Creating Views of Ontologies with Applications to Ontology-Based Query Answering

Yue Xiang[†‡], Wenxing Deng[†], Chang Lu[†§], Peiqi Wei[†¶], Yizheng Zhao[†*], Hao Feng[‖]
[†]National Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, China
[‡]Faculty of Information Technology, Beijing University of Technology, Beijing, China
[§]School of Physics, Nanjing University, Nanjing, China
[¶]College of Mechanical and Electrical Engineering, Beijing University of Chemical Technology, Beijing, China
[‖]Meituan-Dianping Group, Beijing, China

***Abstract*—This paper tackles the problem of computing views of ontologies using a forgetting-based approach. In relational databases, a view is a subset of a database, whereas in ontologies, a view is more than a subset — it contains not only axioms that are contained in the original ontology, but also newly-derived axioms that are entailed by the original ontology (implicitly contained in the original ontology). Specifically, given an ontology $\mathcal{O}$, the *signature* $\mathbf{sig}(\mathcal{O})$ of $\mathcal{O}$ is the set of all the terms in $\mathcal{O}$, and a *view* $\mathcal{V}$ of $\mathcal{O}$ is a new ontology obtained from $\mathcal{O}$ using only part of $\mathcal{O}$'s signature, namely the *target signature*, while keeping the original meanings over the terms in the target signature. Computing views of ontologies is useful for many ontology-based knowledge processing tasks such as (i) ontology-based query answering, in the sense that the view can be used as a substitute of the original ontology to answer queries formulated using only the target signature, and (ii) data protection, in the sense that it restricts users from viewing certain information of an ontology.**

Forgetting is a *Knowledge Representation and Reasoning* technique; in particular, forgetting is a form of non-standard reasoning concerned with eliminating from an ontology a subset of its signature, namely the *forgetting signature*, in such a way that all logical consequences are preserved up to the remaining signature. Forgetting can therefore be used as an effective means for computing views of ontologies — the solution of forgetting a sub-signature $\mathcal{F} \subseteq \mathbf{sig}(\mathcal{O})$ (the forgetting signature) from an ontology $\mathcal{O}$ is the view $\mathcal{V}$ of $\mathcal{O}$ for the target signature $\mathbf{sig}(\mathcal{O})\backslash\mathcal{F}$.

In this paper, we present a novel, practical forgetting method for computing views of ontologies. The method is terminating and sound, and allows a set of terms to be forgotten from ontologies expressible in the description logic $\mathcal{ALCOIH}$, which is the basic $\mathcal{ALC}$ extended with nominals ($\mathcal{O}$), inverse roles ($\mathcal{I}$), and role inclusions ($\mathcal{H}$), and is one of the most expressive description logics that has frequently been used for modelling domain knowledge but has never been considered as source/target language for forgetting. Despite the inherent difficulty of forgetting for this level of expressivity, an evaluation with a prototype of the method on a corpus of real-world ontologies has shown very good success rates and performance results. A case study on the ontology-based query answering problem has further verified the practicality of the forgetting method in real-world application scenarios.

I. Introduction

A. Basics of Ontologies

In the context of Information Science and Artificial Intelligence, an *ontology* defines a set of representational primitives with which to model a domain of discourse. These representational primitives include classes (*concepts*), class members (*individuals*), and their properties (*attributes* of concepts and individuals or *relationships* to other concepts and individuals). The definitions of the representational primitives include information about their meanings and constraints on their logically consistent application. In the context of database systems, an ontology can be viewed as a level of abstraction of data models, analogous to hierarchical and relational models, but intended for modeling knowledge about concepts, individuals, and their properties.

Ontologies are typically formulated in languages that allow abstraction away from data structures and implementation strategies; in practice, the languages of ontologies are closer in expressive power to first-order logic than languages used to model databases. For this reason, ontologies are said to be at the "semantic" level, whereas database schema are models of data at the "logical" or "physical" level. Due to their independence from lower level data models, ontologies are used for integrating heterogeneous databases, enabling interoperability among disparate systems, and specifying interfaces to independent, knowledge-based services.

In particular, modern ontologies are formulated in the Web Ontology Language (OWL) that has a formal semantics based mainly on description logics (DLs) [3]; OWL is now the most prevailing textual language for developing ontologies. Using a logic-based, well-structured language such as DLs has two notable advantages: (i) they have unambiguous semantics, which means that the meaning of terms is specified in an unambiguous way, thereby enabling shared understanding of domain knowledge – the terms must have a standard interpretation among all users, and (ii) one can make use of the reasoning services of DL reasoners for ontology-based knowledge processing and related tasks.

Given an application domain, we use *concept names*, *role names*, and *individual names*, the three basic building blocks of DLs, and a set of standard *logical constructors* to describe knowledge from the application domain. For different types of knowledge, DLs build concepts in four different ways:

i. As in an encyclopedia, DLs describe the meaning of a concept name in terms of a concept expression. For example, we can describe the meaning of Footballer and

Yizheng Zhao* is the corresponding author. This paper was supported by the Fundamental Research Funds for the Central Universities (No. 14380006).

Olympian using the following DL expressions:

$$\mathsf{Footballer} \equiv \mathsf{Person} \sqcap \exists \mathsf{plays}.\mathsf{Football}$$
$$\mathsf{Olympian} \equiv \mathsf{Person} \sqcap \exists \mathsf{participatesIn}.\mathsf{OlympicGames}$$

Intuitively, the first expression says that footballers are persons who play football and the second expression says that Olympians are persons who participate in Olympic Games.

ii. DLs capture background knowledge. For example, we can state that a footballer is also a sportsperson, and that a person has sex either male or female, using the following DL expressions:

$$\mathsf{Footballer} \sqsubseteq \mathsf{Sportsperson}$$
$$\mathsf{Person} \sqsubseteq \exists \mathsf{hasSex}.(\mathsf{Male} \sqcup \mathsf{Female})$$

iii. DLs assert that individual names stand for instances of concepts. For example, we can assert that COMP60332 stands for an instance of Course, and Phelps stands for an instance of Person who participates in OlympicGames, using the following DL expressions:

$$\{\mathsf{COMP60332}\} \sqsubseteq \mathsf{Course}$$
$$\{\mathsf{Phelps}\} \sqsubseteq \mathsf{Person} \sqcap \exists \mathsf{participatesIn}.\mathsf{OlympicGames}$$

iv. DLs relate individuals by roles. For example, we can say that John is a teacher who teaches the course COMP60332 using the following expression:

$$\{\mathsf{John}\} \sqsubseteq \exists \mathsf{teaches}.\{\mathsf{COMP60332}\}.$$

DLs separate domain knowledge into two parts: a terminological part, namely, the *TBox*, and an assertional part, namely, the *ABox*. The TBox contains a set of statements of the forms as shown in (i) and (ii). The ABox contains a set of statements of the forms as shown in (iii) and (iv). Together, TBox and ABox statements make up a *knowledge base*. One can think of a knowledge base as a relational database, where the TBox statements correspond to the schema of the database because it expresses general constraints on what the world looks like, and the ABox statements correspond to the data of the database because it talks about concrete elements. In ontologies, TBox statements are called *TBox axioms* and ABox statements are called *ABox assertions*. In the remainder of this paper, the terms *ontology* and *knowledge base* are used interchangeably.

DLs have many variants which differ with each other in expressivity, depending on which building blocks and logical constructors are set to be used to describe domain knowledge. The basic DL is called $\mathcal{ALC}$, which uses concept names, role names, and the logical constructors of $\neg$, $\sqcap$, $\sqcup$, $\exists$, and $\forall$ to build concept descriptions. Extensions of $\mathcal{ALC}$ are indicated by adding a corresponding letter in its name. For example, $\mathcal{ALCO}$ denotes the extension of $\mathcal{ALC}$ by nominals. $\mathcal{ALCO}$ is more expressive than $\mathcal{ALC}$ in the sense that there is an $\mathcal{ALCO}$ concept C such that $C \not\equiv D$ holds for all $\mathcal{ALC}$ concepts D [3]. Hence, additional expressivity brings more power and flexibility for making statements about domain knowledge. However, on the other hand, such power and flexibility come with a computational cost, and one of the most important topics of DL research has been exploring the trade-off between the expressive power of the language available for making statements and the computational complexity of various reasoning tasks for this language. The expressive power of DLs is invariably constrained so as to at least ensure that standard reasoning (satisfiability testing) is decidable, i.e., reasoning can always be correctly completed within a finite amount of time. It is known that reasoning in $\mathcal{ALCOIH}$ (the language considered in this paper) is decidable [17].

B. Why Computing Views of Ontologies?

With the growing utilization of ontologies in real-world applications, not only has the number of available ontologies increased considerably, but also they are often large in size, complex in structure, and thus are becoming more difficult to manage. Moreover, capturing domain knowledge in the form of ontologies is labor-intensive work from the engineering perspective. There is therefore a strong demand for techniques and tools for re-engineering with ontologies, so that existing ontologies can be reused to their full potential — new ontologies can be generated from existing ones, and are not necessarily scratch-developed; the latter is costly, time-consuming and error-prone. Creating views of ontologies is one of such ontology re-engineering operations that seeks to generate new ontologies from existing ones. A *view* $\mathcal{V}$ of an ontology $\mathcal{O}$ is a new ontology obtained from $\mathcal{O}$ using only part of $\mathcal{O}$'s signature, namely the *target signature*, while preserving the original meanings of the terms in the target signature. Computing views of ontologies is useful for many ontology engineering and related tasks. These include, but are not limited to the following one.

i. **Ontology Summary**: Creating a summary of an ontology is helpful when an ontology engineer wants to gain a quick understanding of the ontology, inexpensively examining its content (to decide whether the ontology is suitable for use). The summary may ignore more specific information and concentrate on more general terms that are expressed using concept and role names of high level.

ii. **Ontology Reuse**: Knowledge modelled in large ontologies is often rich, heterogeneous, and multi-topic related, while applications are interested in or relevant to specific topics. Compared to using existing ontologies or building new ontologies from scratch, extracting fragments relative to specific topics from existing ontologies and reusing them in a specialized context is simpler, cheaper, and thus more interesting to ontology engineers.

iii. **Ontology-Based Query Answering (OBQA)**: Taking ontological knowledge into account when retrieving data from relational databases has been widely acknowledged. It is however found in many scenarios [7] that querying a large knowledge base often involves extensive reasoning, which, due to the high computational complexity of reasoning in DLs, can be expensive both in terms of time and space. Instead, querying a view of the knowledge base which contains full information about the query seems an economical solution. In relational databases, such a view can be created using SQL, but in ontologies, mature approaches are short. The only existing approach

for creating views of ontologies, which is based on a notion of ontology modularization [19], has been found deficient because the view computed by the approach often involves a large amount of irrelevant information, and thus is still large. Querying such a view still takes a long time to reason about and requires much memory.

iv. **Information Hiding**: As pointed out by Cuenca Grau [6], ontologies are increasingly used in a range of knowledge-based systems that deal with sensitive information, for example, in modern healthcare systems. If such information is accessed by different users, it is a critical requirement that complete confidentiality of the information is preserved, and that users have different access on the information depending on their privileges. Such privileges could for example restrict the visibility of certain terms. One approach to handle hidden terms is to share a view that only uses the terms specific users are allowed to access. This approach is particularly useful when the owners of an ontology decide to share the ontology with other users or with the public, but only want to reveal non-confidential information.

Creating view of ontologies is also useful for many other tasks such as ontology alignment and merging [13], [21], versioning [14], debugging and repair [15], [20], and logical difference computation [10], [22]. For space reasons, we do not elaborate on them in great detail in this paper.

C. Basics of Forgetting

The *signature* (*vocabulary*) of an ontology is the set of the terms in the ontology. *Forgetting* is a form of non-standard reasoning concerned with eliminating from an ontology a set of concept and role names in its signature, namely the *forgetting signature*, in such a way that all logical consequences are preserved up to the remaining signature. Forgetting can therefore be used as a means for computing views of ontologies in such a way that the ontology obtained from forgetting, namely the *forgetting solution*, is a view of the original ontology for *the target signature*, which amounts to the remaining signature in forgetting; we will elaborate this in the next section when formalizing the notion of forgetting.

Forgetting is an inherently difficult problem; it is much harder than standard reasoning (satisfiability testing or SAT), and very few logics are known to be complete for forgetting.[1] Foundational studies [11] have shown that: (i) forgetting solutions do not always exist for $\mathcal{ALC}$, (ii) deciding the existence of forgetting solutions is 2ExpTime-complete for $\mathcal{ALC}$, and (iii) forgetting solutions can be triple exponential in size w.r.t. the input ontologies for $\mathcal{ALC}$.

[1]When we say a logic $\mathcal{L}$ is complete for forgetting, we mean that for any forgetting problem specified in $\mathcal{L}$, there always exists a forgetting solution in $\mathcal{L}$. When we say a forgetting method is complete for a logic $\mathcal{L}$, we mean that for any forgetting problem specified in $\mathcal{L}$, the method can always compute a forgetting solution; in this case, the forgetting solution is not necessarily in $\mathcal{L}$. This means that for a specific logic $\mathcal{L}$, if $\mathcal{L}$ is not complete for forgetting, and the target language is not allowed to be extended, then any forgetting methods must be incomplete for $\mathcal{L}$, because, in principle, such a forgetting solution does not exist in $\mathcal{L}$ and thus cannot be computed by any methods.

Nevertheless, there is general consensus on the tremendous potential of forgetting for ontology-based knowledge processing, and there has been continuous dedication to the development of practical methods for computing solutions of forgetting. A few such methods have thus been developed and automated for various description logics.

Existing forgetting methods for ontologies are LETHE [8], the method of [10], and FAME [24]. LETHE is based on the classic inference system *resolution* [4], [5]; the method can eliminate concept and role names from $\mathcal{ALCH}$-ontologies. The method of [10] is another resolution-based approach able to eliminate concept names from $\mathcal{ALC}$-TBoxes. FAME is based on a monotonicity property called *Ackermann's Lemma* [1], and it considers a stronger notion of forgetting called model-theoretic forgetting [23], which is different from the notion used in this paper.

A major drawback of LETHE is that the method can only handle small ontologies, and cannot perform forgetting on slightly larger ones; A casual test showed that LETHE always got stuck during the forgetting process when eliminating a number of concept names from the Gene Ontology (GO) [2], which contains 127,830 axioms, 50,119 concept names, and 9 role names. This is far from satisfactory for industry uses. The method of [10] adapts a depth-bounded version of its core algorithm to guarantee completeness, which leads to the ontology obtained from forgetting being only an approximation of the real forgetting solution, and thus weaker than the real solution. This does not meet the needs of most realistic applications. Moreover, it eliminates only concept names from $\mathcal{ALC}$-TBoxes, but does not support forgetting of role names or with ABoxes. Hence, there have been no practical methods and automated tools available so far for forgetting for very large ontologies and for very expressive DLs.

D. Our Contribution

In this paper, we consider catering for the DL $\mathcal{ALCOIH}$ as both source and target languages. In particular, we develop a practical forgetting method for computing views of ontologies expressible in the description logic $\mathcal{ALCOIH}$, one of the most expressive DLs that has frequently been used as a representational underpinning but has never been considered for forgetting before. Being based on a set of inference rules, the method always *terminates*, and is *sound* in the sense that the forgetting solution computed by the method preserves the same logical consequences with the original ontology in the remaining signature. Despite the inherent difficulty of forgetting for this level of expressivity, for which our forgetting method is incomplete, an empirical evaluation with a prototype implementation showed superb results on a large corpus of real-world ontologies: (i) the method succeeded in more than 92% of the test cases, and (ii) in more than 91% of these successful cases, the forgetting solution was computed within only a few seconds. We compared our prototype with

the peer LETHE system on $\mathcal{ALCH}$-ontologies,[2] with the results showing that: our prototype was about 27 times faster on average, attained better success rates by 28.0% when a timeout of 1000 seconds was used, and had lower memory consumption during the forgetting process. We then applied our prototype to the ontology-based query answering problem; in particular, we used the prototype to compute 8 specific views of the current version of the SNOMED CT ontology [18], and retrieved data from both the original ontology and each view with a set of 100 artifact DL queries generated using Protégé [12]. The results showed that: in all the test cases, the view returned the same answers as the original ontology, while with a reduction of the response time by an average of 498%, depending on how much percent of terms in the signature was selected into the forgetting signature and eliminated from the original ontology. Usually, the more the terms are forgotten, the smaller the size of the computed view was, and the shorter the response time could be expected.

To sum up, the *contribution* of this paper is a novel and effective framework for creating views of ontologies. The framework is based on a reasoning procedure called forgetting, for which we have developed the first method for ontologies expressible in $\mathcal{ALCOIH}$, one of the most expressive DLs that has frequently been used as a representational underpinning but has never been considered for forgetting before. Compared to the peer LETHE system, a prototype of our method has shown much better success rates and performance results considering speed and memory consumption. This is very useful from the perspective of ontology engineering, as it provides ontology curators with a powerful tool for creating views of ontologies.

II. Preliminaries

Let $\mathsf{N_C}$, $\mathsf{N_R}$ and $\mathsf{N_I}$ be pairwise disjoint and countably infinite sets of *concept names*, *role names* and *individual names* (aka *nominals*), respectively. *Concepts* in $\mathcal{ALCOIH}$ (or *concepts* for short) have one of the following forms:

$$\top \mid \bot \mid \{a\} \mid A \mid \neg C \mid C \sqcap D \mid C \sqcup D \mid \exists R.C \mid \forall R.C,$$

where $a \in \mathsf{N_I}$, $A \in \mathsf{N_C}$, C and D are arbitrary concepts, and R is an arbitrary role name $r \in \mathsf{N_R}$ or the inverse r^- of a role name r (an inverse role). We assume without loss of generality that concepts and roles are equivalent relative to associativity and commutativity of $\sqcap$ and $\sqcup$, $\neg$ and $^-$ are involutions, and $\top$ ($\bot$) is a unit w.r.t. $\sqcap$ ($\sqcup$).

An $\mathcal{ALCOIH}$-*ontology* consists of a TBox, an RBox, and an ABox. A *TBox* is a finite set of axioms of the form $C \sqsubseteq D$ (*concept inclusions*) and the form $C \equiv D$ (*concept equivalences*), where C and D are concepts. An *RBox* is a finite set of axioms of the form $R \sqsubseteq S$ (*role inclusions*) and the form $R \equiv S$ (role equivalences), where R and S are roles. Usually, the RBox is regarded as part of the TBox; we distinguish them in this paper for the sake of clarity. An *ABox* is a finite set of axioms of the form $C(a)$ (*concept assertions*) and the form $R(a, b)$ (role assertions), where $a, b \in \mathsf{N_I}$, C is a concept, and R is a role. In DLs with nominals, ABox assertions are superfluous, because they can be internalized as concept inclusions via nominals, namely $C(a)$ as $\{a\} \sqsubseteq C$ and $R(a, b)$ as $\{a\} \sqsubseteq \exists R.\{b\}$. Similarly, concept (role) equivalences $C \equiv D$ ($R \equiv S$) can be internalized as concept (role) inclusions $C \sqsubseteq D$ ($R \sqsubseteq S$) and $D \sqsubseteq C$ ($S \sqsubseteq R$). Hence, in this paper, an $\mathcal{ALCOIH}$-ontology is assumed to contain only concept and role inclusions.

The semantics of $\mathcal{ALCOIH}$ is defined using an *interpretation* $\mathcal{I} = \langle \Delta^{\mathcal{I}}, \cdot^{\mathcal{I}} \rangle$, where $\Delta^{\mathcal{I}}$ denotes the *domain of the interpretation* (a non-empty set), and $\cdot^{\mathcal{I}}$ denotes the *interpretation function*, which assigns to every nominal $a \in \mathsf{N_I}$ a singleton $a^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$, to every concept name $A \in \mathsf{N_C}$ a set $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$, and to every role name $r \in \mathsf{N_R}$ a binary relation $r^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$. The interpretation function $\cdot^{\mathcal{I}}$ is inductively extended to concepts as follows:

$$\top^{\mathcal{I}} = \Delta^{\mathcal{I}} \qquad \bot^{\mathcal{I}} = \emptyset \qquad (\neg C)^{\mathcal{I}} = \Delta^{\mathcal{I}} \backslash C^{\mathcal{I}}$$
$$(C \sqcap D)^{\mathcal{I}} = C^{\mathcal{I}} \cap D^{\mathcal{I}} \qquad (C \sqcup D)^{\mathcal{I}} = C^{\mathcal{I}} \cup D^{\mathcal{I}}$$
$$(\exists R.C)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid \exists y.(x, y) \in R^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}$$
$$(\forall R.C)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid \forall y.(x, y) \in R^{\mathcal{I}} \rightarrow y \in C^{\mathcal{I}}\}$$
$$(R^-)^{\mathcal{I}} = \{(y, x) \in \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \mid (x, y) \in R^{\mathcal{I}}\}$$

Let $\mathcal{I}$ be an interpretation. A concept inclusion $C \sqsubseteq D$ is *true* in $\mathcal{I}$ ($\mathcal{I}$ satisfies $C \sqsubseteq D$) iff $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$. A role inclusion $r \sqsubseteq s$ is *true* in $\mathcal{I}$ ($\mathcal{I}$ satisfies $r \sqsubseteq s$) iff $r^{\mathcal{I}} \subseteq s^{\mathcal{I}}$. $\mathcal{I}$ is a *model* of an ontology $\mathcal{O}$ iff every axiom in $\mathcal{O}$ is *true* in $\mathcal{I}$. In this case, we write $\mathcal{I} \models \mathcal{O}$.

By $\mathsf{sig_C}(X)$, $\mathsf{sig_R}(X)$ and $\mathsf{sig_I}(X)$ we denote respectively the sets of the concept names, role names and individual names occurring in X, where X ranges over concepts, roles, axioms, and ontologies. We let $\mathsf{sig}(X) = \mathsf{sig_C}(X) \cup \mathsf{sig_R}(X)$.

Definition 1 (Forgetting). *Let $\mathcal{O}$ be an $\mathcal{ALCOIH}$-ontology and let $\mathcal{F} \subseteq \mathsf{sig}(\mathcal{O})$ be a set of concept and role names. We say that an $\mathcal{ALCOIH}$-ontology $\mathcal{V}$ is a* solution of forgetting $\mathcal{F}$ from $\mathcal{O}$ *iff the following conditions hold: (i) $\mathsf{sig}(\mathcal{V}) \subseteq \mathsf{sig}(\mathcal{O}) \backslash \mathcal{F}$, and (ii) for any axiom α with $\mathsf{sig}(\alpha) \subseteq \mathsf{sig}(\mathcal{O}) \backslash \mathcal{F}$, $\mathcal{V} \models \alpha$ iff $\mathcal{O} \models \alpha$.*

Definition 1 says that $\mathcal{V}$ (the forgetting solution) has the same logical consequences with $\mathcal{O}$ (the original ontology) in the remaining signature $\mathsf{sig}(\mathcal{O}) \backslash \mathcal{F}$. $\mathcal{F}$ is called the *forgetting signature*, i.e., the set of concept and role names to be eliminated. $\mathcal{V}$ can be regarded as a *view* of $\mathcal{O}$ w.r.t. the remaining signature $\mathsf{sig}(\mathcal{O}) \backslash \mathcal{F}$ in the sense that it gives the same answers as $\mathcal{O}$ to the queries formulated using the names in $\mathsf{sig}(\mathcal{O}) \backslash \mathcal{F}$. In traditional databases, a view is a subset of the database, whereas in ontologies, a view is more than a subset; it contains not only axioms that are contained in the original ontology, but also newly derived axioms that are entailed by the original ontology (implicitly contained in the original ontology). Such

[2] LETHE can also handle $\mathcal{SIF}$- and $\mathcal{SHQ}$-TBoxes [8], but is only able to eliminate concept names from them, but not role names. We did not consider a comparison of our method with the method of [10], because the latter computes only approximations of the real forgetting solutions, and thus do not meet most realistic applications' needs. Moreover, the implementation of the method is not publicly available at present.

new axioms can be derived during the forgetting process. The remaining signature in forgetting corresponds to the *target signature* in the problem of computing views of ontologies.

A view is the strongest entailment of the original ontology in the target signature. By definition, $\mathcal{V}$ is a *strongest entailment* of $\mathcal{O}$ in $\mathsf{sig}(\mathcal{O})\backslash\mathcal{F}$, if $\mathcal{O} \models \mathcal{V}$ and for any ontology $\mathcal{V}'$ such that $\mathcal{O} \models \mathcal{V}'$ and $\mathsf{sig}(\mathcal{V}') \subseteq \mathsf{sig}(\mathcal{O})\backslash\mathcal{F}$, then $\mathcal{V} \models \mathcal{V}'$. In general it can be shown that: $\mathcal{V}$ is a view of an ontology $\mathcal{O}$ for a specific target signature iff $\mathcal{V}$ is the strongest entailment of $\mathcal{O}$ in this signature. Views are unique up to logical equivalence, that is, if both $\mathcal{V}$ and $\mathcal{V}'$ are views of $\mathcal{O}$ for a specific target signature, then they are logically equivalent, though their representations may not be identical.

III. Ontology Normalization

A. Clausal Normal Form Transformation

Reasoning procedures are typically based on a set of generalized inference rules, so they require the input ontology to be normalized in some way in order to facilitate the generalization of the inference rules. Our forgetting method works on $\mathcal{ALCOIH}$-ontologies in *clausal normal form*.

Definition 2 (**Clausal Normal Form**). *A* TBox literal *in $\mathcal{ALCOIH}$ is a concept of the form a, $\neg a$, A, $\neg A$, $\exists R.C$ or $\forall R.C$, where $a \in \mathsf{N_I}$, $A \in \mathsf{N_C}$, C is a concept, and R is a role. A* TBox clause *in $\mathcal{ALCOIH}$ is a finite disjunction of literals. An* RBox clause *in $\mathcal{ALCOIH}$ is a disjunction of a role and a negated role. A clause is called an X-*clause *if it contains an occurrence of X, for $X \in \mathsf{N_C} \cup \mathsf{N_R}$. An $\mathcal{ALCOIH}$-ontology $\mathcal{O}$ is in* clausal normal form *if every axiom in $\mathcal{O}$ is either a TBox clause or an ABox clause.*

Clauses are obtained from corresponding axioms by incrementally applying the *standard transformations*, shown in Figure 1. The transformations are based on logical equivalence, i.e., the left-hand side expression of the '$\Longrightarrow$' relation is logically equivalent to the right-hand side one. In the remainder of this paper, we use the notation $\mathcal{N}$ to denote an $\mathcal{ALCOIH}$-ontology $\mathcal{O}$ in clausal normal form (a set $\mathcal{N}$ of clauses).

Lemma 1. *By incrementally applying the standard transformations, any $\mathcal{ALCOIH}$-ontology can be transformed into an equivalent one in clausal normal form.*

Example 1. *Consider the following ontology $\mathcal{O}$:*

1. $A \sqcap B \sqsubseteq C$
2. $\exists r.B \sqsubseteq A$
3. $a \sqsubseteq \neg\forall s.\neg B$
4. $\forall r.C \sqsubseteq \forall r.\exists s.A$
5. $\forall r.A \sqsubseteq \exists s.A \sqcup \forall t.A$

Observe that Axioms 1 – 5 are currently not in clausal normal form, but by applying the transformations in Figures 1, $\mathcal{O}$ can be transformed into the following set $\mathcal{N}$ of clauses:

$1'.\ \neg A \sqcup \neg B \sqcup C$
$2'.\ \forall r.\neg B \sqcup A$
$3'.\ \neg a \sqcup \exists s.B$
$4'.\ \exists r.\neg C \sqcup \forall r.\exists s.A$
$5'.\ \exists r.\neg A \sqcup \exists s.A \sqcup \forall t.A$

TBox axioms into TBox clauses:

$$C \sqsubseteq D \quad \Longrightarrow \quad \neg C \sqcup D$$
$$C \equiv D \quad \Longrightarrow \quad \neg C \sqcup D, \neg D \sqcup C$$

ABox assertions into TBox clauses:

$$C(a) \quad \Longrightarrow \quad \neg a \sqcup C$$
$$r(a,b) \quad \Longrightarrow \quad \neg a \sqcup \exists r.b$$

De Morgan's laws:

$$\neg(C \sqcap D) \quad \Longrightarrow \quad \neg C \sqcup \neg D$$
$$\neg(C \sqcup D) \quad \Longrightarrow \quad \neg C \sqcap \neg D$$

Duality between $\exists$- and $\forall$-restrictions:

$$\neg\exists r.C \quad \Longrightarrow \quad \forall r.\neg C$$
$$\neg\forall r.C \quad \Longrightarrow \quad \exists r.\neg C$$

Duality between $\top$ and $\bot$:

$$\neg\top \quad \Longrightarrow \quad \bot$$
$$\neg\bot \quad \Longrightarrow \quad \top$$

Double negation elimination:

$$\neg\neg C \quad \Longrightarrow \quad C$$

Distributivity law:

$$C \sqcup (D_1 \sqcap D_2) \quad \Longrightarrow \quad C \sqcup D_1, C \sqcup D_2$$

Fig. 1: Transformations into clausal normal form

Let X be a concept/role name. An occurrence of X is said to be *positive* (*negative*) in an X-clause if it is under an *even* (*odd*) number of explicit and implicit negations. For instance, the concept name A is positive in $C \sqcup A$, $C \sqcup \exists r.A$, and $C \sqcup \forall r.A$, and negative in $C \sqcup \neg A$, $C \sqcup \exists r.\neg A$, and $C \sqcup \forall r.\neg A$; the role name r is positive in $C \sqcup \exists r.D$ and $\neg s \sqcup r$, and negative in $C \sqcup \forall r.D$ and $\neg r \sqcup s$.[3]

B. Reduced Form Transformation

Next, we introduce two specialized normal forms, which are based on clausal normal form, namely *A-reduced form* and *r-reduced form*. These forms are crucial because they are used in the main calculi of our forgetting method for concept name

[3]Note that a role name is assumed to be negative under the $\forall$-restriction, because the first-order expression of $\forall r.$ is $\neg r(x, y)$, where r is negative.

and role name elimination, respectively, described in detail in the next section.

Definition 3 (A**-Reduced Form**). *Let $A \in \mathsf{sig_C}(\mathcal{N})$. A clause is in* A-reduced form *if it has the form $C \sqcup A$, $C \sqcup \neg A$, $C \sqcup \exists r.A$, $C \sqcup \exists r.\neg A$, $C \sqcup \exists r^-.A$, $C \sqcup \exists r^-.\neg A$, $C \sqcup \forall r.A$ or $C \sqcup \forall r.\neg A$, where $r \in \mathsf{N_R}$ is an arbitrary role name, and C (D) is a clause (concept) that does not contain A. $\mathcal{N}$ is in* A-reduced form *if every A-clause in $\mathcal{N}$ is in A-reduced form.*

A-reduced form includes all reachable basic forms of A-clauses in which a concept name A could occur. In particular, A could occur (either positively or negatively) at the surface level of an A-clause, or under an $\exists$- or a $\forall$-restriction.

Definition 4 (r**-Reduced Form**). *Let $r \in \mathsf{sig_R}(\mathcal{N})$. A TBox clause is in* r-reduced form *if it has the form $C \sqcup \exists r.D$, $C \sqcup \exists r^-.D$ or $C \sqcup \forall r.D$, where C (D) is a clause (concept) that does not contain r. An RBox clause is in* r-reduced form *if it has the form $\neg S \sqcup r$ or $\neg r \sqcup S$, where S is a role that does not contain r. $\mathcal{N}$ is in* r-reduced form *iff every r-clause in $\mathcal{N}$ is in r-reduced form.*

Like A-reduced form, r-reduced form includes all reachable basic forms of r-clauses in which a role name r could occur; in particular, r could occur (either in itself or being inverted) immediately under an $\exists$- or a $\forall$-restriction.

Apparently, not every A- or r-clause is initially in A- or r-reduced form. TBox A-clauses not in A-reduced form could have the form $C \sqcup \exists S.D$ or $C \sqcup \forall S.D$, where S is any role, C is a clause that contains A, and D is a concept that contains A. TBox r-clauses not in r-reduced form could have the form $C \sqcup \forall r^-.D$, where C is a clause that does not contain r, and D is a concept that does not contain r, or have the form $C \sqcup \exists S.D$ or $C \sqcup \forall S.D$, where S is any role, and the entire clause contains at least two occurrences of r. RBox r-clauses not in r-reduced form could have the form $\neg S \sqcup r^-$ or $\neg r^- \sqcup S$, where S is a role that does not contain r.[4]

TBox $A(r)$-clauses not in $A(r)$-reduced form can be transformed into the form (i) by introducing auxiliary concept names, namely *definers*, and (ii) by applying the surfacing rule, shown in Figure 2. RBox r-clauses not in r-reduced form can be transformed into the form by applying the inverting rules, shown in Figure 3.

Definers are fresh concept names externally introduced to facilitate the transformation of TBox clauses into the right reduced form [9]: let $\mathsf{N_D} \subset \mathsf{N_C}$ be a set of definers disjoint from $\mathsf{sig_C}(\mathcal{N})$. Definers are introduced and used as substitutes, incrementally replacing the clause "C" and the concept "D" in every TBox A-clause not in A-reduced form and in every TBox r-clause not in r-reduced form, until A-reduced form (r-reduced form) has been reached. For each clause (concept) replacement, a new clause $\neg \mathsf{D} \sqcup C$ ($\neg \mathsf{D} \sqcup D$) is added to $\mathcal{N}$, where $\mathsf{D} \in \mathsf{N_D}$ is a fresh definer.

[4]It is generally assumed that $\mathcal{ALCOIH}$ does not admit the DL expressivity of reflexivity, which can be encoded with the axioms $S \sqsubseteq S^-$ or $S^- \sqsubseteq S$.

Lemma 2. *The ontology obtained from the definer introduction has the same logical consequences as the original ontology in the signature of the original ontology.*

Proof. Definer introduction amounts to the standard structural transformation [8], which has polynomial (often constant) overhead and preserves all logical consequences in the signature of the original ontology. □

Example 2. *Consider the following set $\mathcal{N}$ of clauses:*

1. $\neg A \sqcup \neg B \sqcup C$ 2. $\forall r.\neg B \sqcup A$ 3. $\neg a \sqcup \exists s.B$
4. $\exists r.\neg C \sqcup \forall r.\exists s.A$ 5. $\exists r.\neg A \sqcup \exists s.A \sqcup \forall t.A$

Let $\mathcal{F} = \{A\}$. Observe that A-clauses 4 and 5 are not in A-reduced form. In this case, the first step is to introduce a definer to replace the concept $\exists s.A$ in Clause 4. This yields the following two clauses:

$$4'.\ \exists r.\neg C \sqcup \forall r.\mathsf{D}_1 \qquad 4''.\ \neg \mathsf{D}_1 \sqcup \exists s.A$$

The second step is to introduce a new definer to replace the concept $\exists r.\neg A$ in Clause 5, yielding the following clauses:

$$5'.\ \mathsf{D}_2 \sqcup \exists s.A \sqcup \forall t.A \qquad 5''.\ \neg \mathsf{D}_2 \sqcup \exists r.\neg A$$

At this stage, observe that Clause 5′ is still not in A-reduced form yet. The next step is then to introduce a definer to replace the concept $\exists s.A$ in Clause 5′. This case is somewhat special however, in that one can either introduce a fresh definer, or use the introduced definer D_1 to replace the concept $\exists s.A$, because $\exists s.A$ has been previously defined in Clause 4″ by D_1. Our method follows the latter; it implements a mechanism of reusing definers. In this manner, $\mathcal{N}$ is transformed into the following clause set in A-reduced form:

1. $\neg A \sqcup \neg B \sqcup C$ 2. $\forall r.\neg B \sqcup A$ 3. $\neg a \sqcup \exists s.B$
4. $\exists r.\neg C \sqcup \forall r.\mathsf{D}_1$ 5. $\neg \mathsf{D}_1 \sqcup \exists s.A$
6. $\mathsf{D}_2 \sqcup \mathsf{D}_1 \sqcup \forall t.A$ 7. $\neg \mathsf{D}_2 \sqcup \exists r.\neg A$

Definer reuse is a novel yet practical feature of our method – definers are desired not to be in the forgetting solutions, and they must be eliminated once the names in the forgetting signature have been eliminated, so introducing as few definers as possible is good for the efficiency of the method.

$$C \sqcup \forall r^-.D \qquad \Longrightarrow \qquad D \sqcup \forall r.C$$

Fig. 2: The surfacing rule

Observe that A-reduced form does not include the form $C \sqcup \forall r^-.A$ and $C \sqcup \forall r^-.\neg A$, where A occurs under a $\forall$-restriction of an inverse role. This is because A-clauses of these two forms can be transformed into A-reduced form using *the surfacing rule*, shown in Figure 2. The aim of the surfacing rule is, on the one hand, to move outward a concept name under a $\forall$-restriction between $\forall r$ and $\forall r^-$ (this is useful for concept forgetting; the rule facilitates the transformation into A-reduced form), and on the other hand, to remove an

inverse operator upon a role name immediately under a $\forall$-restriction (this is useful for role forgetting; the rule facilitates the transformation into r-reduced form, described immediately next). The surfacing rule is sound in the same sense as the standard transforms: the left-hand side expression of the '$\Longrightarrow$' relation is logically equivalent to the right-hand side one.

Lemma 3. *The surfacing rule preserves logical equivalence.*

Proof. We first prove the "*right-left*" direction. The proof can be done by contradiction. Suppose there exists an element $d \in \Delta^{\mathcal{I}}$ such that $d \notin (\forall R^{-}.C \sqcup D)^{\mathcal{I}}$.

$$d \notin (\forall R^{-}.C \sqcup D)^{\mathcal{I}} \tag{1}$$
$$\Rightarrow d \notin (\forall R^{-}.C)^{\mathcal{I}} \text{ and } d \notin D^{\mathcal{I}} \tag{2}$$
$$\Rightarrow d \notin (\forall R^{-}.C)^{\mathcal{I}} \tag{3}$$

This means that there exists an element $d' \in \Delta^{\mathcal{I}}$ such that:

$$(d^{\mathcal{I}}, d'^{\mathcal{I}}) \in (R^{-})^{\mathcal{I}} \text{ and } d' \notin C^{\mathcal{I}} \tag{4}$$
$$\Rightarrow (d'^{\mathcal{I}}, d^{\mathcal{I}}) \in R^{\mathcal{I}} \text{ and } d' \notin C^{\mathcal{I}} \tag{5}$$

Since the premise is true in $\mathcal{I}$, then for every x,

$$x \in (C \sqcup \forall R.D)^{\mathcal{I}} \tag{6}$$
$$\Rightarrow x \in C^{\mathcal{I}} \text{ or } x \in (\forall R.D)^{\mathcal{I}} \tag{7}$$
$$\Rightarrow x \in (\forall R.D)^{\mathcal{I}} \tag{8}$$
$$\Rightarrow (x^{\mathcal{I}}, y^{\mathcal{I}}) \in R^{\mathcal{I}} \rightarrow y \in D^{\mathcal{I}} \tag{9}$$
$$\Rightarrow (d'^{\mathcal{I}}, d^{\mathcal{I}}) \in R^{\mathcal{I}} \rightarrow d \in D^{\mathcal{I}} \tag{10}$$
$$\Rightarrow d \in D^{\mathcal{I}} \tag{11}$$

Contradiction. We then prove the "*left-right*" direction. We do the proof by contradiction too. Suppose there exists an element $d \in \Delta^{\mathcal{I}}$ such that $d \notin (C \sqcup \forall R.D)^{\mathcal{I}}$.

$$d \notin (C \sqcup \forall R.D)^{\mathcal{I}} \tag{12}$$
$$\Rightarrow d \notin C^{\mathcal{I}} \text{ and } d \notin (\forall R.D)^{\mathcal{I}} \tag{13}$$
$$\Rightarrow d \notin (\forall R.D)^{\mathcal{I}} \tag{14}$$

This means that there exists an element $d' \in \Delta^{\mathcal{I}}$ such that:

$$(d^{\mathcal{I}}, d'^{\mathcal{I}}) \in R^{\mathcal{I}} \text{ and } d' \notin D^{\mathcal{I}} \tag{15}$$

Since the premise is true in $\mathcal{I}$, then for every x,

$$x \in (\forall R^{-}.C \sqcup D)^{\mathcal{I}} \tag{16}$$
$$\Rightarrow x \in (\forall R^{-}.C)^{\mathcal{I}} \text{ or } x \in D^{\mathcal{I}} \tag{17}$$
$$\Rightarrow x \in (\forall R^{-}.C)^{\mathcal{I}} \tag{18}$$
$$\Rightarrow (x^{\mathcal{I}}, y^{\mathcal{I}}) \in (R^{-})^{\mathcal{I}} \rightarrow y \in C^{\mathcal{I}} \tag{19}$$
$$\Rightarrow (y^{\mathcal{I}}, x^{\mathcal{I}}) \in R^{\mathcal{I}} \rightarrow y \in C^{\mathcal{I}} \tag{20}$$
$$\Rightarrow (d^{\mathcal{I}}, d'^{\mathcal{I}}) \in R^{\mathcal{I}} \rightarrow d \in C^{\mathcal{I}} \tag{21}$$
$$\Rightarrow d \in C^{\mathcal{I}} \tag{22}$$

Contradiction. Therefore, the surfacing rule preserves logical equivalence in the transformation. □

Likewise, the form $C \sqcup \forall r^{-}.D$ is not included in the specification of r-reduced form because of the surfacing rule.

$$\neg S \sqcup r^{-} \quad \Longrightarrow \quad \neg S^{-} \sqcup r$$
$$\neg r^{-} \sqcup S \quad \Longrightarrow \quad \neg r \sqcup S^{-}$$

Fig. 3: The inverting rules

The form $\neg S \sqcup r^{-}$ and $\neg r^{-} \sqcup S$ are not included because of the inverting rules, shown in Figure 3, which can transform them into r-reduced form. Specifically, the aim of the inverting rules is to free a role from an inverse operator; they are used to invert back an inverse role.

Example 3. *Consider the following set $\mathcal{N}$, obtained from the normalization of $\mathcal{O}$ initially given in Example 2:*

1. $\neg A \sqcup \neg B \sqcup C$
2. $\forall r.\neg B \sqcup A$
3. $\neg a \sqcup \exists s.B$
4. $\exists r.\neg C \sqcup \forall r.\exists s.A$
5. $\exists r.\neg A \sqcup \exists s.A \sqcup \forall t.A$

Let $\mathcal{F} = \{A\}$. Observe that A-clauses 4 and 5 are not in A-reduced form. In this case, we can introduce definers to transform them into A-reduced form. The first step is to introduce a definer to replace the concept $\exists s.A$ in Clause 4. This would yield the following clauses:

$4'$. $\exists r.\neg C \sqcup \forall r.\mathsf{D}_1$
$4''$. $\neg \mathsf{D}_1 \sqcup \exists s.A$

The second step is to introduce a new definer to replace the concept $\exists r.\neg A$ in Clause 5, yielding the following clauses:

$5'$. $\mathsf{D}_2 \sqcup \exists s.A \sqcup \forall t.A$
$5''$. $\neg \mathsf{D}_2 \sqcup \exists r.\neg A$

At this stage, observe that Clause $5'$ is still not in A-reduced form yet. The next step is then to introduce a definer to replace the concept $\exists s.A$ in Clause $5'$. This case is somewhat special however, in that one can either introduce a fresh definer, or use the introduced definer D_1 to replace the concept $\exists s.A$, because $\exists s.A$ has been previously defined in Clause $4''$ by D_1. Our method follows the latter; it implements a mechanism of reusing definers. In this manner, $\mathcal{N}$ is transformed into the following clause set in A-reduced form:

1. $\neg A \sqcup \neg B \sqcup C$
2. $\forall r.\neg B \sqcup A$
3. $\neg a \sqcup \exists s.B$
$4'$. $\exists r.\neg C \sqcup \forall r.\mathsf{D}_1$
$4''$. $\neg \mathsf{D}_1 \sqcup \exists s.A$
$5'$. $\mathsf{D}_2 \sqcup \mathsf{D}_1 \sqcup \forall t.A$
$5''$. $\neg \mathsf{D}_2 \sqcup \exists r.\neg A$

Lemma 4. *The inverting rules preserve logical equivalence.*

Proof. We prove the "left-right" direction by contradiction for the first inverting rule. Suppose there exists an element $d \in \Delta^{\mathcal{I}}$ and an element $d' \in \Delta^{\mathcal{I}}$ such that $(d, d') \notin (\neg \mathcal{S}^- \sqcup r)^{\mathcal{I}}$.

$$(d, d') \notin (\neg S^- \sqcup r)^{\mathcal{I}} \tag{23}$$
$$\Rightarrow (d, d') \notin (\neg S^-)^{\mathcal{I}} \text{ and } (d, d') \notin r^{\mathcal{I}} \tag{24}$$
$$\Rightarrow (d, d') \in (S^-)^{\mathcal{I}} \text{ and } (d, d') \notin r^{\mathcal{I}} \tag{25}$$
$$\Rightarrow (d', d) \in S^{\mathcal{I}} \text{ and } (d, d') \notin r^{\mathcal{I}} \tag{26}$$

Since the premise of the rule is true in $\mathcal{I}$, then for every x and y:

$$(x, y) \in (\neg S \sqcup r^-)^{\mathcal{I}} \tag{27}$$
$$\Rightarrow (x, y) \in (\neg S)^{\mathcal{I}} \text{ or } (x, y) \in (r^-)^{\mathcal{I}} \tag{28}$$
$$\Rightarrow (x, y) \notin S^{\mathcal{I}} \text{ or } (y, x) \in r^{\mathcal{I}} \tag{29}$$

Contradiction. Next, we prove the "right-left" direction. Suppose there exists an element $d \in \Delta^{\mathcal{I}}$ and an element $d' \in \Delta^{\mathcal{I}}$ such that $(d, d') \notin (\neg S \sqcup r^-)^{\mathcal{I}}$.

$$(d, d') \notin (\neg S \sqcup r^-)^{\mathcal{I}} \tag{30}$$
$$\Rightarrow (d, d') \notin (\neg S)^{\mathcal{I}} \text{ and } (d, d') \notin (r^-)^{\mathcal{I}} \tag{31}$$
$$\Rightarrow (d, d') \in S^{\mathcal{I}} \text{ and } (d', d) \notin r^{\mathcal{I}} \tag{32}$$

Since the conclusion of the rule is true in $\mathcal{I}$, then for every x and y:

$$(x, y) \in (\neg S^- \sqcup r)^{\mathcal{I}} \tag{33}$$
$$\Rightarrow (x, y) \in (\neg S^-)^{\mathcal{I}} \text{ or } (x, y) \in r^{\mathcal{I}} \tag{34}$$
$$\Rightarrow (x, y) \notin S^{-\mathcal{I}} \text{ or } (x, y) \in r^{\mathcal{I}} \tag{35}$$
$$\Rightarrow (y, x) \notin S^{\mathcal{I}} \text{ or } (x, y) \in r^{\mathcal{I}} \tag{36}$$

Contradiction. Therefore, the first inverting rule preserves logical equivalence in the transformation. The second inverting rule can be proved similarly. □

Definer introduction is performed before the application of the surfacing rule. Since concept names are never present in an RBox, the inverting rules are not used for concept forgetting.

Lemmas 1, 2, 3 and 4 guarantee soundness of the ontology normalization: the normalized ontology, which is a set $\mathcal{N}$ of clauses, has the same consequences as the original ontology $\mathcal{O}$ in the signature of $\mathcal{O}$. The normalization can be seen as the reverse operation of concept forgetting: concept forgetting eliminates concept names and preserves all logical consequences in the remaining signature, and the normalization introduces new concept names (definers) and preserves all logical consequences in the signature of the original ontology. In this sense, the intention of normalization seems against the that of forgetting, but we will show in a later example that, sometimes, a step backward is a step forward.

Lemma 5. *By introducing definers as described above and by applying the surfacing and the inverting rules, any set of clauses can be transformed into A-reduced form or r-reduced form in finite steps.*

IV. The Forgetting Method

Let $\mathcal{N}$ be a set of clauses and $\mathcal{F} \subseteq \mathsf{sig}(\mathcal{N})$ be a forgetting signature. The solution of forgetting $\mathcal{F}$ from $\mathcal{N}$ is computed by (one-by-one) eliminating single names in $\mathcal{F}$. Our forgetting method consists of two mutually independent calculi: (i) a calculus for eliminating a concept name A from $\mathcal{N}$, and (ii) a calculus for eliminating a role name r from $\mathcal{N}$.

These calculi are respectively based on a generalized inference rule. The rules are *replacement rules* that replace the clauses above the line (the *premises*) by those under the line (the *conclusion*). The rules are sound in the sense that the premises and the conclusion of the rule have the same logical consequences in the remaining signature (Conditions (i) and (ii) of Definition 1 hold). The calculi are sound in the sense that the output and the input of the calculus have the same logical consequences in the remaining signature.

A. Calculus for Concept Elimination

The calculus for eliminating a concept name A from $\mathcal{N}$ is based on an inference rule, namely *the combination rule*, shown in Figure 4. The combination rule is applicable to $\mathcal{N}$ to eliminate A iff $\mathcal{N}$ is in *A-reduced form*. Observe that clauses in A-reduced form, containing exactly one occurrence of A, can have at most eight distinct forms, which can be classified into two types, namely the *positive premises* and *negative premises*, denoted respectively by $\mathcal{P}^+(A)$ and $\mathcal{P}^-(A)$. The positive premises are the A-clauses in $\mathcal{N}$ where A occurs positively. The negative premises are the A-clauses in $\mathcal{N}$ where A occurs negatively. We use different notation to denote the set of the premises of different A-reduced forms; see Figure 5. By definition, $\mathcal{P}^+(A) = \mathcal{P}^+_{\mho}(A) \cup \mathcal{P}^+_{\exists}(A) \cup \mathcal{P}^+_{\exists,\_}(A) \cup \mathcal{P}^+_{\forall}(A)$ and $\mathcal{P}^-(A) = \mathcal{P}^-_{\mho}(A) \cup \mathcal{P}^-_{\exists}(A) \cup \mathcal{P}^-_{\exists,\_}(A) \cup \mathcal{P}^-_{\forall}(A)$. By $\mathcal{N}^{-A}$ we denote the set of clauses not containing A, i.e., non A-clauses.

The idea of the combination rule is to *combine* every positive premise with every negative one. More specifically, the idea is to resolve every positive premise and every negative one on the name being eliminated, which is A in this case. This is where the term "*combination*" comes from. For distinct forms of positive and negative premises, there are in total $4 \times 4 = 16$ different cases of combination. The result of combining a positive premise α with a negative one β is a finite set of clauses, denoted by $\mathsf{combine}(\alpha, \beta)$. $\mathsf{combine}(\alpha, \beta)$ does not contain A, and is the strongest entailment of α and β in the signature of $\mathsf{sig}(\alpha) \cup \mathsf{sig}(\beta)$ excluding A. That is, $\mathsf{combine}(\alpha, \beta)$ preserves all logical consequences of the original two premises in the remaining signature. Since every premise set like $\mathcal{P}^+_{\mho}(A)$ contains A-clauses of the same form, we consider them as a whole when applying the combination rule. Thus, the result of combining every premise in $\mathcal{P}^+_{\mho}(A)$ with every one in $\mathcal{P}^-_{\mho}(A)$ is the union $\mathsf{combine}(\mathcal{P}^+_{\mho}(A), \mathcal{P}^-_{\mho}(A))$ of the results obtained from every ground combination. More generally, the conclusion of the combination rule is the union $\mathsf{combine}(\mathcal{P}^+(A), \mathcal{P}^-(A))$ of every such $\mathsf{combine}(\mathcal{P}^+_{\mho}(A), \mathcal{P}^-_{\mho}(A))$. For space reasons, we show the conclusion of the combination rule "at the set level".

$$\frac{\mathcal{N}^{-\mathsf{A}}, \overbrace{B_1 \sqcup \mathsf{A}, \ldots, B_l \sqcup \mathsf{A}}^{\mathcal{P}^+_{\mathfrak{U}}(\mathsf{A})}, \overbrace{C_1 \sqcup \exists r_1.\mathsf{A}, \ldots, C_m \sqcup \exists r_m.\mathsf{A}}^{\mathcal{P}^+_{\exists}(\mathsf{A})}, \overbrace{D_1 \sqcup \exists s_1^-.\mathsf{A}, \ldots, D_n \sqcup \exists s_n^-.\mathsf{A}}^{\mathcal{P}^+_{\exists,-}(\mathsf{A})}, \overbrace{\phi_1 \sqcup \forall t_1.\mathsf{A}, \ldots, \phi_o \sqcup \forall t_o.\mathsf{A}}^{\mathcal{P}^+_{\forall}(\mathsf{A})} \quad \overbrace{E_1 \sqcup \neg\mathsf{A}, \ldots, E_{l'} \sqcup \neg\mathsf{A}}^{\mathcal{P}^-_{\mathfrak{U}}(\mathsf{A})}, \overbrace{F_1 \sqcup \exists u_1.\neg\mathsf{A}, \ldots, F_{m'} \sqcup \exists u_{m'}.\neg\mathsf{A}}^{\mathcal{P}^-_{\exists}(\mathsf{A})}, \overbrace{G_1 \sqcup \exists v_1^-.\neg\mathsf{A}, \ldots, G_{n'} \sqcup \exists v_{n'}^-.\neg\mathsf{A}}^{\mathcal{P}^-_{\exists,-}(\mathsf{A})}, \overbrace{\psi_1 \sqcup \forall w_1.\neg\mathsf{A}, \ldots, \psi_{o'} \sqcup \forall w_{o'}.\neg\mathsf{A}}^{\mathcal{P}^-_{\forall}(\mathsf{A})}}{\begin{array}{c}\mathcal{N}^{-\mathsf{A}}, \mathsf{combine}(\mathcal{P}^+_{\mathfrak{U}}(\mathsf{A}), \mathcal{P}^-_{\mathfrak{U}}(\mathsf{A})), \mathsf{combine}(\mathcal{P}^+_{\mathfrak{U}}(\mathsf{A}), \mathcal{P}^-_{\exists}(\mathsf{A})), \mathsf{combine}(\mathcal{P}^+_{\mathfrak{U}}(\mathsf{A}), \mathcal{P}^-_{\exists,-}(\mathsf{A})), \mathsf{combine}(\mathcal{P}^+_{\mathfrak{U}}(\mathsf{A}), \mathcal{P}^-_{\forall}(\mathsf{A})),\\ \mathsf{combine}(\mathcal{P}^+_{\exists}(\mathsf{A}), \mathcal{P}^-_{\mathfrak{U}}(\mathsf{A})), \mathsf{combine}(\mathcal{P}^+_{\exists}(\mathsf{A}), \mathcal{P}^-_{\exists}(\mathsf{A})), \mathsf{combine}(\mathcal{P}^+_{\exists}(\mathsf{A}), \mathcal{P}^-_{\exists,-}(\mathsf{A})), \mathsf{combine}(\mathcal{P}^+_{\exists}(\mathsf{A}), \mathcal{P}^-_{\forall}(\mathsf{A})),\\ \mathsf{combine}(\mathcal{P}^+_{\exists,-}(\mathsf{A}), \mathcal{P}^-_{\mathfrak{U}}(\mathsf{A})), \mathsf{combine}(\mathcal{P}^+_{\exists,-}(\mathsf{A}), \mathcal{P}^-_{\exists}(\mathsf{A})), \mathsf{combine}(\mathcal{P}^+_{\exists,-}(\mathsf{A}), \mathcal{P}^-_{\exists,-}(\mathsf{A})), \mathsf{combine}(\mathcal{P}^+_{\exists,-}(\mathsf{A}), \mathcal{P}^-_{\forall}(\mathsf{A})),\\ \mathsf{combine}(\mathcal{P}^+_{\forall}(\mathsf{A}), \mathcal{P}^-_{\mathfrak{U}}(\mathsf{A})), \mathsf{combine}(\mathcal{P}^+_{\forall}(\mathsf{A}), \mathcal{P}^-_{\exists}(\mathsf{A})), \mathsf{combine}(\mathcal{P}^+_{\forall}(\mathsf{A}), \mathcal{P}^-_{\exists,-}(\mathsf{A})), \mathsf{combine}(\mathcal{P}^+_{\forall}(\mathsf{A}), \mathcal{P}^-_{\forall}(\mathsf{A}))\end{array}}$$

Notation in the rule ($1 \leq h \leq l$, $1 \leq i \leq m$, $1 \leq j \leq n$, $1 \leq k \leq o$, $1 \leq h' \leq l'$, $1 \leq i' \leq m'$, $1 \leq j' \leq n'$, $1 \leq k' \leq o'$):
B_h, C_i, D_j, ϕ_k, $E_{h'}$, $F_{i'}$, $G_{j'}$ and $\psi_{k'}$ are any concepts that do not contain A; r_i, s_j, t_k, $u_{i'}$, $v_{j'}$ and $w_{k'}$ are any role names.

1: $\mathsf{combine}(\mathcal{P}^+_{\mathfrak{U}}(\mathsf{A}), \mathcal{P}^-_{\mathfrak{U}}(\mathsf{A})) = \bigcup_{1\leq h\leq l} \bigcup_{1\leq h'\leq l'} \{B_h \sqcup E_{h'}\}$ **2:** $\mathsf{combine}(\mathcal{P}^+_{\mathfrak{U}}(\mathsf{A}), \mathcal{P}^-_{\exists}(\mathsf{A})) = \bigcup_{1\leq h\leq l} \bigcup_{1\leq i'\leq m'} \{F_{i'} \sqcup \exists u_{i'}.B_h\}$

3: $\mathsf{combine}(\mathcal{P}^+_{\mathfrak{U}}(\mathsf{A}), \mathcal{P}^-_{\exists,-}(\mathsf{A})) = \bigcup_{1\leq h\leq l} \bigcup_{1\leq j'\leq n'} \{G_{j'} \sqcup \exists v_{j'}^-.B_h\}$ **4:** $\mathsf{combine}(\mathcal{P}^+_{\mathfrak{U}}(\mathsf{A}), \mathcal{P}^-_{\forall}(\mathsf{A})) = \bigcup_{1\leq h\leq l} \bigcup_{1\leq k'\leq o'} \{\psi_{k'} \sqcup \forall w_{k'}.B_h\}$

5: $\mathsf{combine}(\mathcal{P}^+_{\exists}(\mathsf{A}), \mathcal{P}^-_{\mathfrak{U}}(\mathsf{A})) = \bigcup_{1\leq i\leq m} \bigcup_{1\leq h'\leq l'} \{C_i \sqcup \exists r_i.E_{h'}\}$

6: $\mathsf{combine}(\mathcal{P}^+_{\exists}(\mathsf{A}), \mathcal{P}^-_{\exists}(\mathsf{A})) = \bigcup_{1\leq i\leq m} \bigcup_{1\leq i'\leq m'} \{C_i \sqcup \exists r_i.\top, F_{i'} \sqcup \exists u_{i'}.\top\}$

7: $\mathsf{combine}(\mathcal{P}^+_{\exists}(\mathsf{A}), \mathcal{P}^-_{\exists,-}(\mathsf{A})) = \bigcup_{1\leq i\leq m} \bigcup_{1\leq h'\leq l'} \{C_i \sqcup \exists r_i.\top, G_{j'} \sqcup \exists v_{j'}^-.\top\}$

8: $\mathsf{combine}(\mathcal{P}^+_{\exists}(\mathsf{A}), \mathcal{P}^-_{\forall}(\mathsf{A})) = \bigcup_{1\leq i\leq m} \bigcup_{1\leq k'\leq o'} \{C_i \sqcup \exists r_i.\top, C_i \sqcup \psi_{k'}\}$, for any r_i and $w_{k'}$ such that $r_i = w_{k'}$

9: $\mathsf{combine}(\mathcal{P}^+_{\exists,-}(\mathsf{A}), \mathcal{P}^-_{\mathfrak{U}}(\mathsf{A})) = \bigcup_{1\leq j\leq n} \bigcup_{1\leq h'\leq l'} \{D_j \sqcup \exists s_j^-.E_{h'}\}$

10: $\mathsf{combine}(\mathcal{P}^+_{\exists,-}(\mathsf{A}), \mathcal{P}^-_{\exists}(\mathsf{A})) = \bigcup_{1\leq j\leq n} \bigcup_{1\leq i'\leq m'} \{D_j \sqcup \exists s_j^-.\top, F_{i'} \sqcup \exists u_{i'}.\top\}$

11: $\mathsf{combine}(\mathcal{P}^+_{\exists,-}(\mathsf{A}), \mathcal{P}^-_{\exists,-}(\mathsf{A})) = \bigcup_{1\leq j\leq n} \bigcup_{1\leq j'\leq n'} \{D_j \sqcup \exists s_j^-.\top, G_{j'} \sqcup \exists v_{j'}^-.\top\}$

12: $\mathsf{combine}(\mathcal{P}^+_{\exists,-}(\mathsf{A}), \mathcal{P}^-_{\forall}(\mathsf{A})) = \bigcup_{1\leq j\leq n} \bigcup_{1\leq k'\leq o'} \{D_j \sqcup \exists s_j^-.\top\}$ **13:** $\mathsf{combine}(\mathcal{P}^+_{\forall}(\mathsf{A}), \mathcal{P}^-_{\mathfrak{U}}(\mathsf{A})) = \bigcup_{1\leq k\leq o} \bigcup_{1\leq h'\leq l'} \{\phi_k \sqcup \forall t_k.E_{h'}\}$

14: $\mathsf{combine}(\mathcal{P}^+_{\forall}(\mathsf{A}), \mathcal{P}^-_{\exists}(\mathsf{A})) = \bigcup_{1\leq k\leq o} \bigcup_{1\leq i'\leq m'} \{F_{i'} \sqcup \phi_k, F_{i'} \sqcup \exists u_{i'}.\top\}$, for any t_k and $u_{i'}$ such that $t_k = u_{i'}$

15: $\mathsf{combine}(\mathcal{P}^+_{\forall}(\mathsf{A}), \mathcal{P}^-_{\exists,-}(\mathsf{A})) = \bigcup_{1\leq k\leq o} \bigcup_{1\leq j'\leq n'} \{G_{j'} \sqcup \exists v_{j'}^-.\top\}$

16: $\mathsf{combine}(\mathcal{P}^+_{\forall}(\mathsf{A}), \mathcal{P}^-_{\forall}(\mathsf{A})) = \bigcup_{1\leq k\leq o} \bigcup_{1\leq k'\leq o'} \{\phi_k \sqcup \psi_{k'} \sqcup \forall t_k.\bot\}$, for any t_k and $w_{o'}$ such that $t_k = w_{o'}$

Fig. 4: The combination rule for eliminating a concept name $\mathsf{A} \in \mathsf{sig_C}(\mathcal{N})$ from a set $\mathcal{N}$ of clauses in A-reduced form

Positive Premise	Notation	Negative Premise	Notation
$C \sqcup A$	$\mathcal{P}^+_{\mathfrak{U}}(A)$	$C \sqcup \neg A$	$\mathcal{P}^-_{\mathfrak{U}}(A)$
$C \sqcup \exists r.A$	$\mathcal{P}^+_{\exists}(A)$	$C \sqcup \exists r.\neg A$	$\mathcal{P}^-_{\exists}(A)$
$C \sqcup \exists r^-.A$	$\mathcal{P}^+_{\exists,-}(A)$	$C \sqcup \exists r^-.\neg A$	$\mathcal{P}^-_{\exists,-}(A)$
$C \sqcup \forall r.A$	$\mathcal{P}^+_{\forall}(A)$	$C \sqcup \forall r.\neg A$	$\mathcal{P}^-_{\forall}(A)$

Fig. 5: Distinct forms of clauses in A-reduced form

Lemma 6. *The combination rule in Figure 4 is sound.*

Proof. To show that the combination rule is sound is to show that the premises and the conclusion of the rule have the same logical consequences up to the remaining signature. To this end, we first introduce a notion of equivalence, namely $\mathcal{F}$-*equivalence*.

Let $\mathcal{S} \in \mathsf{N_C} \cup \mathsf{N_R}$ be a designated concept/role name, and let $\mathcal{I}$ and $\mathcal{I}'$ be two interpretations. We say that $\mathcal{I}$ and $\mathcal{I}'$ are *equivalent up to* $\mathcal{S}$, or $\mathcal{S}$-*equivalent*, if $\mathcal{I}$ and $\mathcal{I}'$ coincide but differ possibly in the interpretations of $\mathcal{S}$. More generally, we say that $\mathcal{I}$ and $\mathcal{I}'$ are *equivalent up to a set* $\mathcal{F}$ *of concept and role names*, or $\mathcal{F}$-*equivalent*, if $\mathcal{I}$ and $\mathcal{I}'$ coincide but

$$\frac{\mathcal{N}^{-\mathsf{r}}, \overbrace{C_1 \sqcup \exists \mathsf{r}.D_1, \ldots, C_m \sqcup \exists \mathsf{r}.D_m}^{\mathcal{P}^+_\exists(\mathsf{r})}, \overbrace{E_1 \sqcup \exists \mathsf{r}^-.F_1, \ldots, E_n \sqcup \exists \mathsf{r}^-.F_n}^{\mathcal{P}^+_{\exists,-}(\mathsf{r})}, \overbrace{\neg S_1 \sqcup \mathsf{r}, \ldots, \neg S_o \sqcup \mathsf{r}}^{\mathcal{P}^+_{\mathcal{R}}(\mathsf{r})}, \overbrace{V_1 \sqcup \forall \mathsf{r}.W_1, \ldots, V_p \sqcup \forall \mathsf{r}.W_p}^{\mathcal{P}^-_{\forall}(\mathsf{r})}, \overbrace{\neg \mathsf{r}_1 \sqcup T_1, \ldots, \neg r_q \sqcup T_q}^{\mathcal{P}^-_{\mathcal{R}}(\mathsf{r})}}{\mathcal{N}^{-\mathsf{r}}, \mathsf{combine}(\mathcal{P}^+_\exists(\mathsf{r}), \mathcal{P}^-_\forall(\mathsf{r})), \mathsf{combine}(\mathcal{P}^+_\exists(\mathsf{r}), \mathcal{P}^-_{\mathcal{R}}(\mathsf{r})), \mathsf{combine}(\mathcal{P}^+_{\exists,-}(\mathsf{r}), \mathcal{P}^-_\forall(\mathsf{r})), \mathsf{combine}(\mathcal{P}^+_{\exists,-}(\mathsf{r}), \mathcal{P}^-_{\mathcal{R}}(\mathsf{r})), \mathsf{combine}(\mathcal{P}^+_{\mathcal{R}}(\mathsf{r}), \mathcal{P}^-_\forall(\mathsf{r})), \mathsf{combine}(\mathcal{P}^+_{\mathcal{R}}(\mathsf{r}), \mathcal{P}^-_{\mathcal{R}}(\mathsf{r}))}$$

1: $\mathsf{combine}(\mathcal{P}^+_\exists(\mathsf{r}), \mathcal{P}^-_\forall(\mathsf{r})) = \bigcup_{1 \le i \le m} \bigcup_{1 \le x \le p} \{C_i \sqcup V_x\}$, for any D_i, W_x s.t. $D_i \sqcap W_x \sqsubseteq \bot$

2: $\mathsf{combine}(\mathcal{P}^+_\exists(\mathsf{r}), \mathcal{P}^-_{\mathcal{R}}(\mathsf{r})) = \bigcup_{1 \le i \le m} \bigcup_{1 \le y \le q} \{C_i \sqcup \exists T_y.D_i\}$ **4:** $\mathsf{combine}(\mathcal{P}^+_{\exists,-}(\mathsf{r}), \mathcal{P}^-_{\mathcal{R}}(\mathsf{r})) = \bigcup_{1 \le j \le n} \bigcup_{1 \le y \le q} \{E_j \sqcup \exists T^-_y.F_j\}$

3: $\mathsf{combine}(\mathcal{P}^+_{\exists,-}(\mathsf{r}), \mathcal{P}^-_\forall(\mathsf{r})) = \bigcup_{1 \le j \le n} \bigcup_{1 \le x \le p} \{E_j \sqcup W_x\}$, for any F_j, V_x s.t. $F_j \sqcap V_x \sqsubseteq \bot$

5: $\mathsf{combine}(\mathcal{P}^+_{\mathcal{R}}(\mathsf{r}), \mathcal{P}^-_\forall(\mathsf{r})) = \bigcup_{1 \le k \le o} \bigcup_{1 \le x \le p} \{V_x \sqcup \forall S_k.W_x\}$ **6:** $\mathsf{combine}(\mathcal{P}^+_{\exists,-}(\mathsf{r}), \mathcal{P}^-_{\mathcal{R}}(\mathsf{r})) = \bigcup_{1 \le k \le o} \bigcup_{1 \le y \le q} \{\neg S_k \sqcup T_y\}$

Fig. 6: The combination rule for eliminating a role name $\mathsf{r} \in \mathsf{sig}_{\mathsf{R}}(\mathcal{N})$ from a set $\mathcal{N}$ of clauses in r-reduced form

differ possibly in the interpretations of the names in $\mathcal{F}$. This can be understood as follows: (i) $\mathcal{I}$ and $\mathcal{I}'$ have the same domain, i.e., $\Delta^{\mathcal{I}} = \Delta^{\mathcal{I}'}$, and interpret every individual name identically, i.e., $a^{\mathcal{I}} = a^{\mathcal{I}'}$ for every $a \in \mathsf{N_I}$, and (ii) for every concept/role name $\mathcal{S} \in \mathsf{N_C} \cup \mathsf{N_R}$ not in $\mathcal{F}$, $\mathcal{S}^{\mathcal{I}} = \mathcal{S}^{\mathcal{I}'}$.

Suppose that $\mathcal{I}$ is an interpretation that satisfies the premises of the rule, and $\mathcal{I}'$ is another interpretation that satisfies the conclusion of the rule. If we can prove that, for any interpretation $\mathcal{I}$, $\mathcal{I} \models \mathcal{C}$ iff $\mathcal{I}' \models \mathcal{P}$, for some interpretation $\mathcal{I}'$ A-equivalent to $\mathcal{I}$, where $\mathcal{P}$ denotes the premises and $\mathcal{C}$ denotes the conclusion, it is proved that the premises and the conclusion of the combination rule have the same logical consequences up to the remaining signature $\mathsf{sig}(\mathcal{N}) \backslash \{A\}$, since, except for A, any concept/role name in the premises and the conclusion are interpreted the same by $\mathcal{I}$ and $\mathcal{I}'$.

We prove soundness of the 16 combination cases in Figure 4. We first consider Case 1, where the premises are $B_h \sqcup A$ ($1 \le h \le l$) and $E_{h'} \sqcup \neg A$ ($1 \le h' \le l'$), and the conclusion is $B_h \sqcup E_{h'}$. The conclusion is obtained from the replacement of A in $B_h \sqcup A$ by $E_{h'}$ in $E_{h'} \sqcup \neg A$, or from the other perspective, is obtained from the replacement of A in $E_{h'} \sqcup \neg A$ by $\neg B_h$ in $B_h \sqcup A$. The conclusions in Cases 2, 3, 4, 5, 9 and 13 are obtained from the premises in a similar way by replacement, which, by observation, requires the positive premises to be in the form $B_h \sqcup A$ or the negative premises to be in the form $E_{h'} \sqcup \neg A$, where A occurs (either positively or negatively) at the top level of the clause. Although A does not explicitly occur at the top level of the premises in the other cases (Cases 6, 7, 8, 10, 11, 12, 14, 15 and 16), it occurs at the top level of the first-order representation of each premise (implicitly occurs at the top level of the DL representation of each premise). Therefore, these cases can be proved in the same way as with Cases 1, 2, 3, 4, 5, 9 and 13.

The idea of the replacement described above can be generalized as follows: Let $\mathcal{N}$ be a set of clauses in A-reduced form.

i. If the positive premises are $B_1 \sqcup A, \ldots, B_h \sqcup A$, the conclusion, which is obtained from the premises by replacing every occurrence of A in the negative premises by the concept $\neg B_1 \sqcup \ldots \sqcup \neg B_h$, is A-equivalent to the premises.
ii. If the negative premises are $E_1 \sqcup \neg A, \ldots, E_{h'} \sqcup \neg A$, the conclusion, which is obtained from the premises by replacing every occurrence of A in the positive premises by the concept $E_1 \sqcap \ldots \sqcap E_{h'}$, is A-equivalent to the premises.

By $\mathcal{N}^A_D$ we denote the clause set obtained from $\mathcal{N}$ by replacing every occurrence of A by the concept D. Now we prove that for any interpretation $\mathcal{I}$, $\mathcal{I} \models \mathcal{C}$ iff $\mathcal{I}' \models \mathcal{P}$, for some interpretation $\mathcal{I}'$ A-equivalent to $\mathcal{I}$. Recall that $\mathcal{P}^+(A)$ denotes the set of the positive premises and $\mathcal{P}^-(A)$ denotes the set of the negative premises.

We prove Case i. We first prove the "only if" direction. Assume

$$\mathcal{I} \models \mathcal{P}^-(A)^A_{\neg B_1 \sqcup \ldots \sqcup \neg B_h}.$$

Let $\mathcal{I}'$ be an interpretation extending $\mathcal{I}$ by additionally assigning A a subset of $\Delta^{\mathcal{I}}$ such that $A^{\mathcal{I}'} = (\neg B_1 \sqcup \ldots \sqcup \neg B_h)^{\mathcal{I}'}$. Intuitively, we have $(\neg B_1)^{\mathcal{I}'} \subseteq A^{\mathcal{I}'}, \ldots, (\neg B_h)^{\mathcal{I}'} \subseteq A^{\mathcal{I}'}$. Therefore,

$$\mathcal{I}' \models B_1 \sqcup A, \ldots, B_h \sqcup A \text{ and } \mathcal{I}' \models \mathcal{P}^-(A), B_1 \sqcup A, \ldots, B_h \sqcup A.$$

We then prove the "if" direction. Assume

$$\mathcal{I}' \models \mathcal{P}^-(A), B_1 \sqcup A, \ldots, B_h \sqcup A.$$

Then, we have $\mathcal{I}' \models B_1 \sqcup A, \ldots, B_h \sqcup A$ and $(\neg B_1 \sqcup \ldots \sqcup \neg B_h)^{\mathcal{I}'} \subseteq A^{\mathcal{I}'}$. Since $\mathcal{P}^-(A)$ is negative w.r.t. A, to order to show

$$\mathcal{I} \models \mathcal{P}^-(A)^A_{\neg B_1 \sqcup \ldots \sqcup \neg B_h},$$

we show the following monotonicity property:

$$(\mathcal{P}^-(A))^{\mathcal{I}'} \subseteq (\mathcal{P}^-(A)^A_{\neg B_1 \sqcup \ldots \sqcup \neg B_h})^{\mathcal{I}'}.$$

Equivalently, we show that for every clause α in $\mathcal{P}^-(A)$, we have

$$\alpha^{\mathcal{I}'} \subseteq (\alpha^A_{\neg B_1 \sqcup \ldots \sqcup \neg B_h})^{\mathcal{I}'}.$$

We prove this by induction, starting with the base case $\alpha = \neg A$ and proceeding to the induction hypothesis.

i. If $\alpha = \neg A$, then $\alpha^A_{\neg B_1 \sqcup \ldots \sqcup \neg B_h} = \neg(\neg B_1 \sqcup \ldots \sqcup \neg B_h)$. Since $(\neg B_1 \sqcup \ldots \sqcup \neg B_h)^{\mathcal{I}'} \subseteq A^{\mathcal{I}'}$, we have $\alpha^{\mathcal{I}'} \subseteq (\alpha^A_{\neg B_1 \sqcup \ldots \sqcup \neg B_h})^{\mathcal{I}'}$.

The induction hypothesis is that this statement holds for clauses of other forms.

ii. If $\alpha = \beta \sqcap \theta$, suppose

$$\beta^{\mathcal{I}'} \subseteq (\beta^A_{\neg B_1 \sqcup \ldots \sqcup \neg B_h})^{\mathcal{I}'} \text{ and } \theta^{\mathcal{I}'} \subseteq (\theta^A_{\neg B_1 \sqcup \ldots \sqcup \neg B_h})^{\mathcal{I}'}.$$

Directly, we have

$$(\beta \sqcap \theta)^{\mathcal{I}'} \subseteq ((\beta \sqcap \theta)^A_{\neg B_1 \sqcup \ldots \sqcup \neg B_h})^{\mathcal{I}'}.$$

Therefore, we have $\alpha^{\mathcal{I}'} \subseteq (\alpha^A_{\neg B_1 \sqcup \ldots \sqcup \neg B_h})^{\mathcal{I}'}$.

iii. If $\alpha = \beta \sqcup \theta$, suppose

$$\beta^{\mathcal{I}'} \subseteq (\beta^A_{\neg B_1 \sqcup \ldots \sqcup \neg B_h})^{\mathcal{I}'} \text{ and } \theta^{\mathcal{I}'} \subseteq (\theta^A_{\neg B_1 \sqcup \ldots \sqcup \neg B_h})^{\mathcal{I}'}.$$

Directly, we have

$$(\beta \sqcup \theta)^{\mathcal{I}'} \subseteq ((\beta \sqcup \theta)^A_{\neg B_1 \sqcup \ldots \sqcup \neg B_h})^{\mathcal{I}'}.$$

Therefore, we have $\alpha^{\mathcal{I}'} \subseteq (\alpha^A_{\neg B_1 \sqcup \ldots \sqcup \neg B_h})^{\mathcal{I}'}$.

iv. If $\alpha = \exists R.\beta$, suppose

$$\beta^{\mathcal{I}'} \subseteq (\beta^A_{\neg B_1 \sqcup \ldots \sqcup \neg B_h})^{\mathcal{I}'}.$$

Suppose there exists an element $d \in \Delta^{\mathcal{I}'}$ such that

$$d \notin (\exists R.\beta)^A_{\neg B_1 \sqcup \ldots \sqcup \neg B_h}{}^{\mathcal{I}'}.$$

This means that for every $y \in \Delta^{\mathcal{I}'}$, we have

$$(d, y) \notin R^{\mathcal{I}'} \text{ or } y \notin \beta^A_{\neg B_1 \sqcup \ldots \sqcup \neg B_h}{}^{\mathcal{I}'}.$$

Since $\exists R.\beta$ is true in $\mathcal{I}'$, for every $x \in \Delta^{\mathcal{I}'}$, there exists an element $d' \in \Delta^{\mathcal{I}'}$ such that

$$(x, d') \in R^{\mathcal{I}'} \text{ and } d' \in \beta^{\mathcal{I}'}.$$

Contradiction. Since $\beta^{\mathcal{I}'} \subseteq \beta^A_{\neg B_1 \sqcup \ldots \sqcup \neg B_n}{}^{\mathcal{I}'}$, we have $d' \in \beta^A_{\neg B_1 \sqcup \ldots \sqcup \neg B_n}{}^{\mathcal{I}'}$. **Contradiction**.

v. If $\alpha = \forall R.\beta$, suppose

$$\beta^{\mathcal{I}'} \subseteq (\beta^A_{\neg B_1 \sqcup \ldots \sqcup \neg B_h})^{\mathcal{I}'}.$$

Suppose there exists an element $d \in \Delta^{\mathcal{I}'}$ such that

$$d \notin (\forall R.\beta)^A_{\neg B_1 \sqcup \ldots \sqcup \neg B_h}{}^{\mathcal{I}'}.$$

This means that there exists an element $d' \in \Delta^{\mathcal{I}'}$ such that

$$(d, d') \in R^{\mathcal{I}'} \text{ and } d' \notin \beta^A_{\neg B_1 \sqcup \ldots \sqcup \neg B_h}{}^{\mathcal{I}'}.$$

Since $\forall R.\beta$ is true in $\mathcal{I}'$, for every $x, y \in \Delta^{\mathcal{I}'}$, we have

$$(x, y) \notin R' \text{ or } y \in \beta^{\mathcal{I}'}.$$

Contradiction. Since $\beta^{\mathcal{I}'} \subseteq \beta^A_{\neg B_1 \sqcup \ldots \sqcup \neg B_n}{}^{\mathcal{I}'}$, we have $d' \in \beta^A_{\neg B_1 \sqcup \ldots \sqcup \neg B_n}{}^{\mathcal{I}'}$. **Contradiction**.

Thus, the premises and the conclusion in Case i are A-equivalent. Case ii can be proved similarly. This indicates that the premises and the conclusion of the combination rule in Figure 4 have the same logical consequences in the remaining signature $\mathsf{sig}(\mathcal{N})\backslash\{A\}$. □

Lemma 6 means that the conclusion of the combination rule in Figure 4 is a solution of forgetting A from $\mathcal{N}$.

Theorem 1. *The calculus for concept elimination is sound.*

Theorem 1 follows from soundness of the A-reduced form transformation (Lemmas 2 - 3) and that of the combination rule for concept name elimination (Lemma 6).

In case definers were introduced during the forgetting process (to facilitate the transformation of $\mathcal{N}$ into a specific reduced form), they must be eliminated from the present $\mathcal{N}$, so that the computed forgetting solutions do not contain names other than those specified in the target signature. Our forgetting method eliminates definers using the calculus for concept name elimination once all concept and role names in the forgetting signature have been eliminated.

Example 4. Let $\mathcal{N} = \{1.C \sqcup \exists r.A, 2.\neg a \sqcup \exists s.\forall r.\neg A\}$. We consider the case of forgetting $\{A\}$ from $\mathcal{N}$. The first step is to transform $\mathcal{N}$ into A-reduced form: $\{1.C \sqcup \exists r.A, 2.\neg a \sqcup \exists s.\mathsf{D}, 3.\neg \mathsf{D} \sqcup \forall r.\neg A\}$, where D is a fresh definer. The second step is to apply the combination rule (Case 8) to Clauses 1 and 3 to eliminate A, yielding $\{1.C \sqcup \exists r.\top, 2.C \sqcup \neg \mathsf{D}, 3.\neg a \sqcup \exists s.\mathsf{D}\}$. The final step is to apply the combination rule (Case 5) to Clauses 2 and 3 to eliminate the introduced definer D, yielding the forgetting solution: $\{1.C \sqcup \exists r.\top, 2.\neg a \sqcup \exists s.C\}$.

B. Calculus for Role Elimination

The calculus for eliminating a role name r from a set $\mathcal{N}$ of clauses is based on another combination rule, shown in Figure 6. The combination rule is applicable to $\mathcal{N}$ to eliminate r iff $\mathcal{N}$ is in *r-reduced form*. Clauses in r-reduced form have 5 distinct forms, which, as with A-reduced form, can be classified into positive premises and negative premises, denoted respectively by $\mathcal{P}^+(r)$ and $\mathcal{P}^-(r)$; see Figure 7. By definition, $\mathcal{P}^+(r) = \mathcal{P}^+_\exists(r) \cup \mathcal{P}^+_{\exists,\_}(r) \cup \mathcal{P}^+_\mathcal{R}(r)$ and $\mathcal{P}^-(r) = \mathcal{P}^-_\forall(r) \cup \mathcal{P}^-_\mathcal{R}(r)$. By $\mathcal{N}^{-r}$ we denote the set of clauses not containing r, i.e., non r-clauses.

Positive Premises	Notation	Negative Premises	Notation
$C \sqcup \exists r.D$	$\mathcal{P}^+_\exists(r)$	$C \sqcup \forall r.D$	$\mathcal{P}^-_\forall(r)$
$C \sqcup \exists r^-.D$	$\mathcal{P}^+_{\exists,\_}(r)$		
$\neg S \sqcup r$	$\mathcal{P}^+_\mathcal{R}(r)$	$\neg r \sqcup S$	$\mathcal{P}^-_\mathcal{R}(r)$

Fig. 7: Distinct forms of clauses in r-reduced form

The idea of the combination rule for role name elimination is analogous to that for concept name elimination: to *combine* every positive premise with every negative one so as to derive from them all logical consequences not considering r. For distinct forms of positive and negative premises, there are in total $3 \times 2 = 6$ different cases of combination. The result of each combination is a finite set of clauses that do not contain r, denoted by $\mathsf{combine}(\alpha, \beta)$, where $\alpha \in \mathcal{P}^+(r)$ and $\beta \in \mathcal{P}^-(r)$. For space reasons, we represent the conclusion of the combination rule as union of the results obtained from combining all positive premises of the same form with all negative premises of the same form. The conclusion of the rule is the solution of forgetting r from $\mathcal{N}$.

Lemma 7. *The combination rule in Figure 6 is sound.*

Proof. That the premises and the conclusion of the combination rule have the same logical consequences in the remaining signature can be shown using the notion of $\mathcal{F}$-equivalence as described in the proof for Lemma 6. Observe that there are 6 combination cases in Figure 6, among which, Cases 2, 4, 5 and 6 satisfy the condition for the use of the replacement operation, and the first-order representations of Cases 1 and 3 satisfy the condition as well, though this is not made explicit with their DL representations. □

Lemma 7 means that the conclusion of the combination rule in Figure 6 is a solution of forgetting r from $\mathcal{N}$.

Theorem 2. *The calculus for role elimination is sound.*

Theorem 1 follows from soundness of the r-reduced form transformation (Lemmas 2 - 4) and that of the combination rule for role name elimination (Lemma 2).

Example 5. Let $\mathcal{N} = \{1.C \sqcup \exists r.A, 2.a \sqcup \exists s.\forall r.\neg A\}$. We consider the case of forgetting $\{r\}$ from $\mathcal{N}$. The first step is to transform $\mathcal{N}$ into r-reduced form $\mathcal{N} = \{1.C \sqcup \exists r.A, 2.a \sqcup \exists s.\mathsf{D}, 3.\neg\mathsf{D} \sqcup \forall r.\neg A\}$, where $\mathsf{D} \in \mathsf{N_D}$ is a fresh definer. The second step is to apply the combination rule for role name elimination to Clauses 1 and 3 to eliminate r, yielding an intermediate result $\mathcal{N}' = \{1.C \sqcup \neg\mathsf{D}, 2.a \sqcup \exists s.\mathsf{D}\}$. The final step is to apply the combination rule for concept name elimination (Case 4) to Clauses 1 and 2 in $\mathcal{N}'$ to eliminate the definer D, yielding a forgetting solution: $\{1.a \sqcup \exists s.C\}$.

C. The Forgetting Process

Our method can eliminate names in a goal-oriented manner. If a name has been successfully eliminated, it is immediately removed from $\mathcal{F}$, otherwise it remains in $\mathcal{F}$. The forgetting process in our method consists of the following phases, which are executed in sequence:

i. Transformation of the given $\mathcal{O}$ into a set $\mathcal{N}$ of clauses
ii. Elimination of the names in $\mathcal{F}$
iii. Elimination of the definers (if introduced)
iv. Transformation of the result $\mathcal{N}$ into an ontology $\mathcal{V}$

When eliminating definers, our method may introduce new definers. Definers cannot always be eliminated; the elimination may fail when the original ontology contains cyclic dependencies over the names in $\mathcal{F}$.[5] For example, the solution of forgetting A from $\mathcal{N} = \{A \sqsubseteq \exists r.A\}$ (cyclic over A) is $\mathcal{N}' = \{\mathsf{D}_1 \sqsubseteq \exists r.\mathsf{D}_1\}$, where $\mathsf{D}_1 \in \mathsf{N_D}$ is a fresh definer. Easy to see that eliminating D_1 from $\mathcal{N}'$ would yield $\mathcal{N}'' = \{\mathsf{D}_2 \sqsubseteq \exists r.\mathsf{D}_2\}$, which has the same form with $\mathcal{N}'$. Definers would thus be infinitely introduced, and the forgetting process would never terminate. Our method guarantees termination of forgetting by imposing the restriction that our method can only introduce fewer definers than existing ones; otherwise, our method terminates immediately.

Theorem 3. *Given an $\mathcal{ALCOIH}$-ontology $\mathcal{O}$ and a forgetting signature $\mathcal{F} \subseteq \mathsf{sig}(\mathcal{O})$, our method always terminates and returns an ontology $\mathcal{V}$. If $\mathcal{V}$ does not contain any definer names (if they are introduced during the forgetting process), then our method* succeeds *and $\mathcal{V}$ is a* solution *of forgetting $\mathcal{F}$ from $\mathcal{O}$.*

This follows from soundness of the calculi for concept and role elimination and that of the simplification rules.

V. Evaluation of the Forgetting Method

In order to evaluate the practicality of the forgetting method, we implemented a prototype of the method in Java using the OWL API (Version 3.4.7),[6] which is a Java API for creating, parsing, manipulating, and serializing OWL ontologies, and is released under the open source licenses LGPL[7] and Apache[8].

The source code and an executable .jar file of the prototype can be found at https://github.com/anonymous-ai-researcher/icde2020. Another way to try out the prototype is via the online platform http://www.forgettingshow.info/.

A. Test Data

TABLE I: Statistics of three Oxford-ISG snapshots

		Min	Max	Medium	Mean	90th percentile
I	$\lvert N_C\rvert$	0	1582	86	191	545
	$\lvert N_R\rvert$	0	332	10	29	80
	$\lvert Onto\rvert$	0	990	162	262	658
II	$\lvert N_C\rvert$	200	5877	1665	1769	2801
	$\lvert N_R\rvert$	0	887	11	34	61
	$\lvert Onto\rvert$	1008	4976	2282	2416	3937
III	$\lvert N_C\rvert$	1162	9809	4042	5067	8758
	$\lvert N_R\rvert$	1	158	4	23	158
	$\lvert Onto\rvert$	5112	9783	7277	7195	9179

The ontologies used for the evaluation were taken from the Oxford Ontology Library (Oxford-ISG).[9] Oxford-ISG contained the largest number of $\mathcal{ALCOIH}$-ontologies collected from multiple sources. In particular, Oxford-ISG contained 797 ontologies, and we took 494 of them with the number $|Onto|$ of axioms in the ontology not exceeding 10000. We

[5] A "cyclic dependency" means that a name is defined in terms of itself
[6] http://owlcs.github.io/owlapi/
[7] https://www.gnu.org/licenses/lgpl-3.0.html
[8] https://www.apache.org/licenses/LICENSE-2.0
[9] https://www.cs.ox.ac.uk/isg/ontologies/

then split the entire corpus of 494 ontologies into three sub-corpora: Corpus I with $10 \leq |Onto| \leq 1000$, containing 356 ontologies, Corpus II with $1000 \leq |Onto| \leq 5000$, containing 108 ontologies, and Corpus III with $5000 \leq |Onto| \leq 10000$, containing 26 ontologies. This gave a clearer insight into how LETHE and our prototype performed forgetting for ontologies of different sizes. Table I shows statistical information about the test ontologies, where $|N_C|$ and $|N_R|$ denote the average numbers of the concept and role names in the ontologies.

B. Settings

The evaluation was conducted for two settings: forgetting 10% and 30% of the concept and role names in the signature of each ontology. The results gave us insights into the practicality of the method for various real-world applications where the expected task was to forget a moderate or a large number of concept and role names (in line with the two settings). The names to be forgotten were randomly chosen. The experiments were run on a desktop with an Intel Core i7-9750H processor, four cores running at up to 2.60 GHz, and 16 GB of DDR4-1330 MHz RAM. The experiments were run 100 times on each ontology and we averaged the results in order to verify the accuracy of our findings.

To fit for real-world applications, the standard of forgetting success was set as: (1) forgetting all the terms in $\mathcal{F}$; (2) not introducing any extra expressivity outside of $\mathcal{ALC}$ in the forgetting solutions (definers); (3) finished in the given timeout; (4) finished in the given space limit. In this experiment, we limited the timeout to 1000 seconds and heap space to 8GB.

C. Evaluation on $\mathcal{ALCOIH}$-Ontologies

We evaluated the prototype on the $\mathcal{ALCOIH}$-fragments of the selected ontologies, obtained simply by removing from the ontologies those axioms not expressible in $\mathcal{ALCOIH}$.

TABLE II: Performance Results (Mem: memory consumption of successful cases, Success: success rate, TO: timeout rate, MO: out of memory rate, Extra: extra expressivity rate)

			Time (s)	Mem (MB)	Success	TO	MO	Extra
Prototype	0.1	I	6.9	56	96.7	0.9	0.0	2.4
		II	14.5	388	91.7	4.5	0.0	3.8
		III	4.1	331	90.5	6.9	0.0	2.6
	0.3	I	5.3	146	95.4	3.3	0.0	1.3
		II	22.5	311	87.5	12.5	0.0	0.0
		III	30.6	459	88.6	11.4	0.0	0.0

The results are shown in Table II. The most encouraging results are that: (i) our prototype succeeded in more than 92% of the test cases, and (ii) in more than 91% of these successful cases, the forgetting solutions were computed within only a few seconds. Part of the failures were due to the timeouts (see the "TO" column) while others due to definers being unable to be all eliminated (see the "Extra" column). There was no space explosion during the forgetting process (see the "MO" column), though a forgetting solution *can grow* exponentially in size with the given ontology in the worst cases. In fact, an observation from the results was that a forgetting solution was usually smaller than the original ontology, and the more terms were forgotten, the smaller the solution was. This was quite interesting, meaning that the worst cases occurred very rarely.

D. Comparison with LETHE on $\mathcal{ALCH}$

In order to find the best forgetting tools for ontologies, we compared our prototype with LETHE.[10] Since nominals and inverse roles were not supported by LETHE, only the $\mathcal{ALCH}$-fragments of the ontologies were used for the comparison.

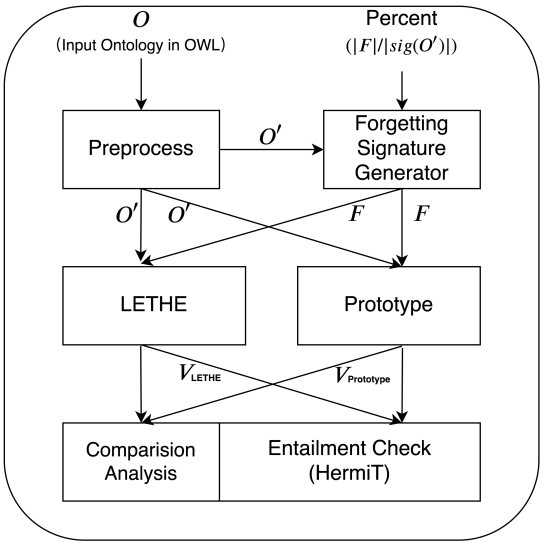

Fig. 8: Comparison Framework of LETHE and our prototype

Figure 8 depicts the overall comparison framework. Given $\mathcal{O}$ as the input ontology, an $\mathcal{ALCH}$-fragment $\mathcal{O}'$ was obtained via the preprocessing module. Based on $\mathsf{sig}(\mathcal{O}')$, forgetting signatures, which were randomly generated in the Forgetting Signature Generator module, were delivered into LETHE and our prototype together with $\mathcal{O}'$. $\mathcal{V}_{\text{LETHE}}$ and $\mathcal{V}_{\text{Prototype}}$ were the ontologies obtained from forgetting (not necessarily forgetting solutions) using respectively LETHE and our prototype. We used TestNG[11], one of the most powerful testing frameworks, for performance evaluation, and HermiT[12], one of the most reliable DL reasoners, for entailment relationship checking of the forgetting solutions computed by LETHE and our prototype. We explored the semantic relationships between the forgetting solutions computed by the two systems so as to check whether they were logically equivalent as desired.

According to the results in Table III, our prototype attained much better success rates than LETHE (93.4% for our prototype and 65.4% for LETHE on average) because of a lower timeout rate (TO-Rate). Apparently, our prototype was notably faster than LETHE; in particular, it was 27 times faster on average. This is partly because the elimination in LETHE is based on resolution, while our prototype uses both resolution and a generalization of a monotonicity property called Ackermann's Lemma for DLs (Cases 1, 2, 3, 4, 5,

[10]Here "best" means the system has better overall performance considering success rate, speed, and memory consumption.

[11]https://testng.org/doc/

[12]http://www.hermit-reasoner.com/

TABLE III: Performance Results of LETHE and our prototype

			Time (s)	Mem (MB)	Success	TO	MO	Extra
LETHE	0.1	I	19.7	199	85.4	7.1	0.0	7.3
		II	30.2	1101	66.7	31.8	0.0	1.5
		III	76.1	1881	65.4	25.6	0.0	5.1
	0.3	I	31.7	322	73.9	18.2	0.0	7.7
		II	71.4	868	51.2	47.5	0.0	1.2
		III	109.7	1083	50.0	46.2	0.0	5.0
Prototype	0.1	I	6.2	43	97.1	0.6	0.0	2.3
		II	13.3	349	92.4	4.0	0.0	3.6
		III	3.6	306	92.5	4.9	0.0	2.6
	0.3	I	4.8	106	95.8	3.1	0.0	1.1
		II	19.3	301	91.1	8.9	0.0	0.0
		III	26.9	424	91.3	8.7	0.0	0.0

9); the latter allows concept names to be eliminated more cheaply. In addition, we found that in almost 97% of the elimination rounds, a concept name could be eliminated using the generalized Ackermann's Lemma. Another important reason for such performance results is that our prototype introduces definers in a *conservative* manner (only when really necessary), while LETHE introduces them in a *systematic* and *exhaustive* manner, as is illustrated in the following example.

Example 6. Let $\mathcal{N} = \{C \sqcup \exists r.A, E \sqcup \forall r.\neg A\}$ and $\mathcal{F} = \{A\}$. Our method applies directly the combination rule (Case 8) to $\mathcal{N}$ to eliminate A, yielding the solution $\{C \sqcup \exists r.\top, C \sqcup E\}$. LETHE computes the same solution as our method does, but the derivation is more complicated, involving these steps:
Step 1: Normalization ($\mathsf{D}_1, \mathsf{D}_2 \in \mathsf{N_D}$):
$\{1.C \sqcup \exists r.\mathsf{D}_1, 2.\neg\mathsf{D}_1 \sqcup A, 3.E \sqcup \forall r.\mathsf{D}_2, 4.\neg\mathsf{D}_2 \sqcup \neg A\}$.
Step 2: Role propagation ($\mathsf{D}_3 \in \mathsf{N_D}$):
$\{5.C \sqcup E \sqcup \exists r.\mathsf{D}_3$ (from 1, 3), $6.\neg\mathsf{D}_3 \sqcup \mathsf{D}_1, 7.\neg\mathsf{D}_3 \sqcup \mathsf{D}_2\}$.
Step 3: Classical resolution:
$\{8.\neg\mathsf{D}_3 \sqcup A$ (2, 6), $9.\neg\mathsf{D}_3 \sqcup \neg A$ (4, 7), $10.\neg\mathsf{D}_3$ (8, 9)$\}$
Step 4: Existential role elimination: $\{11.C \sqcup E$ (5, 10)$\}$.
Step 5: At this point, $\mathcal{N}$ is saturated w.r.t. A, and no further inferences can be performed. LETHE removes all clauses that contain A; Clauses 2, 4, 8 and 9 are thus removed.
Step 6: Clause 5 is redundant because of 11. Clauses 6 and 7 are redundant because of 10. Hence, 5, 6 and 7 are removed.
Step 7: Only Clauses 1, 3, 10, 11 remain. LETHE eliminates the definers in Clauses 1, 3 and 10 by purification, yielding: $\{12.C \sqcup \exists r.\top, 13.E \sqcup \forall r.\top, 14.E \sqcup F\}$. As 13 is a tautology, the forgetting solution computed by LETHE is $\{12, 14\}$.

Our prototype had lower memory consumption during the forgetting process; see the "Mem" column. In particular, LETHE's memory consumption was about four times of our prototype. We believe that this could be attributed to the definer introduction mechanism employed by LETHE.

In principle, LETHE and our prototype should compute logically equivalent forgetting solutions for the same problems, which are the strongest entailment sets in the target signature, though their solutions may look different (having distinct representations). However, our experimental results showed that in 97.08% cases LETHE and our prototype computed logically equivalent forgetting solutions, while in 2.06% cases LETHE's solution entailed our prototype's solution but not the other way round, in 0.79% cases our prototype's solution entailed LETHE's solution but not the other way round, and in 0.07% cases they had no mutual entailment relationship. This is somewhat undesirable and thus needs further investigation.

VI. Case Study

We studied how our forgetting prototype performed in practice for computing views of ontologies in the case of ontology-based query answering. In particular, we used our forgetting prototype to compute views of the latest international version of the SNOMED CT ontology, and then queried each view and the base SNOMED CT with 100 artifact DL queries generated using Protégé,[13] a dominantly-used ontology editor. The purpose of this case study was to check: (i) if the view computed by our prototype returned the same answers as the original ontology, and (ii) if the average query response time was reduced by using a view of an ontology.

SNOMED CT is currently the most comprehensive, multilingual clinical healthcare terminology in the world. It covers a wide range of healthcare and biomedical topics. We choose SNOMED CT as our test data mainly for two reasons: (i) it is the largest publicly accessible ontology; the current version contains 335,000 axioms, 357,533 concept names, and 218 role names. This is the most suitable example to challenge our forgetting method. (ii) it is one of the most popular ontologies used in real-world applications; as of December 2019, it is the fifth most visited one among the 828 ontologies in BioPortal. This makes our case study with more practical value.

Topic of View	$\#\mathcal{F}_{\mathsf{C}}$	$\#\mathcal{F}_{\mathsf{R}}$	$\#\mathcal{V}$	ARTO	ARTV
Body Structure	305K	205	50K	31.8s	4.3s
Clinical Finding	317K	201	41K	30.2s	4.1s
Observable Entity	291K	204	57K	33.5s	5.6s
Organism	262K	169	87K	32.6s	5.9s
Physical Force	319K	211	41K	39.7s	4.3s
Procedure	216K	161	131K	30.9s	12.7s
Specimen	287K	197	65K	38.7s	5.4s
Substance	271K	199	74K	33.1s	5.6s

Fig. 9: Results of forgetting & querying on SNOMED CT

Views of SNOMED CT were computed based on 8 central topics of the ontology, namely *body structure*, *clinical finding*, *observable entity*, *organism*, *physical force*, *procedure*, *specimen* and *substance*. First we found the terms in each topic domain (the target signature). This was done by accessing the metrics of SNOMED CT.[14] Then we used the Sets.difference(set1, set2) method in Guava to identify the forgetting signature for each task, where set1 is the signature of SNOMED CT and set2 is the target signature. Finally we applied our prototype to compute the 8 views of SNOMED CT by eliminating the names in each forgetting signature.

The results are shown in Figure 9. Forgetting was successful in all cases. By $\#\mathcal{F}_{\mathsf{C}}$ and $\#\mathcal{F}_{\mathsf{R}}$ we denote the numbers of

[13] https://protege.stanford.edu/
[14] https://browser.ihtsdotools.org/?

concept names and role names in the forgetting signature. By $\#\mathcal{V}$ we denote the number of axioms in the computed view.

As desired, when using the generated queries to query a computed view and the original ontology from which the view was created, always, same answers were returned. The column headed "ARTO" shows the average response time when querying on the entire SNOMED CT and the column headed "ARTV" shows the average response time when querying the computed view. Apparently, querying the view took much shorter time than on the entire ontology. Usually, the more the terms were forgotten, the smaller the size of the computed view was, and the shorter the response time could be expected.

VII. Conclusion and Future Work

In this paper, we have developed a novel, practical forgetting method for creating views of $\mathcal{ALCOIH}$-ontologies. The method is terminating and sound. It is the first approach to forgetting in the description logic $\mathcal{ALCOIH}$, a frequently-used language but not considered for forgetting before. An empirical comparison of a prototype of the method has shown notable better success rates and a significant speed-up over the peer LETHE system on the $\mathcal{ALCH}$-fragments of a large number of Oxford-SIG ontologies. A case study on the ontology-based query answering problem has verified the usefulness of forgetting for real-world applications.

Previous work has been largely focused on forgetting concept and role names, while there has been little attention paid to the problem of nominal elimination. This considerably restricts the applicability of forgetting for many real-world applications such as information hiding and privacy protection, where nominals are extensively present. Our immediate step for future work is to develop a forgetting method able to eliminate not only concept names and role names, but also nominals in expressive description logics.

References

[1] W. Ackermann. Untersuchungen über das Eliminationsproblem der mathematischen Logik. *Mathematische Annalen*, 110(1):390–413, 1935.

[2] M. Ashburner. Gene ontology: Tool for the unification of biology. *Nature Genetics*, 25:25–29, 2000.

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

[4] L. Bachmair, H. Ganzinger, D. A. McAllester, and C. Lynch. Resolution theorem proving. In Robinson and Voronkov [16], pages 19–99.

[5] C. G. Fermüller, A. Leitsch, U. Hustadt, and T. Tammet. Resolution decision procedures. In Robinson and Voronkov [16], pages 1791–1849.

[6] B. C. Grau and B. Motik. Reasoning over ontologies with hidden content: The import-by-query approach. *J. Artif. Intell. Res.*, 45:197–255, 2012.

[7] M. Hepp, P. D. Leenheer, A. de Moor, and Y. Sure, editors. *Ontology Management, Semantic Web, Semantic Web Services, and Business Applications*, volume 7 of *Semantic Web and Beyond: Computing for Human Experience*. Springer, 2008.

[8] P. Koopmann. *Practical Uniform Interpolation for Expressive Description Logics*. PhD thesis, The University of Manchester, UK, 2015.

[9] P. Koopmann and R. A. Schmidt. Uniform Interpolation of $\mathcal{ALC}$-Ontologies Using Fixpoints. In *Proc. FroCos'13*, volume 8152 of *LNCS*, pages 87–102. Springer, 2013.

[10] M. Ludwig and B. Konev. Practical Uniform Interpolation and Forgetting for $\mathcal{ALC}$ TBoxes with Applications to Logical Difference. In *Proc. KR'14*. AAAI Press, 2014.

[11] C. Lutz and F. Wolter. Foundations for Uniform Interpolation and Forgetting in Expressive Description Logics. In *Proc. IJCAI'11*, pages 989–995. IJCAI/AAAI Press, 2011.

[12] M. A. Musen. The protégé project: a look back and a look forward. *AI Matters*, 1(4):4–12, 2015.

[13] N. F. Noy and M. A. Musen. PROMPT: Algorithm and Tool for Automated Ontology Merging and Alignment. In *Proc. AAAI/IAAI'00*, pages 450–455. AAAI Press/The MIT Press, 2000.

[14] N. F. Noy and M. A. Musen. Ontology versioning in an ontology management framework. *IEEE Intelligent Systems*, 19(4):6–13, 2004.

[15] M. M. Ribeiro and R. Wassermann. Base revision for ontology debugging. *J. Log. Comput.*, 19(5):721–743, 2009.

[16] J. A. Robinson and A. Voronkov, editors. *Handbook of Automated Reasoning (in 2 volumes)*. Elsevier and MIT Press, 2001.

[17] R. A. Schmidt and D. Tishkovsky. Using tableau to decide description logics with full role negation and identity. *ACM Trans. Comput. Log.*, 15(1):7:1–7:31, 2014.

[18] K. A. Spackman. SNOMED RT and SNOMED CT. promise of an international clinical ontology. *M.D. Computing 17*, 2000.

[19] H. Stuckenschmidt, C. Parent, and S. Spaccapietra, editors. *Modular Ontologies: Concepts, Theories and Techniques for Knowledge Modularization*, volume 5445 of *LNCS*. Springer, 2009.

[20] N. Troquard, R. Confalonieri, P. Galliani, R. Peñaloza, D. Porello, and O. Kutz. Repairing Ontologies via Axiom Weakening. In *Proc. AAAI'18*, pages 1981–1988. AAAI Press, 2018.

[21] K. Wang, G. Antoniou, R. W. Topor, and A. Sattar. Merging and aligning ontologies in DL-programs. In *RuleML*, volume 3791 of *LNCS*, pages 160–171. Springer, 2005.

[22] Y. Zhao, G. Alghamdi, R. A. Schmidt, H. Feng, G. Stoilos, D. Juric, and M. Khodadadi. Tracking Logical Difference in Large-Scale Ontologies: A Forgetting-Based Approach. In *Proc. AAAI'19*, pages 3116–3124. AAAI Press, 2019.

[23] Y. Zhao and R. A. Schmidt. Forgetting Concept and Role Symbols in $\mathcal{ALCOIH}\mu^+(\triangledown, \sqcap)$-Ontologies. In *Proc. IJCAI'16*, pages 1345–1352. IJCAI/AAAI Press, 2016.

[24] Y. Zhao and R. A. Schmidt. FAME: An Automated Tool for Semantic Forgetting in Expressive Description Logics. In *Proc. IJCAR'18*, volume 10900 of *LNCS*, pages 19–27. Springer, 2018.