/A
PASS	39_AKM_USER_ONE_ACCESSKEY	T	USAT	All IAM::Users have max one access key	N/A
FAIL	40_AKM_RDS_BACKUP	F	SAT+	There exists at least one declared RDS::DBInstance that has backup retention set to 0	(dbinstance)
PASS	41_AKM_RDS_MINORUPGRADE_DISABLED	T	USAT	Either there are no RDS::DBInstances or they all have auto minor upgrade disabled	N/A
PASS	42_AKM_RDS_SHORT_BACKUP	T	USAT	All RDS::DBInstances - if any - certainly do not have short backup retention	N/A
FAIL	43_AKM_DECLARED_RDS_ONEAZ	F	SAT+	There is at least one RDS::DBInstance that is certainly not replicated in different Availability Zones	(dbinstance)
FAIL	44_AKM_RDS_ENCRYPTED	F	SAT+	There exists at least one declared RDS::DBInstance that is not encrypted	(dbinstance)
PASS	45_EKM_RDS_SECURITYGROUP_OPEN	F	USAT	If there are RDS::DBInstances - they are certainly not linked to an open security group	N/A
FAIL	46_AKM_BUCKETS_VERSIONING	F	SAT+	There are S3::Buckets - declared in this template - that have versioning disabled	(configs3bucket - datas3bucket)
PASS	47_EKM_VPC_OPEN_NETWORK_ACL	F	USAT	If there are EC2::VPCs - they certainly do not have an open-to-all Network Acl	N/A
PASS	48_EKM_NETWORKACL_NOT_USED	F	USAT	Either there are not EC2::NetworkAcl or there are some and they are all linked to a VPC	N/A
PASS	49_EKM_SUBNET_BAD_ACL	F	USAT	Either there are no EC2::Subnets or there are some and they all have good ACLs	N/A
PASS	50_EKM_SUBNET_NO_FLOWLOG	F	USAT	If there are EC::Subnets - they all have FlowLog	N/A

Chapter 3

Paper B

This chapter is based on

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

written by

Claudia Cauli, Magdalena Ortiz, and Nir Piterman,

published in

Proceedings of the 18th International Conference on Principles of Knowledge Representation and Reasoning (KR), 2021.

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.

3.1 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 [82], 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 [20, 83] 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. 3.2 we motivate the choices made in the contributions put forth by this paper. In Sec. 3.3 we review the background on DL-Lite$^{\mathcal{F}}$ and conjunctive queries. In Sec. 3.4 and 3.5 we introduce core-closed KBs and study KB satisfiability. In Sec. 3.6 and 3.7 we discuss conjunctive query entailment and satisfiability. In Sec. 3.8 we present our security queries. We then discuss related work (Sec. 3.9) and conclude in Sec. 3.10. Results and proofs that are omitted in this paper are found in the full version.

3.2 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 $\mathsf{Storage}$ ($\mathcal{T}$), 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

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

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"*:

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

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 [84], the examples discussed here are very close to real IaC deployments' encoding and to the properties that are of interest for a security review.

3.3 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 $\mathcal{I} = (\Delta^{\mathcal{I}}, \mathsf{fun}^{\mathcal{I}})$, where $\Delta^{\mathcal{I}}$ is a non-empty domain and $\mathsf{fun}^{\mathcal{I}}$ is an interpretation function. The function $\mathsf{fun}^{\mathcal{I}}$ 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.

$$
\begin{aligned}
\bot^{\mathcal{I}} &= \emptyset \\
(\neg \mathsf{B})^{\mathcal{I}} &= \Delta^{\mathcal{I}} \setminus \mathsf{B}^{\mathcal{I}} \\
(\mathsf{R}^-)^{\mathcal{I}} &= \{(a, b) \mid (b, a) \in \mathsf{R}^{\mathcal{I}}\} \\
(\exists \mathsf{P})^{\mathcal{I}} &= \{a \mid \exists b \in \Delta^{\mathcal{I}}.(a, b) \in \mathsf{P}^{\mathcal{I}}\ \}
\end{aligned}
$$

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.

3.4 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 3.2, 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 3.7.1, 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 $\mathsf{fun}^{\mathcal{I}}$. 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 [85]. We measure data complexity in terms of the model $\mathcal{M}$, which we expect to be much larger than $\mathcal{A}$.

3.5 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 [31] 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 $\mathsf{P} = \mathsf{R}^-$.

3.5.1 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* α *applicable in* $\mathcal{Y}_j$ *to* f
let α *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* $\mathrm{bchase}_i(\mathcal{K},\mathcal{M})$ *be* $\mathcal{K}$*'s* i*-th boundary chase. Then,* $\mathrm{bchase}_i(\mathcal{K},\mathcal{M})$ *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:*

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

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 $\cdot^{lts(\mathcal{M})} \bigcup \cdot^{db(\mathcal{A})} = \cdot^{can_0(\mathcal{K})}$.

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 $f \in \mathcal{M}$. 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 $(F_{\mathcal{T}}, \mathcal{A}, F_{\mathcal{S}}, \mathcal{M})$.*

3.5.2 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:*

[A] All NIs in $\mathcal{T} \cup \mathcal{S}$ are in $cln(\mathcal{T} \cup \mathcal{S})$;

[B] All Fs in $\mathcal{T} \cup \mathcal{S}$ are in $cln(\mathcal{T} \cup \mathcal{S})$;

[C] 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 $\mathsf{B}_1 \sqsubseteq \neg \mathsf{B}_3 \in cln(\mathcal{T} \cup \mathcal{S})$;

[D] 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 analogues 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 $cln(\mathcal{T} \cup \mathcal{S}) \models \alpha$.*

3.5.3 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:

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

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 (K):
2     q_unsat ::= ⊥;
3     foreach α ∈ cln(T ∪ S) do
4         q_unsat ::= q_unsat ∨ δ(α);
5     if q_unsat^{db(A)∪lts(M)} = ∅ 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* Consistent($\mathcal{K}$) *terminates, and $\mathcal{K}$ is satisfiable iff* 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.*

3.6 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 [31] 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 [31]. 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 [31].

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 (K,q):
2     if not Consistent(K) then
3         return AllTup(q,K)
4     return PerfectRef(q, T ∪ S)^{db(A)∪lts(M)};
```

3.7 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.

3.7.1 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 3.4). 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* $(\mathsf{func}\ \mathsf{P}) \in \mathcal{T} \cup \mathcal{S}$ 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* $(\mathsf{func}\ \mathsf{P}) \in \mathcal{T} \cup \mathcal{S}$ 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.*

3.7.2 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 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*

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. $\mathcal{K}$

def Sat *($\mathcal{K}$,conj)*:
  if *(conj contains* $x \not\approx x$*) or* $(conj^{\star}_{\mathcal{M}} \not\subseteq \mathcal{M})$ **then**
    return *false*;
  $conj ::= conj \setminus conj_{\mathcal{M}}$;
  if $conj == \emptyset$ **then**
    return *true*;
  if *there is at* $\leftarrow conj^{\star}_{\mathcal{M}?}$ *with free variable* y **then**
    for a *s.t.* $at[a/y] \in \mathcal{M}$ **do**
      if $\mathsf{Sat}(\mathcal{K}, conj[a/y])$ **then**
        return *true*;
    return *false*;
  if *there is at* $\leftarrow conj^{\star}_{?}$ *with free variables* $\vec{y}$ **then**
    for $\vec{a}$ *s.t.* $at[\vec{a}/\vec{y}] \in \mathcal{M}$ **do**
      if $\mathsf{Sat}(\mathcal{K}, conj[\vec{a}/\vec{y}])$ **then**
        return *true*;
    return $\mathsf{Sat}(\mathcal{K}, conj[\vec{a^y}/\vec{y}])$;
  if *there is at* $\leftarrow conj_{\mathcal{A}?}$ *with free variable* y **then**
    for $a \in \mathbf{I}^{\mathcal{M}}$ **do**
      if $\mathsf{Sat}(\mathcal{K}, conj[a/y])$ **then**
        return *true*;
    return $\mathsf{Sat}(\mathcal{K}, conj[a^y/y])$;
  if *there is at* $\leftarrow conj^{\not\approx}_{?}$ **then**
    return $\mathsf{Sat}(\mathcal{K}, conj[\vec{a^y}/\vec{y}])$;
  $conj ::= conj \setminus conj^{\not\approx}_{\mathcal{M}?}$;
  $conj_{\mathcal{A}} ::= conj_{\mathcal{A}} \cup \{a \not\approx b \mid a, b \in \mathbf{I}^{B} \cup \mathbf{I}^{\mathcal{K}'} \wedge a \neq b\}$;
  return $\mathsf{sat}^{\not\approx}_{\mathsf{noUNA}}(\mathcal{T}, \mathcal{A} \cup conj_{\mathcal{A}}, \mathcal{S}, \mathcal{M})$

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 (K,q):
2     if not Consistent(K) then
3         return ∅
4     return { c⃗ | c⃗ ∈ AllTup(q,K) ∧ Sat(K, q[c⃗]) = tt };
```

3.8 Must/May Queries

As introduced in Section 3.2, 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:

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

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.*

3.9 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., [86–89]. One of the most prominent approaches is to extend DLs with *closed predicates* [88], 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 [20] and conjunctive query entailment 2ExpTime-hard [90]. 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 $\mathcal{EL}$ [91]. [92] 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 [93] 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 [94], as done in [80], but this was not in the scope of the application presented in this paper.

3.10 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 TBox 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.

B Appendix

B.1 Proofs of Section 3.5

Proof of Lemma 1

Proof. By construction, $bchase_i(\mathcal{K},\mathcal{M})$ is built by adding fresh assertions to the set $\mathcal{Y}_{i-1}$ according to the rules in Definition 2. It is easy to see from the cases **c1-c8** that none of these fresh assertions is an $\mathcal{M}$-assertion. □

Proof of Lemma 3

Proof. Since we have already shown that $can(\mathcal{K})$ is a model of $\mathcal{M}$, we need to show three things: *(1)* $can(\mathcal{K})$ satisfies all assertions in $\mathcal{A}$, *(2)* $can(\mathcal{K})$ satisfies all positive axioms in $PI_{\mathcal{T}}$, and *(3)* $can(\mathcal{K})$ satisfies all positive axioms in $PI_{\mathcal{S}}$. *(1)* follows from the fact that $can(\mathcal{K})$ contains $\mathcal{A}$. To prove *(2)* and *(3)*, we need to proceed by contradiction considering all cases for axioms in $PI_{\mathcal{T}}$ and $PI_{\mathcal{S}}$ and the fact that $\mathcal{M}$ is assumed to be consistent w.r.t. $\mathcal{S}$. The proof is similar to that of Lemma 7 in [31]. In particular, let us note that the rules **c1-c8**, that are used in the construction of the bchase, cover all cases of positive axioms in $\mathcal{T} \cup \mathcal{S}$. For each of these cases, the rule is triggered when an assertion that should be logically implied by another, according to the implication established by the corresponding rule's axiom, is missing. Hence, for every axiom $\alpha = \mathsf{B}_1 \sqsubseteq \mathsf{B}_2$, it is possible to prove by contradiction that if an object a such that $\mathsf{B}_1(a) \in can(\mathcal{K}) \wedge \mathsf{B}_2(a) \notin can(\mathcal{K})$ exists, then the corresponding rule would have been triggered, and the assertion $\mathsf{B}_2(a)$ added to $can(\mathcal{K})$, leading to a contradiction. While the above statement is valid for all the nodes in $\mathbf{I}^{\mathcal{K}}$, it applies to nodes in $\mathbf{I}^{\mathcal{M}}$ only for rules **c1-c4**. For rules **c5-c8**, that do not apply to objects $a_{\mathcal{M}} \in \mathbf{I}^{\mathcal{M}}$, we need to recall that $\mathcal{M}$ is assumed to be consistent with respect to $\mathcal{S}$. □

Proof of Lemma 4

Proof. $\Rightarrow$ We show that $db(\mathcal{A}) \cup lts(\mathcal{M}) \models (F_{\mathcal{T}}, \mathcal{A}, F_{\mathcal{S}}, \mathcal{M})$ if $can(\mathcal{K}) \models (F_{\mathcal{T}}, \mathcal{A}, F_{\mathcal{S}}, \mathcal{M})$. Since $can(\mathcal{K})$ is built from the union of $lts(\mathcal{M})$, which is the model of $\mathcal{M}$, and of the $bchase(\mathcal{K}, \mathcal{M})$, which contains $\mathcal{A}$, it follows that if a functionality axiom is satisfied by $can(\mathcal{K})$ then it must be satisfied by the union $db(\mathcal{A}) \cup lts(\mathcal{M})$ of its components's minimal interpretations.

$\Leftarrow$ We show that if $db(\mathcal{A}) \cup lts(\mathcal{M}) \models (F_{\mathcal{T}}, \mathcal{A}, F_{\mathcal{S}}, \mathcal{M})$ then $can(\mathcal{K}) \models (F_{\mathcal{T}}, \mathcal{A}, F_{\mathcal{S}}, \mathcal{M})$. We proceed by induction on the construction of the canonical model, when seen by progressing through the construction of the chase.
Base Step. We have that $can_0(\mathcal{K}) = (\Delta^{can(\mathcal{K})}, \cdot^{can_0(\mathcal{K})})$ where $\cdot^{can_0(\mathcal{K})} = \cdot^{lts(\mathcal{M})} \cup \cdot^{db(\mathcal{A})}$ and since by assumption $\cdot^{lts(\mathcal{M})} \cup \cdot^{db(\mathcal{A})} \models (F_{\mathcal{T}}, \mathcal{A}, F_{\mathcal{S}}, \mathcal{M})$ then $can_0(\mathcal{K}) \models (F_{\mathcal{T}}, \mathcal{A}, F_{\mathcal{S}}, \mathcal{M})$.
Inductive Step. Let us assume by contradiction that there exists an i such that $can_i(\mathcal{K})$ is a model of $(F_{\mathcal{T}}, \mathcal{A}, F_{\mathcal{S}}, \mathcal{M})$ but $can_{i+1}(\mathcal{K})$ is not. According to this assumption, there is an axiom α that when applied to an assertion $f \in can_i(\mathcal{K})$, following one of the rules **c1-c8**, violates a functionality assertion. The only rules that add fresh role assertions, and that can therefore violate functionality, are the rules **c3**, **c4**, **c7**, and **c8**. Let us consider the cases **c3**

and **c4**. In both cases, we have $can_{i+1}(\mathcal{K}) = can_i(\mathcal{K}) \cup \{\mathsf{ga}(\mathsf{P}_\mathcal{K}, a, a_{new})\}$ where a_{new} is a new constant symbol introduce according to the total order in the set $\mathbf{I}^+$. Since $can_{i+1}(\mathcal{K})$ is not a model of $(F_\mathcal{T}, \mathcal{A}, F_\mathcal{S}, \mathcal{M})$, there must exist at least a functionality axioms α that is not satisfied by $can_{i+1}(\mathcal{K})$. However, all three following cases lead to a contradiction:

- If $\alpha = \mathsf{Func}(\mathsf{P}_\mathcal{K})$, there must exists pairs (x, y) and (x, z) of objects in $\mathsf{P}_\mathcal{K}^{can_{i+1}(\mathcal{K})}$ s.t. $y \neq z$. However, since rule **c3** or **c4** was applied, then no other $\mathsf{P}_\mathcal{K}$-assertion departing from a was contained in $can_i(\mathcal{K})$ and only $\mathsf{ga}(\mathsf{P}_\mathcal{K}, a, a_{new})$ is added to $can_{i+1}(\mathcal{K})$. Therefore, the functionality axiom must have been not satisfied in $can_i(\mathcal{K})$, which is a contradiction.
- If $\alpha = \mathsf{Func}(\mathsf{P}_\mathcal{K}^-)$, there must exists pairs (y, x) and (z, x) of objects in $\mathsf{P}_\mathcal{K}^{can_{i+1}(\mathcal{K})}$ s.t. $y \neq z$. Since a_{new} is fresh in $can_{i+1}(\mathcal{K})$, there could not have been another pair (a', a_{new}) with $a \neq a'$ contained in $can_i(\mathcal{K})$ and therefore this pair is also not in $can_{i+1}(\mathcal{K})$. Therefore, the functionality axiom must have been not satisfied in $can_i(\mathcal{K})$, which is a contradiction.
- If $\alpha = \mathsf{Func}(\mathsf{P}')$ with $\mathsf{P}' \neq \mathsf{P}_\mathcal{K}$, we would conclude that the axiom is already not satisfied in $can_i(\mathcal{K})$, but this would lead to a contradiction.

Let us now consider the rules **c7** and **c8**. In both cases, we have that $can_{i+1}(\mathcal{K}) = can_i(\mathcal{K}) \cup \{\mathsf{ga}(\mathsf{P}_\mathcal{S}, a_\mathcal{K}, a_{new})\}$. Similarly to the above, all three following cases lead to a contradiction:

- If $\alpha = \mathsf{Func}(\mathsf{P}_\mathcal{S})$, then there must exists pairs (x, y) and (x, z) of objects in $\mathsf{P}_\mathcal{S}^{can_{i+1}(\mathcal{K})}$ s.t. $y \neq z$. However, since rule **c7** or **c8** was applied, then no other $\mathsf{P}_\mathcal{S}$-assertion departing from $a_\mathcal{K}$ was contained in $can_i(\mathcal{K})$ (neither in $\mathcal{X}$ nor in $\mathcal{Y}$). Therefore, the functionality axiom must have been not satisfied in $can_i(\mathcal{K})$, which is a contradiction.
- If $\alpha = \mathsf{Func}(\mathsf{P}_\mathcal{S}^-)$, there must exists pairs (y, x) and (z, x) of objects in $\mathsf{P}_\mathcal{S}^{can_{i+1}(\mathcal{K})}$ s.t. $y \neq z$. Since a_{new} is fresh in $can_{i+1}(\mathcal{K})$, no other pair (a', a_{new}) with $a' \neq a_\mathcal{K}$ could have been contained in $can_i(\mathcal{K})$ and, therefore, is also not contained in $can_{i+1}(\mathcal{K})$. Therefore, the functionality axiom must have been not satisfied in $can_i(\mathcal{K})$, which is a contradiction.
- If $\alpha = \mathsf{Func}(\mathsf{P}')$ with $\mathsf{P}' \neq \mathsf{P}_\mathcal{S}$, we would conclude that the axiom is already not satisfied in $can_i(\mathcal{K})$, but this would lead to a contradiction.

□

Proof of Lemma 5

Proof. The proof follows as, by construction, all assertions in $cln(\mathcal{T} \cup \mathcal{S})$ are logically implied by $\mathcal{T} \cup \mathcal{S}$. □

Proof of Lemma 6

Proof. ⇒ We show that if $can(\mathcal{K})$ is a model of $\mathcal{K}$, then $db(\mathcal{A}) \cup lts(\mathcal{M})$ is a model of $cln(\mathcal{T} \cup \mathcal{S})$, $\mathcal{A}$, and $\mathcal{M}$. Since $can(\mathcal{K})$ models $\mathcal{K}$ and, by Lemma 5, all assertions in $cln(\mathcal{T} \cup \mathcal{S})$ are logically implied by $\mathcal{K}$, then $can(\mathcal{K})$ also models $cln(\mathcal{T} \cup \mathcal{S})$. Let us recall that, for every atomic concept $\mathsf{A} \in \mathbf{C}$ we have that $\mathsf{A}^{db(\mathcal{A})} \cup \mathsf{A}^{lts(\mathcal{M})} = \mathsf{A}^{can_0(\mathcal{K})} \subseteq \mathsf{A}^{can(\mathcal{K})}$, and for every atomic role

$\mathsf{R} \in \mathbf{R}$ we have that $\mathsf{R}^{db(\mathcal{A})} \cup \mathsf{R}^{lts(\mathcal{M})} = \mathsf{R}^{can_0(\mathcal{K})} \subseteq \mathsf{R}^{can(\mathcal{K})}$. Since $can(\mathcal{K})$ is a model of $cln(\mathcal{T} \cup \mathcal{S})$; since the restriction $can(\mathcal{K})$ to $can_0(\mathcal{K})$ only affects extensions in the *bchase* and does not affect extensions in $lts(\mathcal{M})$; and since the nature of functionality and negative inclusion axioms is such that they cannot be contradicted by restricting the *bchase* extension of atomic concepts and roles, then we can conclude that $db(\mathcal{A}) \cup lts(\mathcal{M})$ is a model of $cln(\mathcal{T} \cup \mathcal{S})$.

$\Leftarrow$ We show that if $db(\mathcal{A}) \cup lts(\mathcal{M})$ is a model of $cln(\mathcal{T} \cup \mathcal{S})$, $\mathcal{A}$, and $\mathcal{M}$, then $can(\mathcal{K})$ is a model of $\mathcal{K}$. To do so, we need to prove that $can(\mathcal{K})$ is a model of the functionality and negative inclusion axioms in $\mathcal{K}$ (since, by Lemma 3, $can(\mathcal{K})$ is always a model of the positive inclusion axioms in $\mathcal{K}$, we do not need to include those in the proof). We prove the latter statement by showing that $can(\mathcal{K})$ is a model of $cln(\mathcal{T} \cup \mathcal{S})$, $\mathcal{A}$, and $\mathcal{M}$. That is, we prove that if $db(\mathcal{A}) \cup lts(\mathcal{M})$ is a model of $cln(\mathcal{T} \cup \mathcal{S})$, $\mathcal{A}$, and $\mathcal{M}$, then $can(\mathcal{K})$ is a model of it too. We do so by induction on the construction of the $bchase(\mathcal{K})$.
Base Step. By construction, we have $can_0(\mathcal{K}) = (\Delta^{can(\mathcal{K})}, \cdot^{can_0}(\mathcal{K}))$ where $\cdot^{can_0}(\mathcal{K}) = (\cdot^{lts(\mathcal{M})} \cup \cdot^{db(\mathcal{A})})$. Therefore, $\mathsf{A}^{can_0(\mathcal{K})} = \mathsf{A}^{lts(\mathcal{M})} \cup \mathsf{A}^{db(\mathcal{A})}$, for every atomic concept A; and $\mathsf{R}^{can_0(\mathcal{K})} = \mathsf{R}^{lts(\mathcal{M})} \cup \mathsf{R}^{db(\mathcal{A})}$, for every atomic role R. Given that by assumption $db(\mathcal{A}) \cup lts(\mathcal{M})$ is a model of $cln(\mathcal{T} \cup \mathcal{S})$, $\mathcal{A}$, and $\mathcal{M}$, it follows that $can_0(\mathcal{K})$ is also a model for $cln(\mathcal{T} \cup \mathcal{S})$, $\mathcal{A}$, and $\mathcal{M}$.
Inductive Step. Let us assume by contradiction that $can_i(\mathcal{K})$ is a model of $cln(\mathcal{T} \cup \mathcal{S})$, $\mathcal{A}$, and $\mathcal{M}$, and $can_{i+1}(\mathcal{K})$ is not. Let us recall that $can_{i+1}(\mathcal{K})$ is obtained from its predecessor by applying one of the *bchase* rules **c1-c8**. In the following, we use the symbol B to refer to a complex DL-Lite$^{\mathcal{F}}$ concept, that is, a concept $\mathsf{B} = \mathsf{A}$ or $\mathsf{B} = \exists \mathsf{P}$. A rule is executed because a PI axiom $\mathsf{B}_1 \sqsubseteq \mathsf{B}_2 = \mathsf{A}_1 \sqsubseteq \mathsf{B}_2$ (resp. $= \exists \mathsf{P} \sqsubseteq \mathsf{B}_2$) is applicable to an assertion $\mathsf{A}_1(a)$ (resp. $\mathsf{ga}(\mathsf{P}, a, b)$) in $can_i(\mathcal{K})$. As a result of the rule's application, we have that $can_{i+1}(\mathcal{K}) = can_i(\mathcal{K}) \cup \{\mathsf{A}_2(a)\}$ if $\mathsf{B}_2 = \mathsf{A}_2$ or $can_{i+1}(\mathcal{K}) = can_i(\mathcal{K}) \cup \{\mathsf{ga}(\mathsf{P}', a, b)\}$ if $\mathsf{B}_2 = \exists \mathsf{P}'$. Since $can_i(\mathcal{K})$ is a model of $cln(\mathcal{T} \cup \mathcal{S})$ and $can_{i+1}(\mathcal{K})$ is not, there must exists a negative inclusion axiom $\alpha \in cln(\mathcal{T} \cup \mathcal{S})$ s.t. $can_i(\mathcal{K})$ models α and $can_{i+1}(\mathcal{K})$ does not model α. The axiom α must be of the form $\mathsf{B}_2 \sqsubseteq \neg \mathsf{B}_3$ or $\mathsf{B}_3 \sqsubseteq \neg \mathsf{B}_2$, while $can_{i+1}(\mathcal{K})$ must contain an assertion $\mathsf{A}_3(a)$ if $\mathsf{B}_3 = \mathsf{A}_3$, or $\mathsf{ga}(\mathsf{P}'', a, b)$ if $\mathsf{B}_3 = \exists \mathsf{P}''$. If such axiom exists in $cln(\mathcal{T} \cup \mathcal{S})$, then the set $cln(\mathcal{T} \cup \mathcal{S})$ must also contain the result of its combination with $\mathsf{B}_1 \sqsubseteq \mathsf{B}_2$ (the PI that triggers the application of rule); that is, the axiom $\mathsf{B}_1 \sqsubseteq \neg \mathsf{B}_3$ must also be in $cln(\mathcal{T} \cup \mathcal{S})$. If $\mathsf{B}_2 \neq \mathsf{B}_3$, then the axiom $\mathsf{B}_1 \sqsubseteq \neg \mathsf{B}_3$ would not be satisfied in $can_i(\mathcal{K})$, which leads to a contradiction. If $\mathsf{B}_2 = \mathsf{B}_3$, then $\mathsf{B}_2 \sqsubseteq \neg \mathsf{B}_2$ would not be satisfied in $can_i(\mathcal{K})$, which also leads to a contradiction.

□

Proof of Corollary 1

Proof. We first consider the case in which α is a NI axiom. Let us assume by contradiction that $\mathcal{T} \cup \mathcal{S} \models \alpha$ and $cln(\mathcal{T} \cup \mathcal{S}) \not\models \alpha$. We now show that from the fact that $cln(\mathcal{T} \cup \mathcal{S}) \not\models \alpha$ one can construct a model of $\mathcal{T} \cup \mathcal{S}$ that does not satisfy α, thus obtaining a contradiction.

Let us assume that $\alpha = \mathsf{A}_1 \sqsubseteq \neg\mathsf{A}_2$, and consider the DL-Lite$^{\mathcal{F}}$ KB $\mathcal{K} = \langle \mathcal{T}, \mathcal{A}, \mathcal{S}, \mathcal{M} \rangle$ where $\mathcal{M} = \emptyset$ and $\mathcal{A} = \{\mathsf{A}_1(a), \mathsf{A}_2(a)\}$. We show that $can(\mathcal{K})$ is the model that we are looking for, that is, the model such that $can(\mathcal{K}) \models \mathcal{T} \cup \mathcal{S}$ and $can(\mathcal{K}) \not\models \alpha$. The fact that $can(\mathcal{K}) \not\models \alpha$ is clear from the content of $\mathcal{A}$. We proceed to prove that $can(\mathcal{K}) \models \mathcal{T} \cup \mathcal{S}$. The only NIs that can be violated by $db(\mathcal{A}) \cup lts(\mathcal{M})$ are $\mathsf{A}_1 \sqsubseteq \neg\mathsf{A}_2$, its contrapositive form $\mathsf{A}_2 \sqsubseteq \neg\mathsf{A}_1$, $\mathsf{A}_1 \sqsubseteq \neg\mathsf{A}_1$, and $\mathsf{A}_2 \sqsubseteq \neg\mathsf{A}_2$. Since $\mathcal{M}$ is empty, we concentrate on $db(\mathcal{A})$. By assumption, we know that $cln(\mathcal{T} \cup \mathcal{S}) \not\models \mathsf{A}_1 \sqsubseteq \neg\mathsf{A}_2$ and, therefore, $cln(\mathcal{T} \cup \mathcal{S}) \not\models \mathsf{A}_2 \sqsubseteq \neg\mathsf{A}_1$. It follows that it cannot model neither the two remaining axioms $\mathsf{A}_1 \sqsubseteq \neg\mathsf{A}_1$, and $\mathsf{A}_2 \sqsubseteq \neg\mathsf{A}_2$. Hence, we can conclude that $db(\mathcal{A}) \models cln(\mathcal{T} \cup \mathcal{S})$ and therefore $db(\mathcal{A}) \models cln(\mathcal{T} \cup \mathcal{S}) \cup \mathcal{A} \cup \mathcal{M}$. From Lemma 6 it follows that $can(\mathcal{K})$ is a model of $\mathcal{K}$. The same claim can be proven for the cases where α has a different form.

We now consider the case in which α is a functionality axiom. In a similar way, we assume by contradiction that $\mathcal{T} \cup \mathcal{S} \models \alpha$ and $cln(\mathcal{T} \cup \mathcal{S}) \not\models \alpha$. We show that from $cln(\mathcal{T} \cup \mathcal{S}) \not\models \alpha$ one can construct a model of $\mathcal{T} \cup \mathcal{S}$ that does not satisfy α, obtaining a contradiction. The proof is similar to the one conducted for the case of NI, by using the ABox $\mathcal{A} = \{\mathsf{ga}(\mathsf{P}, a, b), \mathsf{ga}(\mathsf{P}, a, c)\}$. □

Proof of Lemma 7

Proof. ⇒ Follows from the definition of satisfiability for $\mathcal{K}$ KB BSs. If $can(\mathcal{K})$ is a model of $\mathcal{K}$, then $\mathcal{K}$ is obviously satisfiable.

⇐ We prove this direction by contraposition. We show that if $can(\mathcal{K})$ is not a model of $\mathcal{K}$, then $\mathcal{K}$ is not satisfiable. By Lemma 6, it follows that $db(\mathcal{A}) \cup lts(\mathcal{M}) \not\models (cln(\mathcal{T} \cup \mathcal{S}), \mathcal{A}, \mathcal{M})$, and therefore $db(\mathcal{A}) \cup lts(\mathcal{M}) \not\models cln(\mathcal{T} \cup \mathcal{S})$. This means that there exists a negative inclusion axiom α such that $db(\mathcal{A}) \cup lts(\mathcal{M}) \not\models \alpha$ and $cln(\mathcal{T} \cup \mathcal{S}) \models \alpha$. By Lemma 5, we have that $\mathcal{T} \cup \mathcal{S} \models \alpha$. Let us assume that α is of the form $\mathsf{A}_1 \sqsubseteq \neg\mathsf{A}_2$. Then, there exists $a \in \Delta^{db(\mathcal{A}) \cup lts(\mathcal{M})}$ such that $a \in \mathsf{A}_1^{db(\mathcal{A}) \cup lts(\mathcal{M})}$ and $a \in \mathsf{A}_2^{db(\mathcal{A}) \cup lts(\mathcal{M})}$. Let us now assume by contradiction that $\mathcal{K}$ is satisfiable and therefore there exists an interpretation $\mathcal{J} = (\Delta^{\mathcal{J}}, \cdot^{\mathcal{J}})$ that is a model of $\mathcal{K}$. We construct an isomorphism ψ from $\Delta^{db(\mathcal{A}) \cup lts(\mathcal{M})}$ to $\Delta^{\mathcal{J}}$ such that $\psi(a) = a^{\mathcal{J}}$ for each constant occurring in $\mathcal{A} \cup \mathcal{M}$. Since $\mathcal{J}$ is a model of $\mathcal{A} \cup \mathcal{M}$ it satisfies all their membership assertions, including $\psi(a) \in \mathsf{A}_1^{\mathcal{J}}$ and $\psi(a) \in \mathsf{A}_2^{\mathcal{J}}$. However, this makes the NI $\mathsf{A}_1 \sqsubseteq \neg\mathsf{A}_2$ be violated also in $\mathcal{J}$, contradicting the fact that $\mathcal{J}$ is a model of $\mathcal{K}$.

□

Proof of Corollary 2

Proof. Directly follows from Lemma 8. □

B.2 Proofs of Section 3.6

Proof of Theorem 2

Proof. Follows from the correctness of the corresponding procedure Answer in the standard setting, cf. [31], when considered over a single CQ. In particular, let us stress that core-closed KBs do not introduce any significant difference

with respect to the rewrite procedure and, in general, with respect to CQ entailment. □

Proof of Theorem 3

Proof. Follows from the complexity of the standard setting [31] □

B.3 Proofs of Section 3.7.1

Proof of Lemma 9

Proof. We now prove that $\mathcal{K} = \langle \mathcal{T}, \mathcal{A}, \mathcal{S}, \mathcal{M} \rangle$ is satisfiable under $\mathcal{A} - \mathsf{noUNA}$ if and only if $\mathcal{K}' = \langle \mathcal{T}', \mathcal{A}', \mathcal{S}, \mathcal{M} \rangle$ is satisfiable with the UNA.

$\Leftarrow$ If $\mathcal{K}'$ is satisfiable with the UNA, then all the KBs, including $\mathcal{K}$, that are obtained from it by following a procedure that is the inverse of the one presented above, are satisfiable under the $\mathcal{A} - \mathsf{noUNA}$.

$\Rightarrow$ If $\mathcal{K} = \langle \mathcal{T}, \mathcal{A}, \mathcal{S}, \mathcal{M} \rangle$ is satisfiable under the $\mathcal{A} - \mathsf{noUNA}$, then there exists at least one model $\mathcal{I} = (adom(\mathcal{M}) \uplus \mathbf{I}^{\mathcal{K}'} \uplus \mathbf{I}^{+}, \cdot^{\mathcal{I}})$, where $\mathbf{I}^{\mathcal{K}'} = \mathbf{I}^{\mathcal{K}} \setminus \mathbf{I}^{B}$, and a non-injective function $f : \mathbf{I}^{B} \cup \mathbf{I}^{\mathcal{K}'} \uplus \mathbf{I}^{+} \rightarrow \Delta^{\mathbf{I}^{B}} \uplus \Delta^{\mathbf{I}^{\mathcal{K}'} \uplus \mathbf{I}^{+}}$ s.t. $f : \mathbf{I}^{B} \rightarrow \Delta^{\mathbf{I}^{B}}$, that associates each constant from the set of open individual names $\mathbf{I}^{\mathcal{K}'} \uplus \mathbf{I}^{+}$ with an element from $\Delta^{\mathbf{I}^{B}} \uplus \Delta^{\mathbf{I}^{\mathcal{K}'} \uplus \mathbf{I}^{+}}$, and each constant name $\mathbf{I}^{B}$ with a boundary domain node $\Delta^{\mathbf{I}^{B}}$. This function can, thus, map open individuals onto boundary nodes. Since $\mathcal{I}$ is a model of $\mathcal{K}$, it satisfies all functional axioms in $\mathcal{T} \cup \mathcal{S}$. In particular, we have that for all functionality axioms $\alpha = (\mathsf{Func\ P}) \in \mathcal{T} \cup \mathcal{S}$, and for all a, a_k, a_j s.t. $a_j \in (\mathbf{I} \setminus adom(\mathcal{M}))$ and $a_k \in (\mathbf{I} \setminus \mathbf{I}^{\mathcal{M}})$, if $\mathcal{I} \models \mathsf{ga}(\mathsf{P}, a, a_k) \wedge \mathsf{ga}(\mathsf{P}, a, a_j)$ then $f(a_k) = f(a_j)$. Given that $\mathcal{I} \models \mathsf{ga}(\mathsf{P}, a, a_k) \wedge \mathsf{ga}(\mathsf{P}, a, a_j)$ if and only if $\{\mathsf{ga}(\mathsf{P}, a, a_k), \mathsf{ga}(\mathsf{P}, a, a_j)\} \subseteq \mathcal{A}$, then we have that $f(a_k) = f(a_j)$, for all a_k, a_j s.t. $a_j \in (\mathbf{I} \setminus adom(\mathcal{M}))$ and $a_k \in (\mathbf{I} \setminus \mathbf{I}^{\mathcal{M}})$, s.t $\exists(\mathsf{Func\ P}) \in \mathcal{T} \cup \mathcal{S}$ and a such that $\{\mathsf{ga}(\mathsf{P}, a, a_k), \mathsf{ga}(\mathsf{P}, a, a_j)\} \subseteq \mathcal{A}$. Let f be the minimal such function, and let f' be obtained from it by replacing each occurrence of a_k with a_j. Clearly, f' is injective, i.e., there is no pair x, x' of distinct objects such that $f'(x) = f'(x')$. Now, let $\mathcal{K}'$ be computed from $\mathcal{K}$ according to the procedure presented above. We have that $\mathcal{K}'$ is satisfiable under $\mathcal{A} - \mathsf{noUNA}$ and with function f'. But since f' is an injective function, then $\mathcal{K}'$ is satisfiable also under the UNA.

□

Proof of Theorem 4

Proof. The upper bound in combined complexity follows from Corollary 2 and from the fact that the construction of $\mathcal{K}'$ is done in polynomial time in $\mathbf{I}^{B}$ and $\mathbf{I}^{\mathcal{K}'}$. The lower bound in combined complexity follows from results in [30]. The data complexity follows from Corollary 2. □

Proof of Lemma 10

Proof. Consider a CQ-assertion $q[\vec{c}] = \exists \vec{y}.conj(\vec{c}, \vec{y})$.

$\Rightarrow$ If $q[\vec{c}]$ is satisfiable w.r.t. $\mathcal{K} = \langle \mathcal{T}, \mathcal{A}, \mathcal{S}, \mathcal{M} \rangle$ then there is an interpretation $\mathcal{I} = \langle \Delta^{\mathcal{I}}, \mathsf{fun}^{\mathcal{I}} \rangle$ such that $\mathcal{I}$ satisfies $q[\vec{c}]$. Let $\vec{e}$ be the assignment to $\vec{y}$ such that $conj(\vec{c}, \vec{e})$ is satisfied. We partition $\vec{e}$ to elements in $\mathbf{I}^{\mathcal{M}}$ or $\mathbf{I}^{\mathcal{K}}$, and elements not in these sets. Consider the assignment $\vec{d}$ obtained from $\vec{e}$ by replacing every element not in $\mathbf{I}^{\mathcal{M}} \cup \mathbf{I}^{\mathcal{K}}$ by a fresh constant. Let $conj_{\mathcal{M}}$ and $conj_{\mathcal{A}}$ be the sets arising from the grounding of $conj$ as $conj(\vec{d})$. By $\mathcal{I} \models^{\mathsf{CWA}} \mathcal{M}$, we conclude that $conj^{\star}_{\mathcal{M}} \subseteq \mathcal{M}$. The interpretation $\mathcal{I}'$ that enhances $\mathcal{I}$ by assigning the fresh constants appearing in $\vec{d}$ to the elements to which they are assigned by $\vec{e}$ shows that that $\mathcal{K}' = (\mathcal{T}, \mathcal{A} \uplus conj_{\mathcal{A}}, \mathcal{S}, \mathcal{M})$ is satisfiable without the UNA.

$\Leftarrow$ Consider the assignment $\vec{d}$ that grounds $conj$ to $conj_{\mathcal{M}}$ and $conj_{\mathcal{A}}$ and the interpretation $\mathcal{I}$ satisfying $\mathcal{K}'$. By definition, $\mathcal{I} \models^{\mathsf{CWA}} \mathcal{M}$ and $\mathcal{I}$ models $\mathcal{T}$, $\mathcal{S}$, and $\mathcal{A} \cup conj_{\mathcal{A}}$. We convert the assignment $\vec{d}$ to an assignment $\vec{e}$ that ranges over $\mathbf{I}^{\mathcal{M}}$, $\mathbf{I}^{\mathcal{K}}$ and the constants that are identified by $\mathcal{I}$ with the fresh constants appearing in $\vec{d}$. Using the assignment $\vec{e}$ and $\mathcal{I}$ we see that $q[\vec{c}]$ is satisfiable w.r.t. $\mathcal{K} = \langle \mathcal{T}, \mathcal{A}, \mathcal{S}, \mathcal{M} \rangle$.

□

Proof of Theorem 5

Proof. $\Rightarrow$ Suppose that $q[\vec{c}]$ is satisfiable w.r.t. $\mathcal{K}$ and assume by way of contradiction that the algorithm returns false.

We use the formulation of Lemma 10. Consider the assignment $\vec{d}$ guaranteed by the Lemma. We use $\vec{d}$ to guide the selection of the algorithm and show a contradiction.

In the case that $\vec{d}$ is the empty assignment there are no free variables in $conj$. The algorithm cannot return false in line 3 as the query itself is satisfiable by assumption. If the algorithm returns true in line 6, we are done. Otherwise, the algorithm proceeds to lines 24-26. There, we enforce the uniqueness of the pre-existing nodes $\mathbf{I}^{B}$ and $\mathbf{I}^{\mathcal{K}'}$ and call the satisfiability without unique name assumption. By assumption $q[\vec{c}]$ is satisfiable, which implies that the KB is satisfiable. It follows that the same interpretation can be used for the satisfiability without the unique name assumption in line 26.

Consider an $\mathcal{M}$-atom at appearing in $conj$ with a free variable y. The assignment $\vec{d}$ into at is later found in $\mathcal{M}$. It follows that for the free variable y in at there is an a such that $at[a/y] \in \mathcal{M}$ as required on line 8. It follows that returning false at line 11 is not possible.

Consider an atom at that refers to either a role $\mathsf{r} \in \mathbf{R}^{\mathcal{S}}$ or a concept $\mathsf{C} \in \mathbf{C}^{\mathcal{S}}$ and assigned by $\vec{d}$ with at least one value in $\mathbf{I}^{\mathcal{M}}$. It follows for the free variables $\vec{y}$ in at there is an assigment $\vec{a}$ such that $at[\vec{a}/\vec{y}] \in \mathcal{M}$. This matches the condition in line 13.

Consider another atom at that refers to either a role $\mathsf{r} \in \mathbf{R}^{\mathcal{S}}$ or a concept $\mathsf{C} \in \mathbf{C}^{\mathcal{S}}$ and is assigned by $\vec{d}$ with constants not in $\mathbf{I}^{\mathcal{M}}$. As we assume that the algorithm returns false, trying to match this atom by assigning it constants that match atoms in $\mathcal{M}$ (line 13) does not succeed. Thus, the

algorithm attempts to replace the variables in at by fresh constants (line 16). Notice that in the case that $\vec{d}$ assigns both variables in the same atom by one constant (not in $\mathbf{I}^{\mathcal{M}}$), this is not an issue as the satisfiability without unique name assumption (line 26) can unify the two fresh constants.
Consider an atom at that refers to either a role $\mathsf{r} \in \mathbf{R}^{\mathcal{K}}$ or a concept $\mathsf{C} \in \mathbf{C}^{\mathcal{K}}$ with a variable y that is assigned by $\vec{d}$ a constant in $\mathbf{I}^{\mathcal{M}}$. It follows that the loop in lines 18-20 tries to assign y with this value in $\mathbf{I}^{\mathcal{M}}$. As we assume that the algorithm returns false, if the algorithm tries to assign to this variable other values in $\mathbf{I}^{\mathcal{M}}$ this will fail. Notice that if the same variable appears in an atom that refers to predicates $\mathbf{R}^{\mathcal{S}}$ or $\mathbf{C}^{\mathcal{S}}$ then it would already have been replaced by lines 12-15 with a matching atom in $\mathcal{M}$.
Consider an atom at that refers to either a role $\mathsf{r} \in \mathbf{R}^{\mathcal{K}}$ or a concept $\mathsf{C} \in \mathbf{C}^{\mathcal{K}}$ with a variable y that is assigned by $\vec{d}$ a constant not in $\mathbf{I}^{\mathcal{M}}$. As we assume that the algorithm returns false, if the algorithm tries to assign to this variable values in $\mathbf{I}^{\mathcal{M}}$ in lines 17-20 this will fail. It follows that the variable y is replaced by a fresh constant (line 21).
When the algorithm reaches line 22 all the pure atoms have already been made to assertions by previous stages of the algorithm. The only atoms that may remain unassigned in $conj$ are inequalities where either one or both of the variables are unassigned. A variable y appearing in such an inequality is assigned in $\vec{d}$ to either a value not in $\mathbf{I}^{\mathcal{M}}$ or to a value in $\mathbf{I}^{\mathcal{M}}$. In either case, the algorithm assigns to such variables fresh constants. Consider a variable y appearing in inequalities that is assigned in $\vec{d}$ to a value not in $\mathbf{I}^{\mathcal{M}}$. The fresh constant replaced into this variable can be united by the satisfiability algorithm without unique name assumption to have the same interpretation as the value assigned to y by $\vec{d}$. Consider a variable y appearing in inequalities that is assigned in $\vec{d}$ to a value in $\mathbf{I}^{\mathcal{M}}$. Notice that the variable y appears *only* in inequalities as all variables in all pure atoms have been already replaced. Let Y be the set of fresh constants that are assigned by the algorithm to variables y that are assigned by $\vec{d}$ a value in $\mathbf{I}^{\mathcal{M}}$. The interpretation of such fresh constants by the satisfiability without the unique name assumption *cannot* be equal to $\mathbf{I}^{\mathcal{M}}$. However, an interpretation that keeps these fresh constants different from all other constants will satisfy the same inequalities.
Finally, the satisfiability without the unique name assumption is called in line 26. Consider the fresh constants appearing in $conj_{\mathcal{A}}$ in line 26. Those of the fresh constants were replaced into variables that are matched by $\vec{d}$ to values in $\mathbf{I}^{\mathcal{K}'} \cup \mathbf{I}^{B}$. It follows that the algorithm is free to unify them with the value assigned to them by $\vec{d}$. Those of the fresh constants that are in the set Y of variables appearing only in inequalities that are assigned in $\vec{d}$ to values in $\mathbf{I}^{\mathcal{M}}$ are left as unique fresh individuals.
We see that the resulting assignment is satisfiable by the same model that satisfies $\vec{d}$ as, indeed, the fresh constants in Y are compared under $\vec{d}$ to $\mathbf{I}^{\mathcal{M}}$ elements, which are different from all other elements. This is possible also for the fresh elements. It follows that the algorithm returns true leading to a contradiction.

$\Leftarrow$ Suppose that $\mathsf{Sat}(\mathcal{K}, q[\vec{c}])$ returns true. Let $\vec{d}/\vec{y}$ be the sequence of replacements done by the algorithm. Denote $conj = q[\vec{c}, \vec{d}/\vec{y}]$. That is, $conj$ is the

result of replacing the variables in $\vec{y}$ by the found constants $\vec{d}$. Clearly, we have $conj[\vec{d}/\vec{y}]$ is the union of $conj^{\star}_{\mathcal{M}}$, $conj^{\not\approx}_{\mathcal{M}}$, and $conj_{\mathcal{A}}$. That is, $conj^{\star}_{\mathcal{A}?}$, $conj^{\not\approx}_{\mathcal{A}?}$, $conj^{\star}_{\mathcal{M}?}$, and $conj^{\not\approx}_{\mathcal{M}?}$ are all empty.
By the check in line 2, we have that $x \not\approx x$ is not included in $conj$. Furthermore, $conj^{\star}_{\mathcal{M}} \subseteq \mathcal{M}$.
The final check (line 19) is to call $\mathsf{sat}^{\not\approx}_{\mathsf{noUNA}}(\mathcal{T}, \mathcal{A} \cup conj_{\mathcal{A}}, \mathcal{S}, \mathcal{M})$. This corresponds to the requirement in Lemma 10.

□

Proof of Corollary 3

Lemma 11. $\mathsf{Sat}(\mathcal{K}, q[\vec{c}])$ *is computed in* LogSpace *in* $\mathcal{M}$.

Proof. The algorithm can be reorganized by allocating $\lceil \log(|\mathcal{M}| + 1) \rceil$ bits per (a) atoms in $conj^{\star}_{\mathcal{M}?}$, (b) atoms in $conj^{\star}_{?}$, and (c) variables in $conj_{\mathcal{A}?}$. Storing these bits requires at most logarithmic space in $|\mathcal{M}|$.

The algorithm then enumerates for every atom (or variable) all the possible assertions in $\mathcal{M}$ (values in $\mathbf{I}^{\mathcal{M}}$) and tries the assignment that replaces the atom with the assertion (value).

This exhausts all the options to match assertions from $\mathcal{M}$ in lines 7-10 and 12-15 and values from $\mathbf{I}^{\mathcal{M}}$ in lines 17-20. Whenever all the options have been tried the variable can be replaced with a fresh constant.

For every such replacement, the algorithm has to solve an instance *sat* of satisfiability without the unique name assumption. By Lemma 9 *sat* is then converted to a "normal" satisfiability with the same $\mathcal{M}$ and $\mathcal{S}$ that is polynomial at most in $\mathbf{I}^{\mathcal{K}'}$ and $\mathbf{I}^{B}$. The latter satisfiability is first-order reducible. Thus, the entire algorithm requires logarithmic space in $\mathcal{M}$. □

Proof. The proof for the LogSpace data complexity follows from the above proof for Lemma 11. The combined complexity follows from the complexity of satisfiability over DL-Lite$^{\mathcal{F}}$ core-closed KBs under the $\mathcal{A}$-noUNA assumption.

□

Proof of Theorem 6

Proof. Follows from Theorem 5. □

B.4 Proofs of Section 3.8

Proof of Theorem 7

Proof. This follows from the respective complexity of query entailment and query satisfiability. □

Chapter 4

Paper C

This chapter is based on

Actions over Core-closed Knowledge Bases,

written by

Claudia Cauli, Magdalena Ortiz, and Nir Piterman,

to appear in

Proceedings of the 11th International Joint Conference on Automated Reasoning (IJCAR), 2022.

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.

4.1 Introduction

The use of automated reasoning techniques to analyze the properties of cloud infrastructure is gaining increasing attention [6, 65, 66, 95, 96]. 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 [82], 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 [20, 83], we found reasoning in such a framework to be not efficient at cloud scale. In [97], we introduced *core-closed knowledge 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 [82]).

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 Section 4.2, we provide background on core-closed knowledge bases, conjunctive queries, and Must/May queries. In Section 4.3, we motivate and introduce the notion of *core-completeness*. In Section 4.4, we define the action language. In Section 4.5, we describe the static verification problem and characterize its complexity. In Section 4.6, 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 Section 4.7 and conclude in Section 4.8.

4.2 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 [97], 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$^{\mathcal{F}}$ [30, 31] 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 [82], 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 ([97]) is a Boolean combination of nested UCQs in the scope of a MUST or a MAY operator as follows:

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

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 $\mathcal{K}$.

4.3 Core-complete Knowledge Bases

The algorithm Consistent presented in [97] 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 ([97], 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 10 (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 $\mathcal{M} = \{$S3::Bucket$(b)\}$, according to the partially-closed semantics of core-closed knowledge bases, the absence of an assertion* 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 12. *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 [97] 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 [97] 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 [97] 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 10. 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 13. *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:

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

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

4.4 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 (Section 4.2) 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.

4.4.1 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 11. *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* S3::Bucket(x) *and* 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 $\theta = [\ x \leftarrow DataBucket,\ y \leftarrow Private\ ]$, 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 12. *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 $\mathcal{M}_x$:

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

4.4.2 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. 4.1. 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

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

Figure 4.1. Semantic of the action language defined over the MBox $\mathcal{M}$ and set $\mathbf{I}^{\mathcal{M}}$.

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 Section 4.3 for the corresponding definitions). We illustrate two examples next.

Example 13 (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 14 (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:*

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

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.

4.5 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 6 (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 8 (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 Section 4.3, 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 $\mathcal{M}$ [97] which is, thus, also polynomial in $\mathcal{M}$. □

4.6 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 [98, 99]. 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 [100].

Transition Systems In the style of the work done in [101, 102], 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 7 (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 15. *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

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

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 14. *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 5, 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. 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 5 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 include the modified algorithm in Appendix C.1 and the proofs of the following Theorems in Appendices C.2 and C.3, respectively.

Theorem 9 (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*

Algorithm 5: 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, ⟨t⃗, q⟩):
2      Π := ∅;
3      S := ⊥;
4      AllPlanSearch(M_0, ϵ, ∅, K, D, Act, ⟨t⃗, q⟩) ;
5      return Π;
6  def AllPlanSearch ( M, π, V, K, D, Act, ⟨t⃗, q⟩):
7      if M ∈ V then
8          return;
9      if t⃗ ∈ ANS(q, ⟨T, A, S, M⟩) then
10         Π := Π ∪ {π};
11         return;
12     Q := ∅;
13     foreach ⟨γθ⃗, M′⟩ ∈ Successors(M, Act, D) do
14         Q.push(⟨γθ⃗, M′⟩);
15     V := V ∪ {M};
16     while Q ≠ ∅ do
17         ⟨γθ⃗, M′⟩ = Q.pop();
18         AllPlanSearch(M′, π · γθ⃗, V, K, D, Act, ⟨t⃗, 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, θ⃗ ∈ D^{ar(γ)} do
26         M′ := T_{γθ⃗}(M);
27         if M′ is fully satisfiable then
28             N := N ∪ {⟨γθ⃗, M′⟩}
29     S[M] := N;
30     return N;
```

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

4.7 Related Work

The syntax of the action language that we presented in this paper is similar to that of [100, 103, 104]. 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 [102] and, in general, to ABox updates [105, 106]. Like the frameworks introduced in [101, 107–109], our actions are parameterized and when combined with a core-closed knowledge base generate a transition system. In [108], the authors focus on a variant of *Knowledge and Action Bases* [102] 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 [110], 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 [108], 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 [111]. 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 [101] and [109], 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.

4.8 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.

C Appendix

C.1 FindPlan algorithm

C.2 Proof of FindPlans Correctness

Proof. $\Leftarrow$ Let us assume that the sequence π is a minimal plan from $\mathcal{K}$ to $\langle \vec{t}, q \rangle$. Since π is a plan, then it is a consistent sequence of grounded actions that generates a run $\rho = \mathcal{M}_0 \xrightarrow{\gamma_1 \vec{\theta}_1} \mathcal{M}_1 \xrightarrow{\gamma_2 \vec{\theta}_2} \cdots \xrightarrow{\gamma_n \vec{\theta}_n} \mathcal{M}_n$ over the transition system $\Upsilon_{\mathcal{K}}$, terminating in a state $\mathcal{M}_n$ such that $\vec{t} \in \mathsf{ANS}(q, \langle \mathcal{T}, \mathcal{A}, \mathcal{S}, \mathcal{M}_n \rangle)$. Since π is minimal, there is no index $i < n$ in the run ρ s.t. $\vec{t} \in \mathsf{ANS}(q, \langle \mathcal{T}, \mathcal{A}, \mathcal{S}, \mathcal{M}_i \rangle)$, which also implies that the run does not have any loops. The algorithm starts from $\mathcal{M}_0$, and recursively explores all consistent runs (via the Successors procedure, lines 21-30), marking the current state as visited before the recursive invocation (line 15), and skipping already visited states when re-entering the AllPlanSearch function (therefore avoiding loops within a given explored path, lines 7-8). The search space of the algorithm includes ρ and the node $\mathcal{M}_n$ will be reached. Finally, since the condition over $\mathcal{M}_n$ is true (line 9), then the sequence π explored so far is added to set Π (line 10), and will ultimately be returned by the FindPlans function (line 4).

$\Rightarrow$ Let us assume that a sequence π is returned by Algorithm 5 but that π is not a minimal plan from $\mathcal{K}$ to $\langle \vec{t}, q \rangle$. The sequence π is either (1) not a valid sequence, or (2) not a consistent sequence, or (3) does not reach a state satisfying the goal, or (4) is not minimal. The algorithm starts with an empty sequence ϵ (line 4) and appends a new grounded action $\gamma\vec{\theta}$ to the sequence only upon recursive invocation of the AllPlanSearch function. The grounded action $\gamma\vec{\theta}$ is returned by the Successors function, which explores only valid grounded actions and returns a pair $\left\langle \gamma\vec{\theta}, \mathcal{M}' \right\rangle$ only if $\mathcal{M}'$ is fully satisfiable (line 27). Since every prefix of π is built by appending valid grounded actions that lead to a fully-satisfiable $\mathcal{M}'$ then the whole sequence π is valid and consistent (contradicting *(1)* and *(2)*). The sequence π is returned by the algorithm, therefore it was inserted in the set Π at line 10 and the goal condition was satisfied (contradicting *(3)*). Since the algorithm returns after adding any sequence π to the set Π then there cannot exists an extension of such sequence (contradicting *(4)*).

□

C.3 Proof of FindPlans Complexity

Proof. The algorithm implements a depth-first search over a graph whose nodes are fully satisfiable MBoxes and edges are labeled by grounded actions. Nodes are never visited twice along the same recursion path, but can be re-visited along different recursion paths. Let us assume that the maximum number n of input parameters that an action can have is known; and let us also assume that $n \ll \mathcal{D}$ and generally independent of either $\mathcal{D}$ or $\mathcal{M}$. Each node can have at most $|\mathsf{Act}| \cdot |\mathcal{D}|^n$ successors (line 25) and the recursion depth of the graph exploration is linear in $\mathcal{D}$. We have that the number of nodes is bounded by

Algorithm 6: FindPlan($\mathcal{K}, \mathcal{D}, \mathsf{Act}, \langle \vec{t}, q \rangle$)

Inputs : A core-closed KB $\mathcal{K} = (\mathcal{T}, \mathcal{A}, \mathcal{S}, \mathcal{M}_0)$, a domain $\mathcal{D}$, a set of ungrounded action labels Act and a pair $\langle \vec{t}, q \rangle$ of an answer tuple and a MUST/MAY query

Output : NULL if a plan does not exist, a consistent plan π otherwise

```
1  def FindPlan ( K, D, Act, ⟨t⃗, q⟩):
       // V and S have global scope
2      V := ∅;
3      S := ⊥;
4      return PlanSearch(M_0, ϵ, K, D, Act, ⟨t⃗, q⟩);
5  def PlanSearch (M, π, K, D, Act, ⟨t⃗, q⟩):
6      if M ∈ V then
7          return NULL;
8      V := V ∪ {M};
9      if t⃗ ∈ ANS(q, ⟨T, A, S, M⟩) then
10         return π;
11     foreach ⟨γθ⃗, M′⟩ ∈ Successors(M, Act, D) do
12         π′ = PlanSearch(M′, π · γθ⃗, K, D, Act, ⟨t⃗, q⟩);
13         if π′ ≠ NULL then
14             return π′;
15     return NULL;
16 def Successors (M, Act, D):
17     if S[M] is defined then
18         return S[M];
19     N := ∅;
20     foreach γ ∈ Act do
21         foreach θ⃗ ∈ D^ar(γ) do
22             M′ := T_γθ⃗(M);
23             if M′ is fully satisfiable then
24                 N := N ∪ {⟨γθ⃗, M′⟩}
25     S[M] := N;
26     return N;
```

a function that is exponential in the size of the domain $\mathcal{D}$. The nodes are computed on the fly in the Successors sub-procedure, which checks whether the current MBox is fully satisfiable (check done in polynomial time in $\mathcal{M}$ and in $\mathcal{D}$) and stores it. The AllPlanSearch algorithm explores the graph looking for minimal paths achieving a goal assertion. There can be max $\left(|\mathsf{Act}| \cdot |\mathcal{D}|^n\right)^{\mathcal{D}}$ such paths (that do not visit the same node twice). □

Bibliography

[1] Flexera, *State of the Cloud Report*, 2019. [Online]. Available: https://resources.flexera.com/web/media/documents/rightscale-2019-state-of-the-cloud-report-from-flexera.pdf

[2] S. Pearson, *Privacy, Security and Trust in Cloud Computing.* London: Springer, 2013, pp. 3–42.

[3] *OWASP Top 10 - 2021, The Ten Most Critical Web Application Security Risks*, 2021, https://owasp.org/Top10/ Last Accessed: 2-04-2022.

[4] A. Shostack, *Threat Modeling: Designing for Security*, 1st ed. Wiley Publishing, 2014.

[5] C. Newcombe, T. Rath, F. Zhang, B. Munteanu, M. Brooker, and M. Deardeuff, "How amazon web services uses formal methods," *Commun. ACM*, vol. 58, no. 4, pp. 66–73, 2015.

[6] B. Cook, "Formal reasoning about the security of amazon web services," in *Computer Aided Verification - 30th International Conference, CAV 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings, Part I*, ser. Lecture Notes in Computer Science, H. Chockler and G. Weissenbacher, Eds., vol. 10981. Springer, 2018, pp. 38–47. [Online]. Available: https://doi.org/10.1007/978-3-319-96145-3_3

[7] T. Ball, "Formal methods and tools for distributed systems @ Microsoft," 2019, https://www.microsoft.com/en-us/research/uploads/prod/2019/01/NUS2019.pdf.

[8] S. Bouchenak, G. V. Chockler, H. Chockler, G. Gheorghe, N. Santos, and A. Shraer, "Verifying cloud services: present and future," *ACM SIGOPS Oper. Syst. Rev.*, vol. 47, no. 2, pp. 6–19, 2013. [Online]. Available: https://doi.org/10.1145/2506164.2506167

[9] K. Tuma, L. Sion, R. Scandariato, and K. Yskout, "Automating the early detection of security design flaws," in *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems*, ser. MODELS '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 332–342. [Online]. Available: https://doi.org/10.1145/3365438.3410954

[10] K. Tuma, "Efficiency and automation in threat analysis of software systems," 2021.

[11] T. Binz, U. Breitenbücher, O. Kopp, and F. Leymann, *TOSCA: Portable Automated Deployment and Management of Cloud Applications*. New York, NY: Springer New York, 2014, pp. 527–549.

[12] A. Rahman, R. Mahdavi-Hezaveh, and L. A. Williams, "A systematic mapping study of infrastructure as code research," *Inf. Softw. Technol.*, vol. 108, pp. 65–77, 2019. [Online]. Available: https://doi.org/10.1016/j.infsof.2018.12.004

[13] Y. Jiang and B. Adams, "Co-evolution of infrastructure and source code - an empirical study," in *12th IEEE/ACM Working Conference on Mining Software Repositories, MSR 2015, Florence, Italy, May 16-17, 2015*, M. D. Penta, M. Pinzger, and R. Robbes, Eds. IEEE Computer Society, 2015, pp. 45–55. [Online]. Available: https://doi.org/10.1109/MSR.2015.12

[14] T. Sharma, M. Fragkoulis, and D. Spinellis, "Does your configuration code smell?" in *Proceedings of the 13th International Conference on Mining Software Repositories, MSR 2016, Austin, TX, USA, May 14-22, 2016*, M. Kim, R. Robbes, and C. Bird, Eds. ACM, 2016, pp. 189–200. [Online]. Available: https://doi.org/10.1145/2901739.2901761

[15] A. Rahman and L. A. Williams, "Different kind of smells: Security smells in infrastructure as code scripts," *IEEE Secur. Priv.*, vol. 19, no. 3, pp. 33–41, 2021. [Online]. Available: https://doi.org/10.1109/MSEC.2021.3065190

[16] A. Rahman, C. Parnin, and L. A. Williams, "The seven sins: security smells in infrastructure as code scripts," in *Proceedings of the 41st International Conference on Software Engineering, ICSE 2019, Montreal, QC, Canada, May 25-31, 2019*, J. M. Atlee, T. Bultan, and J. Whittle, Eds. IEEE / ACM, 2019, pp. 164–175. [Online]. Available: https://doi.org/10.1109/ICSE.2019.00033

[17] O. Hanappi, W. Hummer, and S. Dustdar, "Asserting reliable convergence for configuration management scripts," in *Proceedings of the 2016 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2016, part of SPLASH 2016, Amsterdam, The Netherlands, October 30 - November 4, 2016*, E. Visser and Y. Smaragdakis, Eds. ACM, 2016, pp. 328–343. [Online]. Available: https://doi.org/10.1145/2983990.2984000

[18] K. Ikeshita, F. Ishikawa, and S. Honiden, "Test suite reduction in idempotence testing of infrastructure as code," in *Tests and Proofs - 11th International Conference, TAP@STAF 2017, Marburg, Germany, July 19-20, 2017, Proceedings*, ser. Lecture Notes in Computer Science, S. Gabmeyer and E. B. Johnsen, Eds., vol. 10375. Springer, 2017, pp. 98–115. [Online]. Available: https://doi.org/10.1007/978-3-319-61467-0_6

[19] J. Lepiller, R. Piskac, M. Schäf, and M. Santolucito, "Analyzing infrastructure as code to prevent intra-update sniping vulnerabilities," in *Tools and Algorithms for the Construction and Analysis of Systems - 27th International Conference, TACAS 2021, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2021, Luxembourg City, Luxembourg, March 27 - April 1, 2021, Proceedings, Part II*, ser. Lecture Notes in Computer Science, J. F. Groote and K. G. Larsen, Eds., vol. 12652. Springer, 2021, pp. 105–123. [Online]. Available: https://doi.org/10.1007/978-3-030-72013-1_6

[20] F. Baader, I. Horrocks, C. Lutz, and U. Sattler, *An Introduction to Description Logic*. Cambridge University Press, 2017. [Online]. Available: http://www.cambridge.org/de/academic/subjects/computer-science/knowledge-management-databases-and-data-mining/introduction-description-logic?format=PB#17zVGeWD2TZUeu6s.97

[21] F. Baader, I. Horrocks, and U. Sattler, "Description logics," in *Handbook on Ontologies*, ser. International Handbooks on Information Systems, S. Staab and R. Studer, Eds. Springer, 2009, pp. 21–43. [Online]. Available: https://doi.org/10.1007/978-3-540-92673-3_1

[22] S. Grimm and B. Motik, "Closed world reasoning in the semantic web through epistemic operators," in *Proceedings of the OWLED*05 Workshop on OWL: Experiences and Directions, Galway, Ireland, November 11-12, 2005*, ser. CEUR Workshop Proceedings, B. C. Grau, I. Horrocks, B. Parsia, and P. F. Patel-Schneider, Eds., vol. 188. CEUR-WS.org, 2005. [Online]. Available: http://ceur-ws.org/Vol-188/sub12.pdf

[23] C. Lutz, I. Seylan, and F. Wolter, "Mixing open and closed world assumptionin ontology-based data access: Non-uniform data complexity," in *Proceedings of the 2012 International Workshop on Description Logics, DL-2012, Rome, Italy, June 7-10, 2012*, ser. CEUR Workshop Proceedings, Y. Kazakov, D. Lembo, and F. Wolter, Eds., vol. 846. CEUR-WS.org, 2012. [Online]. Available: http://ceur-ws.org/Vol-846/paper_17.pdf

[24] D. M. Gabbay, "Theoretical foundations for non-monotonic reasoning in expert systems," in *Logics and Models of Concurrent Systems*, K. R. Apt, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1985, pp. 439–457.

[25] D. McDermott and J. Doyle, "Non-monotonic logic i," *Artificial Intelligence*, vol. 13, no. 1, pp. 41–72, 1980, special Issue on Non-Monotonic Logic. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0004370280900120

[26] I. Horrocks, O. Kutz, and U. Sattler, "The even more irresistible SROIQ," in *Proceedings, Tenth International Conference on Principles of Knowledge Representation and Reasoning, Lake District of the United Kingdom, June 2-5, 2006*, P. Doherty, J. Mylopoulos, and C. A. Welty, Eds. AAAI Press, 2006, pp. 57–67. [Online]. Available: http://www.aaai.org/Library/KR/2006/kr06-009.php

[27] M. Krötzsch, F. Simancik, and I. Horrocks, "A description logic primer," *CoRR*, vol. abs/1201.4089, 2012. [Online]. Available: http://arxiv.org/abs/1201.4089

[28] F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider, Eds., *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.

[29] F. Baader, S. Brandt, and C. Lutz, "Pushing the el envelope," in *Proceedings of the 19th International Joint Conference on Artificial Intelligence*, ser. IJCAI'05. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2005, p. 364–369.

[30] A. Artale, D. Calvanese, R. Kontchakov, and M. Zakharyaschev, "The dl-lite family and relations," *J. Artif. Intell. Res.*, vol. 36, pp. 1–69, 2009. [Online]. Available: https://doi.org/10.1613/jair.2820

[31] D. Calvanese, G. D. Giacomo, D. Lembo, M. Lenzerini, and R. Rosati, "Tractable reasoning and efficient query answering in description logics: The *DL-Lite* family," *J. Autom. Reason.*, vol. 39, no. 3, pp. 385–429, 2007. [Online]. Available: https://doi.org/10.1007/s10817-007-9078-x

[32] E. Franconi, Y. A. Ibáñez-García, and I. Seylan, "Query answering with dboxes is hard," *Electron. Notes Theor. Comput. Sci.*, vol. 278, pp. 71–84, 2011. [Online]. Available: https://doi.org/10.1016/j.entcs.2011.10.007

[33] F. M. Donini, M. Lenzerini, D. Nardi, A. Schaerf, and W. Nutt, "Adding epistemic operators to concept languages," in *Proceedings of the 3rd International Conference on Principles of Knowledge Representation and Reasoning (KR'92). Cambridge, MA, USA, October 25-29, 1992*, B. Nebel, C. Rich, and W. R. Swartout, Eds. Morgan Kaufmann, 1992, pp. 342–353.

[34] F. M. Donini, M. Lenzerini, D. Nardi, W. Nutt, and A. Schaerf, "An epistemic operator for description logics," *Artif. Intell.*, vol. 100, no. 1-2, pp. 225–274, 1998. [Online]. Available: https://doi.org/10.1016/S0004-3702(98)00009-5

[35] A. Borgida, "On the relative expressiveness of description logics and predicate logics," *Artif. Intell.*, vol. 82, no. 1-2, pp. 353–367, 1996. [Online]. Available: https://doi.org/10.1016/0004-3702(96)00004-5

[36] K. Schild, "A correspondence theory for terminological logics: Preliminary report," in *Proceedings of the 12th International Joint Conference on Artificial Intelligence. Sydney, Australia, August 24-30, 1991*, J. Mylopoulos and R. Reiter, Eds. Morgan Kaufmann, 1991, pp. 466–471. [Online]. Available: http://ijcai.org/Proceedings/91-1/Papers/072.pdf

[37] F. Baader, "Augmenting concept languages by transitive closure of roles: An alternative to terminological cycles," in *Proceedings of the 12th International Joint Conference on Artificial Intelligence - Volume 1*, ser. IJCAI'91. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1991, p. 446–451.

[38] U. Hustadt, R. A. Schmidt, and L. Georgieva, "A survey of decidable first-order fragments and description logics," *Journal of Relational Methods in Computer Science*, vol. 1, no. 251-276, p. 3, 2004.

[39] S. Ceri, G. Gottlob, and L. Tanca, "What you always wanted to know about datalog (and never dared to ask)," *IEEE Trans. Knowl. Data Eng.*, vol. 1, no. 1, pp. 146–166, 1989. [Online]. Available: https://doi.org/10.1109/69.43410

[40] M. Krötzsch, S. Rudolph, and P. H. Schmitt, "A closer look at the semantic relationship between datalog and description logics," *Semantic Web*, vol. 6, no. 1, pp. 63–79, 2015. [Online]. Available: https://doi.org/10.3233/SW-130126

[41] D. Jackson, "Alloy: a language and tool for exploring software designs," *Commun. ACM*, vol. 62, no. 9, pp. 66–76, 2019. [Online]. Available: https://doi.org/10.1145/3338843

[42] A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella, "Nusmv 2: An opensource tool for symbolic model checking," in *Computer Aided Verification*, E. Brinksma and K. G. Larsen, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 359–364.

[43] R. Cavada, A. Cimatti, M. Dorigatti, A. Griggio, A. Mariotti, A. Micheli, S. Mover, M. Roveri, and S. Tonetta, "The nuxmv symbolic model checker," in *CAV*, 2014, pp. 334–342.

[44] G. Bruns and P. Godefroid, "Model checking partial state spaces with 3-valued temporal logics," in *Computer Aided Verification*, N. Halbwachs and D. Peled, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 274–287.

[45] P. Godefroid, M. Huth, and R. Jagadeesan, "Abstraction-based model checking using modal transition systems," in *CONCUR 2001 — Concurrency Theory*, K. G. Larsen and M. Nielsen, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 426–440.

[46] M. Huth, R. Jagadeesan, and D. Schmidt, "Modal transition systems: A foundation for three-valued program analysis," in *Programming Languages and Systems*, D. Sands, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 155–169.

[47] K. Morris, *Infrastructure as code: managing servers in the cloud.* " O'Reilly Media, Inc.", 2016.

[48] "Terraform," 2021, https://www.terraform.io/, Last accessed on 2021-01-28.

[49] "Microsoft Azure Resource Manager," 2020, https://azure.microsoft.com/en-us/features/resource-manager/, Last accessed on 2021-01-28.

[50] "Google Deployment Manager," 2021, https://cloud.google.com/deployment-manager, Last accessed on 2021-01-28.

[51] "CloudFORMAL: Prototype Implementation," 2021, http://github.com/claudiacauli/CloudFORMAL, Last accessed on 2020-10-15.

[52] I. Horrocks, P. F. Patel-Schneider, and F. van Harmelen, "From SHIQ and RDF to OWL: the making of a web ontology language," *J. Web Semant.*, vol. 1, no. 1, pp. 7–26, 2003. [Online]. Available: https://doi.org/10.1016/j.websem.2003.07.001

[53] B. C. Grau, I. Horrocks, B. Motik, B. Parsia, P. F. Patel-Schneider, and U. Sattler, "OWL 2: The next step for OWL," *J. Web Semant.*, vol. 6, no. 4, pp. 309–322, 2008. [Online]. Available: https://doi.org/10.1016/j.websem.2008.05.001

[54] P. Patel-Schneider, B. C. Grau, and B. Motik, "OWL 2 web ontology language direct semantics (second edition)," W3C, W3C Recommendation, December 2012, http://www.w3.org/TR/2012/REC-owl2-direct-semantics-20121211/.

[55] M. A. Musen, "The protégé project: a look back and a look forward," *AI Matters*, vol. 1, no. 4, pp. 4–12, 2015. [Online]. Available: https://doi.org/10.1145/2757001.2757003

[56] D. Tsarkov and I. Horrocks, "Fact++ description logic reasoner: System description," in *Automated Reasoning, Third International Joint Conference, IJCAR 2006, Seattle, WA, USA, August 17-20, 2006, Proceedings*, ser. Lecture Notes in Computer Science, U. Furbach and N. Shankar, Eds., vol. 4130. Springer, 2006, pp. 292–297. [Online]. Available: https://doi.org/10.1007/11814771_26

[57] B. Glimm, I. Horrocks, B. Motik, G. Stoilos, and Z. Wang, "Hermit: An OWL 2 reasoner," *J. Autom. Reason.*, vol. 53, no. 3, pp. 245–269, 2014. [Online]. Available: https://doi.org/10.1007/s10817-014-9305-1

[58] E. Sirin, B. Parsia, B. C. Grau, A. Kalyanpur, and Y. Katz, "Pellet: A practical OWL-DL reasoner," *J. Web Semant.*, vol. 5, no. 2, pp. 51–53, 2007. [Online]. Available: https://doi.org/10.1016/j.websem.2007.03.004

[59] M. Schmidt-Schauß and G. Smolka, "Attributive concept descriptions with complements," *Artif. Intell.*, vol. 48, no. 1, pp. 1–26, 1991. [Online]. Available: https://doi.org/10.1016/0004-3702(91)90078-X

[60] F. Baader, I. Horrocks, and U. Sattler, "Description logics," in *Handbook of Knowledge Representation*, ser. Foundations of Artificial Intelligence, F. van Harmelen, V. Lifschitz, and B. W. Porter, Eds. Elsevier, 2008, vol. 3, pp. 135–179. [Online]. Available: https://doi.org/10.1016/S1574-6526(07)03003-9

[61] "Resource Specification," 2020, https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/cfn-resource-specification.html, Last accessed on 2020-08-13.

[62] S. Challita, "Inferring models from cloud apis and reasoning over them: A tooled and formal approach. (inférer des modèles à partir d'apis

cloud et raisonner dessus: une approche outillée et formelle)," Ph.D. dissertation, Lille University of Science and Technology, France, 2018. [Online]. Available: https://tel.archives-ouvertes.fr/tel-02016442

[63] M. Horridge and S. Bechhofer, "The OWL API: A java API for OWL ontologies," *Semantic Web*, vol. 2, no. 1, pp. 11–21, 2011. [Online]. Available: https://doi.org/10.3233/SW-2011-0025

[64] "Multi-Cloud Security Auditing Tool," 2020, http://github.com/nccgroup/ScoutSuite, Last accessed on 2020-08-04.

[65] J. Backes, S. Bayless, B. Cook, C. Dodge, A. Gacek, A. J. Hu, T. Kahsai, B. Kocik, E. Kotelnikov, J. Kukovec, S. McLaughlin, J. Reed, N. Rungta, J. Sizemore, M. A. Stalzer, P. Srinivasan, P. Subotic, C. Varming, and B. Whaley, "Reachability analysis for aws-based networks," in *Computer Aided Verification - 31st International Conference, CAV 2019, New York City, NY, USA, July 15-18, 2019, Proceedings, Part II*, ser. Lecture Notes in Computer Science, I. Dillig and S. Tasiran, Eds., vol. 11562. Springer, 2019, pp. 231–241. [Online]. Available: https://doi.org/10.1007/978-3-030-25543-5_14

[66] J. Backes, P. Bolignano, B. Cook, C. Dodge, A. Gacek, K. S. Luckow, N. Rungta, O. Tkachuk, and C. Varming, "Semantic-based automated reasoning for AWS access policies using SMT," in *2018 Formal Methods in Computer Aided Design, FMCAD 2018, Austin, TX, USA, October 30 - November 2, 2018*, N. Bjørner and A. Gurfinkel, Eds. IEEE, 2018, pp. 1–9. [Online]. Available: https://doi.org/10.23919/FMCAD.2018.8602994

[67] "Infrastructure Security, Compliance, and Governance," 2020, http://www.cloudconformity.com/, Last accessed on 2020-08-04.

[68] "The CFnNag Linting Tool," 2020, https://github.com/stelligent/cfn_nag, Last accessed on 2020-10-15.

[69] "The AWS CloudFormation Linter," 2020, https://github.com/aws-cloudformation/cfn-python-lint, Last accessed on 2020-10-15.

[70] "Static Analysis Security Scanner for Terraform," 2020, https://tfsec.dev/, Last accessed on 2021-05-10.

[71] D. L. McGuinness, L. A. Resnick, and C. L. I. Jr., "Description logic in practice: A CLASSIC application," in *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence, IJCAI 95, Montréal Québec, Canada, August 20-25 1995, 2 Volumes*. Morgan Kaufmann, 1995, pp. 2045–2046. [Online]. Available: http://ijcai.org/Proceedings/95-2/Papers/132.pdf

[72] D. L. McGuinness and J. R. Wright, "Conceptual modelling for configuration: A description logic-based approach," *Artif. Intell. Eng. Des. Anal. Manuf.*, vol. 12, no. 4, pp. 333–344, 1998. [Online]. Available: http://journals.cambridge.org/action/displayAbstract?aid=38647

[73] "OWASP Ontology-driven Threat Modeling," 2021, https://github.com/OWASP/OdTM, Last accessed on 2021-05-17.

[74] G. Bruns and P. Godefroid, "Model checking partial state spaces with 3-valued temporal logics," in *Computer Aided Verification, 11th International Conference, CAV '99, Trento, Italy, July 6-10, 1999, Proceedings*, ser. Lecture Notes in Computer Science, N. Halbwachs and D. A. Peled, Eds., vol. 1633. Springer, 1999, pp. 274–287. [Online]. Available: https://doi.org/10.1007/3-540-48683-6_25

[75] ——, "Model checking with multi-valued logics," in *Automata, Languages and Programming: 31st International Colloquium, ICALP 2004, Turku, Finland, July 12-16, 2004. Proceedings*, ser. Lecture Notes in Computer Science, J. Díaz, J. Karhumäki, A. Lepistö, and D. Sannella, Eds., vol. 3142. Springer, 2004, pp. 281–293. [Online]. Available: https://doi.org/10.1007/978-3-540-27836-8_26

[76] O. Kupferman and O. Grumberg, "Buy one, get one free!!!" *J. Log. Comput.*, vol. 6, no. 4, pp. 523–539, 1996. [Online]. Available: https://doi.org/10.1093/logcom/6.4.523

[77] U. Sattler and M. Y. Vardi, "The hybrid μ-calculus," in *Automated Reasoning, First International Joint Conference, IJCAR 2001, Siena, Italy, June 18-23, 2001, Proceedings*, ser. Lecture Notes in Computer Science, R. Goré, A. Leitsch, and T. Nipkow, Eds., vol. 2083. Springer, 2001, pp. 76–91. [Online]. Available: https://doi.org/10.1007/3-540-45744-5_7

[78] N. D'Ippolito, D. Fischbein, M. Chechik, and S. Uchitel, "MTSA: the modal transition system analyser," in *23rd IEEE/ACM International Conference on Automated Software Engineering (ASE 2008), 15-19 September 2008, L'Aquila, Italy.* IEEE Computer Society, 2008, pp. 475–476. [Online]. Available: https://doi.org/10.1109/ASE.2008.78

[79] A. Gurfinkel, O. Wei, and M. Chechik, "Yasm: A software model-checker for verification and refutation," in *Computer Aided Verification, 18th International Conference, CAV 2006, Seattle, WA, USA, August 17-20, 2006, Proceedings*, ser. Lecture Notes in Computer Science, T. Ball and R. B. Jones, Eds., vol. 4144. Springer, 2006, pp. 170–174. [Online]. Available: https://doi.org/10.1007/11817963_18

[80] D. Calvanese, G. D. Giacomo, D. Lembo, M. Lenzerini, and R. Rosati, "Eql-lite: Effective first-order query processing in description logics," in *IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, January 6-12, 2007*, M. M. Veloso, Ed., 2007, pp. 274–279. [Online]. Available: http://ijcai.org/Proceedings/07/Papers/042.pdf

[81] C. Corona, E. D. Pasquale, A. Poggi, M. Ruzzi, and D. F. Savo, "When dl-lite met OWL..." in *Proceedings of the Fifth OWLED Workshop on OWL: Experiences and Directions, collocated with the 7th International Semantic Web Conference (ISWC-2008), Karlsruhe, Germany, October 26-27, 2008*, ser. CEUR Workshop Proceedings, C. Dolbear, A. Ruttenberg, and U. Sattler, Eds., vol. 432. CEUR-WS.org, 2008. [Online]. Available: http://ceur-ws.org/Vol-432/owled2008eu_submission_35.pdf

[82] C. Cauli, M. Li, N. Piterman, and O. Tkachuk, "Pre-deployment security assessment for cloud services through semantic reasoning," in *Computer Aided Verification - 33rd International Conference, CAV 2021, Virtual Event, July 20-23, 2021, Proceedings, Part I*, ser. Lecture Notes in Computer Science, A. Silva and K. R. M. Leino, Eds., vol. 12759. Springer, 2021, pp. 767–780. [Online]. Available: https://doi.org/10.1007/978-3-030-81685-8_36

[83] S. Tobies, "A nexptime-complete description logic strictly contained in c^2," in *Computer Science Logic, 13th International Workshop, CSL '99, 8th Annual Conference of the EACSL, Madrid, Spain, September 20-25, 1999, Proceedings*, ser. Lecture Notes in Computer Science, J. Flum and M. Rodríguez-Artalejo, Eds., vol. 1683. Springer, 1999, pp. 292–306. [Online]. Available: https://doi.org/10.1007/3-540-48168-0_21

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

[85] M. Y. Vardi, "The complexity of relational query languages (extended abstract)," in *Proceedings of the 14th Annual ACM Symposium on Theory of Computing, May 5-7, 1982, San Francisco, California, USA*, H. R. Lewis, B. B. Simons, W. A. Burkhard, and L. H. Landweber, Eds. ACM, 1982, pp. 137–146. [Online]. Available: https://doi.org/10.1145/800070.802186

[86] F. Baader and B. Hollunder, "Embedding defaults into terminological knowledge representation formalisms," *J. Autom. Reason.*, vol. 14, no. 1, pp. 149–180, 1995. [Online]. Available: https://doi.org/10.1007/BF00883932

[87] S. Borgwardt and W. Forkel, "Closed-world semantics for conjunctive queries with negation over elh-bottom ontologies," in *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*, S. Kraus, Ed. ijcai.org, 2019, pp. 6131–6135. [Online]. Available: https://doi.org/10.24963/ijcai.2019/849

[88] E. Franconi, Y. A. Ibáñez-García, and I. Seylan, "Query answering with dboxes is hard," *Electron. Notes Theor. Comput. Sci.*, vol. 278, pp. 71–84, 2011. [Online]. Available: https://doi.org/10.1016/j.entcs.2011.10.007

[89] S. A. Gaggl, S. Rudolph, and L. Schweizer, "Fixed-domain reasoning for description logics," in *Proc. of the 22nd Eur. Conf. on Artificial Intelligence (ECAI 2016)*, ser. Frontiers in Artificial Intelligence and Applications, G. A. Kaminka, M. Fox, P. Bouquet, E. Hüllermeier, V. Dignum, F. Dignum, and F. van Harmelen, Eds., vol. 285. IOS Press, 2016, pp. 819–827. [Online]. Available: https://doi.org/10.3233/978-1-61499-672-9-819

[90] N. Ngo, M. Ortiz, and M. Simkus, "Closed predicates in description logics: Results on combined complexity," in *Principles of Knowledge Representation and Reasoning: Proceedings of the Fifteenth International Conference, KR 2016, Cape Town, South Africa, April 25-29, 2016*, C. Baral, J. P. Delgrande, and F. Wolter, Eds. AAAI Press, 2016, pp. 237–246. [Online]. Available: http://www.aaai.org/ocs/index.php/KR/KR16/paper/view/12906

[91] C. Lutz, I. Seylan, and F. Wolter, "The data complexity of ontology-mediated queries with closed predicates," *Log. Methods Comput. Sci.*, vol. 15, no. 3, 2019. [Online]. Available: https://doi.org/10.23638/LMCS-15(3:23)2019

[92] ——, "Ontology-based data access with closed predicates is inherently intractable(sometimes)," in *IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, August 3-9, 2013*, F. Rossi, Ed. IJCAI/AAAI, 2013, pp. 1024–1030. [Online]. Available: http://www.aaai.org/ocs/index.php/IJCAI/IJCAI13/paper/view/6870

[93] A. Hendre and K. P. Joshi, "A semantic approach to cloud security and compliance," in *8th IEEE International Conference on Cloud Computing, CLOUD 2015, New York City, NY, USA, June 27 - July 2, 2015*, C. Pu and A. Mohindra, Eds. IEEE Computer Society, 2015, pp. 1081–1084. [Online]. Available: https://doi.org/10.1109/CLOUD.2015.157

[94] R. Reiter, "What should a database know?" *J. Log. Program.*, vol. 14, no. 1&2, pp. 127–153, 1992. [Online]. Available: https://doi.org/10.1016/0743-1066(92)90049-9

[95] M. Bouchet, B. Cook, B. Cutler, A. Druzkina, A. Gacek, L. Hadarean, R. Jhala, B. Marshall, D. Peebles, N. Rungta, C. Schlesinger, C. Stephens, C. Varming, and A. Warfield, "Block public access: trust safety verification of access control policies," in *ESEC/FSE '20: 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Virtual Event, USA, November 8-13, 2020*, P. Devanbu, M. B. Cohen, and T. Zimmermann, Eds. ACM, 2020, pp. 281–291. [Online]. Available: https://doi.org/10.1145/3368089.3409728

[96] J. Backes, U. Berrueco, T. Bray, D. Brim, B. Cook, A. Gacek, R. Jhala, K. S. Luckow, S. McLaughlin, M. Menon, D. Peebles, U. Pugalia, N. Rungta, C. Schlesinger, A. Schodde, A. Tanuku, C. Varming, and D. Viswanathan, "Stratified abstraction of access control policies," in *Computer Aided Verification - 32nd International Conference, CAV 2020, Los Angeles, CA, USA, July 21-24, 2020, Proceedings, Part I*, ser. Lecture Notes in Computer Science, S. K. Lahiri and C. Wang, Eds., vol. 12224. Springer, 2020, pp. 165–176. [Online]. Available: https://doi.org/10.1007/978-3-030-53288-8_9

[97] C. Cauli, M. Ortiz, and N. Piterman, "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.

[98] K. Erol, D. S. Nau, and V. S. Subrahmanian, "Complexity, decidability and undecidability results for domain-independent planning," *Artif. Intell.*, vol. 76, no. 1-2, pp. 75–88, 1995. [Online]. Available: https://doi.org/10.1016/0004-3702(94)00080-K

[99] D. Chapman, "Planning for conjunctive goals," *Artif. Intell.*, vol. 32, no. 3, pp. 333–377, 1987. [Online]. Available: https://doi.org/10.1016/0004-3702(87)90092-0

[100] S. Ahmetaj, D. Calvanese, M. Ortiz, and M. Simkus, "Managing change in graph-structured data using description logics," *ACM Trans. Comput. Log.*, vol. 18, no. 4, pp. 27:1–27:35, 2017. [Online]. Available: https://doi.org/10.1145/3143803

[101] D. Calvanese, M. Montali, F. Patrizi, and G. D. Giacomo, "Description logic based dynamic systems: Modeling, verification, and synthesis," in *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, Q. Yang and M. J. Wooldridge, Eds. AAAI Press, 2015, pp. 4247–4253. [Online]. Available: http://ijcai.org/Abstract/15/604

[102] B. B. Hariri, D. Calvanese, M. Montali, G. D. Giacomo, R. D. Masellis, and P. Felli, "Description logic knowledge and action bases," *J. Artif. Intell. Res.*, vol. 46, pp. 651–686, 2013.

[103] D. Calvanese, M. Ortiz, and M. Simkus, "Evolving graph databases under description logic constraints," in *Informal Proceedings of the 26th International Workshop on Description Logics, Ulm, Germany, July 23 - 26, 2013*, ser. CEUR Workshop Proceedings, T. Eiter, B. Glimm, Y. Kazakov, and M. Krötzsch, Eds., vol. 1014. CEUR-WS.org, 2013, pp. 120–131. [Online]. Available: http://ceur-ws.org/Vol-1014/paper_82.pdf

[104] ——, "Verification of evolving graph-structured data under expressive path constraints," in *19th International Conference on Database Theory, ICDT 2016, Bordeaux, France, March 15-18, 2016*, ser. LIPIcs, W. Martens and T. Zeume, Eds., vol. 48. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016, pp. 15:1–15:19. [Online]. Available: https://doi.org/10.4230/LIPIcs.ICDT.2016.15

[105] E. Kharlamov, D. Zheleznyakov, and D. Calvanese, "Capturing model-based ontology evolution at the instance level: The case of dl-lite," *J. Comput. Syst. Sci.*, vol. 79, no. 6, pp. 835–872, 2013. [Online]. Available: https://doi.org/10.1016/j.jcss.2013.01.006

[106] H. Liu, C. Lutz, M. Milicic, and F. Wolter, "Foundations of instance level updates in expressive description logics," *Artif. Intell.*, vol. 175, no. 18, pp. 2170–2197, 2011. [Online]. Available: https://doi.org/10.1016/j.artint.2011.08.003

[107] G. D. Giacomo, R. D. Masellis, and R. Rosati, "Verification of conjunctive artifact-centric services," *Int. J. Cooperative Inf. Syst.*, vol. 21, no. 2, pp. 111–140, 2012. [Online]. Available: https://doi.org/10.1142/S0218843012500025

[108] D. Calvanese, M. Montali, F. Patrizi, and M. Stawowy, "Plan synthesis for knowledge and action bases," in *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016*, S. Kambhampati, Ed. IJCAI/AAAI Press, 2016, pp. 1022–1029. [Online]. Available: http://www.ijcai.org/Abstract/16/149

[109] D. Calvanese, G. D. Giacomo, M. Montali, and F. Patrizi, "Verification and synthesis in description logic based dynamic systems," in *Web Reasoning and Rule Systems - 7th International Conference, RR 2013, Mannheim, Germany, July 27-29, 2013. Proceedings*, ser. Lecture Notes in Computer Science, W. Faber and D. Lembo, Eds., vol. 7994. Springer, 2013, pp. 50–64. [Online]. Available: https://doi.org/10.1007/978-3-642-39666-3_5

[110] M. Milicic, "Planning in action formalisms based on dls: First results," in *Proceedings of the 2007 International Workshop on Description Logics (DL2007), Brixen-Bressanone, near Bozen-Bolzano, Italy, 8-10 June, 2007*, ser. CEUR Workshop Proceedings, D. Calvanese, E. Franconi, V. Haarslev, D. Lembo, B. Motik, A. Turhan, and S. Tessaris, Eds., vol. 250. CEUR-WS.org, 2007. [Online]. Available: http://ceur-ws.org/Vol-250/paper_59.pdf

[111] E. P. D. Pednault, "ADL and the State-Transition Model of Action," *Journal of Logic and Computation*, vol. 4, no. 5, pp. 467–512, 10 1994. [Online]. Available: https://doi.org/10.1093/logcom/4.5.467