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

Anonymous Author(s)

ABSTRACT

This paper focuses on the problem of computing views of ontologies using a forgetting-based approach. In traditional 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* $\mathsf{sig}(\mathcal{O})$ of $\mathcal{O}$ is the set of all 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 preserving the original meanings of the terms in the target signature. Computing views of ontologies is useful (i) for applications such as ontology-based query answering, in the sense that the view can be used as a substitute of the original ontology to answer queries formulated with the target signature, and (ii) for security purposes, in the sense that it restricts users from viewing certain information of an ontology.

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 target signature. Forgetting can thus be used as an ontology engineering tool to compute views of ontologies — the solution of forgetting a set $\mathcal{F}$ of terms from an ontology $\mathcal{O}$ is the view $\mathcal{V}$ of $\mathcal{O}$ for the target signature $\mathsf{sig}(\mathcal{O})\backslash\mathcal{F}$.

In this paper, we present a novel, practical forgetting method for computing views of ontologies specified in the description logic $\mathcal{ALCOIH}$, the basic $\mathcal{ALC}$ extended with nominals, inverse roles, and role inclusions. The method is terminating and sound, and it can eliminate concept and role names from $\mathcal{ALCOIH}$-ontologies. An evaluation with a prototypical implementation on a corpus of real-world ontologies has shown superb results. A case study of the prototype on the ontology-based query answering problem has verified the effectiveness of the forgetting technique for ontology-based knowledge processing, and the practicality of our forgetting method for real-world and industry use.

1 INTRODUCTION

1.1 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 knowledge or discourse. The representational primitives include classes (*concepts*), class members (*individuals*), attributes (*properties*), and relationships (*relations* among 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, their properties, and their relations to other concepts and individuals. In ontologies, properties and relationships are collectively called *roles*; concepts, roles, and individuals are collectively called *terms*. The *signature* (*vocabulary*) of an ontology is the set of all the terms in the ontology.

Ontologies are typically specified 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 specified 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 for the Semantic Web. 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 engineering and related tasks.

Given an application domain, we use *concept names*, *role names*, and *individual names* (aka *nominals*), 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 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:

{COMP60332} ⊑ Course

{Phelps} ⊑ Person ⊓ ∃participatesIn.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:

{John} ⊑ ∃teaches.{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 statements and ABox statements make up a *knowledge base*. One can think of a knowledge base as a traditional 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, their properties and relationships. In ontologies, TBox (ABox) statements are called *TBox (ABox) axioms*. In the remainder of this paper, the terms *ontology* and *knowledge base* are used interchangeably.

DLs have many variants which differ in expressivity, depending upon which building blocks and logical constructors are set to be used to describe domain knowledge. The basic DL is that of $\mathcal{ALC}$, which uses concept names, role names, and the logical constructors of ¬, ⊓, ⊔, ∃, and ∀ 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; see [3] for examples and a formal proof of this. 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. This follows from the decidability of $\mathcal{ALBO}^{\mathsf{id}}$ [28], which subsumes $\mathcal{ALCOIH}$.

1.2 Why Computing Views of Ontologies?

With the growing utilization of ontologies in real-world scenarios, 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, modeling 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. To make it clearer, this means that, new ontologies can be generated from existing ones, and they are not necessarily scratch-developed; the latter is costly and error-prone. Computing views of ontologies is one of 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 ones.

i. **Ontology summarization**: Comprehending a complex ontology can be hindered by a large or heterogeneous vocabulary. If the central terms of the ontology are known, one could gain a more focused high-level summary of the ontology by computing a view of the ontology with the central terms being selected as the target signature [20].
ii. **Ontology-based query answering**: Taking ontological knowledge into account when retrieving data from relational databases has been widely acknowledged. It is however found in many cases [7] that querying a large knowledge base often involves extensive reasoning, which, due to its high computational complexity, 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 [31], has been found deficient in the sense that the view computed by the approach often involves a large amount of irrelevant information. Querying such a view still takes a long time to reason about and requires much memory.
iii. **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 upon 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.

Computing view of ontologies is useful for many other ontology engineering tasks such as ontology alignment and merging [17, 24, 33], versioning [8, 9, 25, 29], debugging and repair [26, 32], and logical difference computation [11, 12, 18, 34]. For space reasons, we do not elaborate them all in this paper.

1.3 Basics of Forgetting

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. It is not difficult to see that the intention of forgetting is perfectly

fit for the task of computing views of ontologies, i.e., the ontology obtained from forgetting, namely the *forgetting solution*, is a view of the original ontology for the target signature, which corresponds to the remaining signature in forgetting; we will elaborate this in the next section when formally defining the notion of forgetting.

Forgetting is an inherently difficult problem; it is much harder than standard reasoning (satisfiability testing), and very few logics are known to be complete for forgetting.[1] Foundational studies have shown that: (i) forgetting solutions do not always exist for $\mathcal{EL}$- or $\mathcal{ALC}$ [10, 11, 20], (ii) deciding the existence of forgetting solutions is ExpTime-complete for $\mathcal{EL}$ [19] and 2ExpTime-complete for $\mathcal{ALC}$ [20], and (iii) forgetting solutions can be triple exponential in size w.r.t. the input ontologies for $\mathcal{EL}$ and $\mathcal{ALC}$ [20, 23].

Although forgetting is a challenging problem, there is however general consensus on its tremendous potential for ontology-based knowledge processing, and there has been continuous efforts dedicated into the development and automation 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 include LETHE [13], NUI [12], and FAME [36]. LETHE is based on the classic inference system of *resolution* [4, 5]; it can eliminate concept and role names from $\mathcal{ALCH}$-ontologies. NUI is another resolution-based method but only able to eliminate concept and role names from $\mathcal{ELH}$-ontologies, a lightweight DL less expressive than $\mathcal{ALCH}$. FAME is based on a monotonicity property called *Ackermann's Lemma* [1]; the method can eliminate concept and role names from $\mathcal{ALCOIH}$-ontologies.

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 use. NUI's drawback seems more severe: the results computed by NUI are only approximations of the real forgetting solutions, and thus are weaker than the real ones. This means that there are axioms formulated using the terms in the target signature such that they can only be entailed by the original ontology, but not the resultant one computed by NUI, i.e., the original ontology and the resultant one do not have the same logical consequences in the target signature, which does not strictly follow the definition of forgetting. In the case of ontology-based query answering, this means that the original ontology and the view computed by NUI may give different answers to a same target-signature-formulated query. FAME is fast, and is able to handle large-scale ontologies, but it introduces extra expressivity, namely fresh *nominals* and *inverse roles* during the forgetting process, and such extra expressivity cannot be eliminated from the resultant ontology and will be present in the forgetting solutions. This is because FAME computes solutions of semantic forgetting [35], a stronger notion of forgetting than the notion used in this paper. This is not satisfactory for users like SNOMED CT [30] who do not have the flexibility to easily switch to a more expressive language, or are bound by the application, the available support and tooling, to a specific language. Hence, there have been no practical methods and automated tools available so far for forgetting with large-scale ontologies, without the need of extending the target language with extra expressivity.

1.4 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 specified in the description logic $\mathcal{ALCOIH}$, the basic $\mathcal{ALC}$ extended with nominals, inverse roles, and role inclusions. Being based on a set of inference rules, the method can eliminate concept and role names from $\mathcal{ALCOIH}$-ontologies. 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, a comprehensive empirical evaluation with a prototype implementation showed superb results on a large corpus of real-world ontologies taken from NCBO BioPortal [21]: (i) the method succeeded (i.e., eliminated all concept and role names in the forgetting signature), in more than 94% 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 respectively LETHE and FAME, the existing tools of forgetting,[2] with the results showing that: (i) compared to LETHE on the $\mathcal{ALCH}$-fragments of the test BioPortal ontologies,[3] our prototype was 23 times faster on average, and attained better success rates by 22.5% when a timeout of 1000 seconds was used, (ii) compared to FAME on the $\mathcal{ALCOIH}$-fragments, similar speed was observed, but our prototype fired considerably better success rates. To gain an insight into the practical applicability of the forgetting method, we 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 [30], and then retrieved data from both the original ontology and each view with a set of 100 artifact DL queries generated using Protégé [22]. 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 is, and the shorter the response time can be expected.

To sum up, the *contribution* of this paper is a novel framework for computing views of ontologies. The framework is mainly based on a reasoning procedure called forgetting, for which we have accordingly developed a practical method in the setting of $\mathcal{ALCOIH}$-ontologies. Compared to existing forgetting methods, our method

[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.

[2] we did not consider a comparison of the prototype with NUI, because NUI's results are mostly approximations of the real forgetting solutions, and do not meet the needs of most realistic applications such as ontology-based query answering.

[3] LETHE can also handle $\mathcal{SIF}$- and $\mathcal{SHQ}$-ontologies [15, 16], but is only able to eliminate concept names from them, but not role names.

performs much better in terms of efficiency and success rates. This is extremely useful from the perspective of ontology engineering, as it provides ontology curators with a powerful tool for producing views of ontologies. Moreover, our method does not require the target language to be further extended to represent the forgetting solutions. This is particularly useful for tasks like ontology merging, alignment, ontology-based query answering, etc., where the source language and the target language must coincide.

2 PRELIMINARIES

2.1 $\mathcal{ALCOIH}$-Ontologies

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 $\mathcal{I} \models \mathcal{O}$.

2.2 Forgetting for $\mathcal{ALCOIH}$-Ontologies

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 a set of axioms (ontologies). We define $\mathsf{sig}(X) = \mathsf{sig_C}(X) \cup \mathsf{sig_R}(X)$.

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 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, i.e., if both $\mathcal{V}$ and $\mathcal{V}'$ are views of $\mathcal{O}$ for a target signature, then they are logically equivalent, though their representations may not be identical.

3 ONTOLOGY NORMALIZATION

3.1 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

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:

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

Figure 1: Transformations into clausal normal form

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, the notation $\mathcal{N}$ is uniformly used to denote an $\mathcal{ALCOIH}$-ontology in clausal normal form, i.e., a set $\mathcal{N}$ of clauses.

Lemma 1. *By incrementally applying the standard transformations in Figure 1, any $\mathcal{ALCOIH}$-ontology $\mathcal{O}$ can be transformed into an equivalent one in clausal normal form, i.e., a set $\mathcal{N}$ of clauses.*

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′. $\exists r.\neg A \sqcup \exists s.A \sqcup \forall t.A$

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 $\neg r \sqcup s$.

3.2 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 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.

Note that not every A-clause or r-clause is initially in A-reduced form or r-reduced form. TBox A-clauses not in A-reduced form could have the form $C \sqcup \exists r.D$, $C \sqcup \exists r^-.D$, $C \sqcup \forall r.D$ or $C \sqcup \forall r^-.D$, where $r \in \mathsf{N_R}$, C is a clause that contains A, and D is a concept that contains A. TBox r-clauses not in r-reduced form have the form $C \sqcup \exists S.D$ or $C \sqcup \forall S.D$, where S is any role, C is a clause and D is a concept such that at least one of them contain r. RBox r-clauses not in r-reduced form have the form $\neg S \sqcup r^-$ or $\neg r^- \sqcup S$, where S is a role that does not contain r.[4]

TBox clauses not in 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 clauses not in 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 reduced form [14]: 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

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

A-reduced form, until "C" and "D" do not contain A, and in every TBox r-clause not in r-reduced form, until "C" and "D" do not contain r. Also, 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.

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 [13], 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, thereby 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 unique feature of our method – definers are desired not to be present 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.

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

Figure 2: The surfacing rule

Note that A-reduced form does not include the forms $C \sqcup \forall r^{-}.A$ and $C \sqcup \forall r^{-}.\neg A$, where A occurs (either positively or negatively) 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. □

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

Figure 3: The inverting rules

Likewise, the form of $C \sqcup \forall r^-.D$ is not included in the r-reduced form because of the surfacing rule. The forms of $\neg S \sqcup r^-$ 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.

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 $(d, d') \notin (\neg S^- \sqcup r)^{\mathcal{I}}$.

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

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 are introduced before the application of the surfacing rule. Since concept names are never present in RBox clauses, 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. Therefore, the normalization seems against the intention of forgetting, but we will show in the later 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.*

Lemmas 1 and 5 guarantee completeness of the ontology normalization: any $\mathcal{ALCOIH}$-ontology $\mathcal{O}$, from its initial state, can be transformed into A-reduced form, where A is any concept name in $\mathcal{O}$, or into r-reduced form, where r is any role name in $\mathcal{O}$.

4 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 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. Both 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.

4.1 Calculus for Concept Name 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* of the combination rule, 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, $\mathcal{P}^+(A) = \mathcal{P}^+_{\mathcal{U}}(A) \cup \mathcal{P}^+_{\exists}(A) \cup \mathcal{P}^+_{\exists,-}(A) \cup \mathcal{P}^+_{\forall}(A)$ and $\mathcal{P}^-(A) = \mathcal{P}^-_{\mathcal{U}}(A) \cup \mathcal{P}^-_{\exists}(A) \cup \mathcal{P}^-_{\exists,-}(A) \cup \mathcal{P}^-_{\forall}(A)$. By $\mathcal{N}^{-A}$ we denote the set of clauses not containing A, i.e., non A-clauses.

The basic 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 with the original

$$\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 combination rule ($1 \le h \le l, 1 \le i \le m, 1 \le j \le n, 1 \le k \le o, 1 \le h' \le l', 1 \le i' \le m', 1 \le j' \le n', 1 \le k' \le o'$):
$B_h, C_i, D_j, \phi_k, E_{h'}, F_{i'}, G_{j'}$ 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: $\mathsf{combine}(\mathcal{P}^+_{\mathfrak{U}}(\mathsf{A}), \mathcal{P}^-_{\mathfrak{U}}(\mathsf{A})) = \bigcup_{1\le h\le l}\bigcup_{1\le h'\le l'} \{B_h \sqcup E_{h'}\}$ **2:** $\mathsf{combine}(\mathcal{P}^+_{\mathfrak{U}}(\mathsf{A}), \mathcal{P}^-_{\exists}(\mathsf{A})) = \bigcup_{1\le h\le l}\bigcup_{1\le i'\le m'} \{F_{i'} \sqcup \exists u_{i'}.B_h\}$

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

5: $\mathsf{combine}(\mathcal{P}^+_{\exists}(\mathsf{A}), \mathcal{P}^-_{\mathfrak{U}}(\mathsf{A})) = \bigcup_{1\le i\le m}\bigcup_{1\le h'\le l'} \{C_i \sqcup \exists r_i.E_{h'}\}$ **6:** $\mathsf{combine}(\mathcal{P}^+_{\exists}(\mathsf{A}), \mathcal{P}^-_{\exists}(\mathsf{A})) = \bigcup_{1\le i\le m}\bigcup_{1\le i'\le m'} \{C_i \sqcup \exists r_i.\top, F_{i'} \sqcup \exists u_{i'}.\top\}$

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

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

9: $\mathsf{combine}(\mathcal{P}^+_{\exists,-}(\mathsf{A}), \mathcal{P}^-_{\mathfrak{U}}(\mathsf{A})) = \bigcup_{1\le j\le n}\bigcup_{1\le h'\le l'} \{D_j \sqcup \exists s_j^-.E_{h'}\}$ **10:** $\mathsf{combine}(\mathcal{P}^+_{\exists,-}(\mathsf{A}), \mathcal{P}^-_{\exists}(\mathsf{A})) = \bigcup_{1\le j\le n}\bigcup_{1\le i'\le m'} \{D_j \sqcup \exists s_j^-.\top, F_{i'} \sqcup \exists u_{i'}.\top\}$

11: $\mathsf{combine}(\mathcal{P}^+_{\exists,-}(\mathsf{A}), \mathcal{P}^-_{\exists,-}(\mathsf{A})) = \bigcup_{1\le j\le n}\bigcup_{1\le j'\le n'} \{D_j \sqcup \exists s_j^-.\top, G_{j'} \sqcup \exists v_{j'}^-.\top\}$ **12:** $\mathsf{combine}(\mathcal{P}^+_{\exists,-}(\mathsf{A}), \mathcal{P}^-_{\forall}(\mathsf{A})) = \bigcup_{1\le j\le n}\bigcup_{1\le k'\le o'} \{D_j \sqcup \exists s_j^-.\top\}$

13: $\mathsf{combine}(\mathcal{P}^+_{\forall}(\mathsf{A}), \mathcal{P}^-_{\mathfrak{U}}(\mathsf{A})) = \bigcup_{1\le k\le o}\bigcup_{1\le h'\le l'} \{\phi_k \sqcup \forall t_k.E_{h'}\}$

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

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

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

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

Figure 5: Distinct forms of clauses in A-reduced form

two premises in the remaining signature. Since every premise set like $\mathcal{P}^+_{\mathfrak{U}}(A)$ 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 $\mathcal{P}^+_{\mathfrak{U}}(A)$ with every one in $\mathcal{P}^-_{\mathfrak{U}}(A)$ is the union $\mathsf{combine}(\mathcal{P}^+_{\mathfrak{U}}(A), \mathcal{P}^-_{\mathfrak{U}}(A))$ 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 $\mathsf{combine}(\mathcal{P}^+_{\mathfrak{U}}(A), \mathcal{P}^-_{\mathfrak{U}}(A))$. For space reasons, we represent the conclusion of the combination rule "at the set level".

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

Lemma 6 says that the conclusion and the premises of the combination rule for concept name elimination have the same logical consequences in the remaining signature $\mathsf{sig}(\mathcal{N})\backslash\{A\}$, which means that the conclusion of the rule is a solution of forgetting A from $\mathcal{N}$.

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

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

In case definers were introduced during the forgetting process (to help with the transformation of $\mathcal{N}$ into a specific reduced form), they must be eliminated from $\mathcal{N}$, so that the computed forgetting solutions do not contain names other than those specified in the target signature. Our forgetting method eliminates definers using

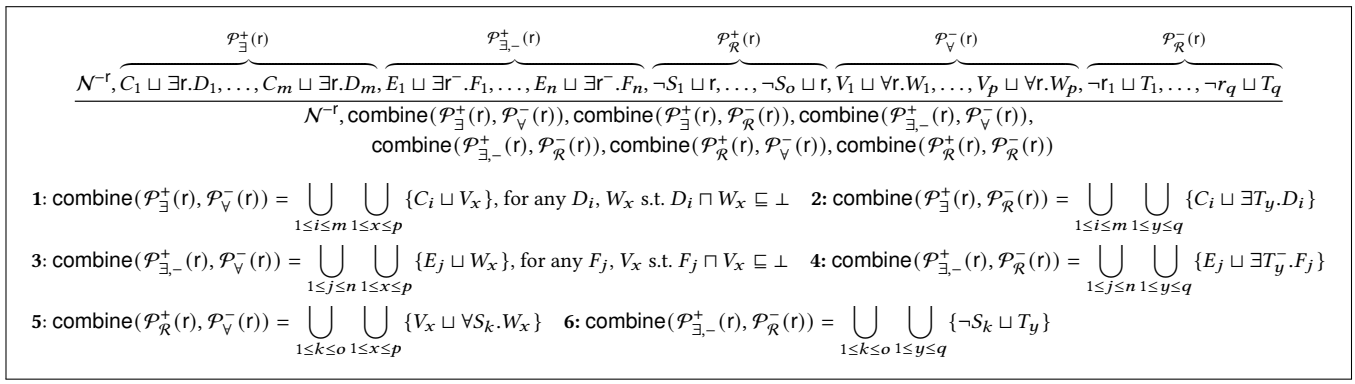

Figure 6: The combination rule for eliminating a role name $\mathsf{r} \in \mathsf{sig_R}(\mathcal{N})$ from a set $\mathcal{N}$ of clauses in r-reduced form

the calculus for concept name elimination once all concept and role names in the forgetting signature have been eliminated.

Example 3. 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 4) to Clauses 2 and 3 to eliminate the introduced definer D, yielding the forgetting solution: $\{1.C \sqcup \exists r.\top, 2.\neg a \sqcup \exists s.C\}$.

4.2 Calculus for Role Name 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
$C \sqcup \exists r.D$	$\mathcal{P}^+_\exists(r)$		$C \sqcup \forall r.D$	$\mathcal{P}^-_\forall(r)$
$C \sqcup \exists r^-.D$	$\mathcal{P}^+_{\exists,-}(r)$			
$\neg S \sqcup r$	$\mathcal{P}^+_\mathcal{R}(r)$		$\neg r \sqcup S$	$\mathcal{P}^-_\mathcal{R}(r)$

Figure 7: Distinct forms of clauses in r-reduced form

The basic 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 that are irrelevant to 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 $\beta \in \mathcal{P}^-(r)$. In order to save space, 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.*

This states that the conclusion and the premises of the combination rule for role name elimination has the same logical consequences in the remaining signature $\mathsf{sig}(\mathcal{N}) \backslash r$.

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

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

Example 4. 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: $\{1.a \sqcup \exists s.C\}$.

4.3 The Forgetting Process

The input to our forgetting method are an $\mathcal{ALCOIH}$-ontology $\mathcal{O}$ and a set $\mathcal{F} \in \mathsf{sig}(\mathcal{O})$ of concept and role names to be forgotten (the forgetting signature). $\mathcal{O}$ must be specified as an OWL/XML file, or a URL pointing to an OWL/XML file. If $\mathcal{O}$ is not expressible in $\mathcal{ALCOIH}$, only the $\mathcal{ALCOIH}$-fragment is taken. This is done simply by replacing the concepts not expressible in $\mathcal{ALCOIH}$ by $\top$. Which names are to be forgotten is determined by the user and their application demands. Our method can eliminate names in any user-specified order. 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 $\mathcal{V}$

(i) and (iv) are done using the standard transformations. (ii) is an iteration of several rounds in which the concept and role names

in $\mathcal{F}$ are eliminated using the calculi for concept name and role name elimination, respectively. (iii) is an iteration of several rounds in which the introduced definers are eliminated using the calculus for concept name elimination. 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 $\mathcal{N} = \{\neg A \sqcup \exists r.A\}$ (cyclic over A) is $\mathcal{N}' = \{\neg \mathsf{D}_1 \sqcup \exists r.\mathsf{D}_1\}$, where $\mathsf{D}_1 \in \mathsf{N}_\mathsf{D}$ is a fresh definer. Easy to see that eliminating D_1 from $\mathcal{N}'$ would yield $\mathcal{N}'' = \{\neg \mathsf{D}_2 \sqcup \exists r.\mathsf{D}_2\}$, 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.

A set of *equivalence-preserving* simplification rules are used during the forgetting process; they are applied as eagerly as possible. This ensures that (i) the clauses in $\mathcal{N}$ are always simpler representations, and (ii) no redundant clauses are present in $\mathcal{N}$. This improves the efficiency of our forgetting method.

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 names in $\mathcal{F}$ or 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 the soundness of the calculi for concept and role elimination and that of the simplification rules.

5 EVALUATION OF THE METHOD

In order to gain an insight into the effectiveness of the method, we implemented a prototype in Java using the OWL API Version 3.5 [6], and evaluated it on a corpus of real-world ontologies taken from the NCBO BioPortal repository.[7] The corpus was created based on a snapshot of the repository released in March 2017 [21], which included 396 OWL API compatible ontologies.

Metrics	Max.	Min.	Mean	Median	90th Percentile
$\#(\mathcal{O})$	1.8M	100	4.6K	1.1K	12.6K
$\#\mathsf{sig}_\mathsf{C}(\mathcal{O})$	847.8K	36	2.1K	502	5.6K
$\#\mathsf{sig}_\mathsf{R}(\mathcal{O})$	1.4K	0	54	12	144
$\#\mathsf{sig}_\mathsf{I}(\mathcal{O})$	87.9K	0	216	0	206

Figure 8: Statistical information about the test ontologies

Figure 8 shows statistical information about the test ontologies. By $\#(\mathcal{O})$, $\#\mathsf{sig}_\mathsf{C}(\mathcal{O})$, $\#\mathsf{sig}_\mathsf{R}(\mathcal{O})$, and $\#\mathsf{sig}_\mathsf{I}(\mathcal{O})$ we denote the average numbers of respectively the axioms, concept names, role names and individual names in the test ontologies.

We compared the results computed by our method with those by LETHE and FAME, respectively. Since nominals and inverse roles were not supported by LETHE, only the $\mathcal{ALCH}$-fragments of the test ontologies were used for the comparison with LETHE. But for the comparison with FAME, the $\mathcal{ALCOIH}$-fragments were used, because FAME fully accommodated $\mathcal{ALCOIH}$. The names to be forgotten were randomly chosen. The experiments were run on a desktop with an Intel Core i7-4790 processor, four cores running at up to 3.60 GHz, and 8 GB of DDR3-1600 MHz RAM. A timeout of 1000 seconds was imposed on each run.

Tool	$\mathcal{F}$%	Time	Timeouts	Success Rates	Extra Exp.
Prototype $\mathcal{ALCH}$	10%	1.3s	0.8%	98.9%	0.3%
	40%	3.4s	2.5%	95.6%	1.9%
	70%	6.5s	5.2%	90.7%	4.1%
	Avg.	**3.7s**	2.8%	**95.1%**	2.1%
Lethe $\mathcal{ALCH}$	10%	27.6s	6.1%	88.5%	5.4%
	40%	87.5s	21.3%	65.3%	13.4%
	70%	141.9s	35.7%	42.8%	21.5%
	Avg.	**85.7s**	21.0%	**72.6%**	13.4%

Figure 9: Comparison of our method & Lethe on $\mathcal{ALCH}$

Tool	$\mathcal{F}$%	Time	Timeouts	Success Rates	Extra Exp.
Prototype $\mathcal{ALCOIH}$	10%	1.5s	1.1%	98.1%	0.6%
	40%	3.9s	2.8%	94.9%	2.1%
	70%	7.9s	5.2%	90.1%	4.7%
	Avg.	**4.5s**	3.1%	**94.4%**	2.5%
Fame $\mathcal{ALCOIH}$	10%	1.3s	2.0%	86.6%	11.4%
	40%	4.4s	4.0%	64.6%	31.4%
	70%	11.1s	7.1%	37.3%	55.6%
	Avg.	5.6s	4.4%	**62.8%**	32.8%

Figure 10: Comparison of our method & Fame on $\mathcal{ALCOIH}$

The comparison was conducted for three settings: forgetting 10%, 40%, 70% of the concept and role names in the signature of each ontology. The results gave us insights into the effectiveness of each method for various real-world applications where the expected task was to forget a small, a moderate or a large number of concept and role names (in line with the three settings).

The results are shown in Figures 9 and 10. The most encouraging results are that: (i) our prototype succeeded in more than 94% 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 while others due to definers being unable to be all eliminated.

According to the results in Figure 9, our prototype was notably faster than LETHE on the $\mathcal{ALCH}$-fragments; in particular, it was 23 times faster on average. This was mostly attributed to the efficiency of the inference rules we developed for the forgetting method. Another reason was that our method introduces definers in a *conservative* manner (only when really necessary), whereas LETHE introduces them in a *systematic* and *exhaustive* manner.

Compared to FAME, similar speed was observed for our prototype, but it fared considerably better with respect to success rates (94.4% as opposed to 62.8%). This is because FAME computes solutions of semantic forgetting, a stronger notion of forgetting than the one used in this paper, and such solutions are often expressed with extra expressivity, which is not in the language of the given

[5] A "cyclic dependency" means that a name is defined in terms of itself
[6] https://owlcs.github.io/owlapi/
[7] http://bioportal.bioontology.org/

ontology.[8] The column headed 'Extra Exp.' shows the percentage of the test cases with undesired terms or extra expressivity in the resultant ontology: our prototype uses definers, and LETHE and FAME uses fixpoints to capture cyclic dependencies. FAME uses additionally inverse roles, the universal role, as well as role conjunction to represent solutions of semantic forgetting.

The source code for our forgetting prototype is *anonymously* distributed at the GitHub hosting site. [9] A runnable jar file, a README file, and the test data for the evaluation are also provided there.

6 CASE STUDY

In this section, we study how our forgetting prototype performs in practice for computing views of ontologies in the case of ontology-based query answering. In particular, we use the prototype to compute views of the latest international version of the SNOMED CT ontology, and then query each view and the base SNOMED CT with 100 artifact DL queries generated using Protégé,[10] a dominantly-used ontology editor. The purpose of this case study is to check: (i) if the view computed by our prototype can return the same answers as the original ontology, and (ii) if the average query response time can be reduced by using a view of an ontology. We aim to show the usefulness of the forgetting technique and the competence of our forgetting method for realistic ontology-based knowledge processing applications such as ontology-based query answering.

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	$\#\mathcal{F}_{\mathsf{C}}$	$\#\mathcal{F}_{\mathsf{R}}$	$\#\mathcal{V}$	Y/N	ARTO	ARTV
body structure	305,007	205	50,205	Y	31.8s	4.3s
clinical finding	317,284	201	41,033	Y	30.2s	4.1s
observable entity	291,000	204	57,840	Y	33.5s	5.6s
organism	262,178	169	87,542	Y	32.6s	5.9s
physical force	319,483	211	41,158	Y	39.7s	4.3s
procedure	216,544	161	131,121	Y	30.9s	12.7s
specimen	287,695	197	65,386	Y	38.7s	5.4s
substance	271,984	199	74,077	Y	33.1s	5.6s

Figure 11: 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 find the terms in each topic domain (the target signature). This can be done by accessing the metrics of SNOMED CT.[11] Then we exploit the Sets.difference(set1, set2) method provided in Google 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 use our prototype to compute the 8 views of SNOMED CT by eliminating the names in each forgetting signature.

Forgetting is successful in all cases. The results are shown in Figure 11. By $\#\mathcal{F}_{\mathsf{C}}$ and $\#\mathcal{F}_{\mathsf{R}}$ we denote the number of concept names and the number of role names in the forgetting signature. By $\#\mathcal{V}$ we denote the number of axioms in the computed view. The column headed "Y/N" indicates if same answers are returned by the computed view and original ontology when querying them with a set of 100 artifact DL queries. only if the same answer is returned for each query, "Y" can be filled in. 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 takes much shorter time than on the entire ontology. Usually, the more the terms are forgotten, the smaller the size of the computed view is, and the shorter the response time can be expected.

7 CONCLUSION AND FUTURE WORK

In this paper, we have explored the problem of computing views of ontologies in expressive description logics not considered before. In particular, we have developed a novel, practical forgetting method for $\mathcal{ALCOIH}$, the basic $\mathcal{ALC}$ extended with nominals, inverse roles and role inclusions. The method is terminating and sound, and can eliminate concept and role names from $\mathcal{ALCOIH}$-ontologies. An empirical comparison of a prototype of method with two existing systems have shown that the method achieves a significant speed-up over LETHE on the $\mathcal{ALCH}$-fragments, and has notably better success rates than FAME, a semantic forgetting tool for $\mathcal{ALCOIH}$. A case study on the ontology-based query answering problem has verified the usefulness of the forgetting technique and the competence of the forgetting method for real-world applications. This is extremely useful from the perspective of ontology engineering, as it provides ontology curators with a powerful approach and tooling support to creating views of ontologies.

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.

[8] It was defined in [36] that FAME succeeds if it has eliminated all names in $\mathcal{F}$. We slightly adjust this definition in this paper: FAME succeeds if it has eliminated all names in $\mathcal{F}$, while without introducing extra expressivity in the resultant ontology.
[9] https://github.com/anonymous-ai-researcher/prototype
[10] https://protege.stanford.edu/
[11] https://browser.ihtsdotools.org/?

REFERENCES

[1] Wilhelm Ackermann. 1935. Untersuchungen ü̈ber das Eliminationsproblem der mathematischen Logik. *Math. Ann.* 110, 1 (1935), 390–413.

[2] Michael Ashburner. 2000. Gene Ontology: Tool for the unification of biology. *Nature Genetics* 25 (2000), 25–29.

[3] Franz Baader, Ian Horrocks, Carsten Lutz, and Ulrike Sattler. 2017. *An Introduction to Description Logic.* Cambridge University Press.

[4] Leo Bachmair, Harald Ganzinger, David A. McAllester, and Christopher Lynch. 2001. Resolution Theorem Proving. See [27], 19–99.

[5] Christian G. Fermüller, Alexander Leitsch, Ullrich Hustadt, and Tanel Tammet. 2001. Resolution Decision Procedures. See [27], 1791–1849.

[6] B. C. Grau and B. Motik. 2012. Reasoning over Ontologies with Hidden Content: The Import-by-Query Approach. *J. Artif. Intell. Res.* 45 (2012), 197–255.

[7] Martin Hepp, Pieter De Leenheer, Aldo de Moor, and York Sure (Eds.). 2008. *Ontology Management, Semantic Web, Semantic Web Services, and Business Applications.* Semantic Web and Beyond: Computing for Human Experience, Vol. 7. Springer.

[8] Michel C. A. Klein and Dieter Fensel. 2001. Ontology versioning on the Semantic Web. In *Proc. SWWS'01.* 75–91.

[9] Michel C. A. Klein, Dieter Fensel, Atanas Kiryakov, and Damyan Ognyanov. 2002. Ontology Versioning and Change Detection on the Web. In *Proc. EKAW'02 (Lecture Notes in Computer Science)*, Asunción Gómez-Pérez and V. Richard Benjamins (Eds.), Vol. 2473. Springer, 197–212.

[10] Boris Konev, Carsten Lutz, Dirk Walther, and Frank Wolter. 2013. Model-theoretic inseparability and modularity of description logic ontologies. *Artif. Intell.* 203 (2013), 66–103.

[11] Boris Konev, Dirk Walther, and Frank Wolter. 2008. The Logical Difference Problem for Description Logic Terminologies. In *IJCAR (Lecture Notes in Computer Science)*, Vol. 5195. Springer, 259–274.

[12] Boris Konev, Dirk Walther, and Frank Wolter. 2009. Forgetting and Uniform Interpolation in Large-Scale Description Logic Terminologies. In *Proc. IJCAI'09.* IJCAI/AAAI Press, 830–835.

[13] Patrick Koopmann. 2015. *Practical Uniform Interpolation for Expressive Description Logics.* Ph.D. Dissertation. The University of Manchester, UK.

[14] Patrick Koopmann and Renate A. Schmidt. 2013. Uniform Interpolation of $\mathcal{ALC}$-Ontologies Using Fixpoints. In *Proc. FroCos'13 (Lecture Notes in Computer Science)*, Vol. 8152. Springer, 87–102.

[15] Patrick Koopmann and Renate A. Schmidt. 2014. Count and Forget: Uniform Interpolation of $\mathcal{SHQ}$-Ontologies. In *Proc. IJCAR'14 (Lecture Notes in Computer Science)*, Vol. 8562. Springer, 434–448.

[16] Patrick Koopmann and Renate A. Schmidt. 2015. Saturated-Based Forgetting in the Description Logic $\mathcal{SIF}$. In *Proc. DL'15 (CEUR Workshop Proceedings)*, Vol. 1350. CEUR-WS.org.

[17] Patrick Lambrix and He Tan. 2008. Ontology Alignment and Merging. In *Anatomy Ontologies for Bioinformatics, Principles and Practice.* Computational Biology, Vol. 6. Springer, 133–149.

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

[19] C. Lutz, I. Seylan, and F. Wolter. 2012. An Automata-Theoretic Approach to Uniform Interpolation and Approximation in the Description Logic $\mathcal{EL}$. In *Proc. KR'12.* AAAI Press, 286–297.

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

[21] Nicolas Matentzoglu and Bijan Parsia. 2017. BioPortal Snapshot 30.03.2017. https://doi.org/10.5281/zenodo.439510

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

[23] Nadeschda Nikitina and Sebastian Rudolph. 2014. (Non-)Succinctness of uniform interpolants of general terminologies in the description logic $\mathcal{EL}$. *Artif. Intell.* 215 (2014), 120–140.

[24] Natalya Fridman Noy and Mark A. Musen. 2000. PROMPT: Algorithm and Tool for Automated Ontology Merging and Alignment. In *Proc. AAAI/IAAI'00.* AAAI Press/The MIT Press, 450–455.

[25] Natalya Fridman Noy and Mark A. Musen. 2004. Ontology Versioning in an Ontology Management Framework. *IEEE Intelligent Systems* 19, 4 (2004), 6–13.

[26] Márcio Moretto Ribeiro and Renata Wassermann. 2009. Base Revision for Ontology Debugging. *J. Log. Comput.* 19, 5 (2009), 721–743.

[27] John Alan Robinson and Andrei Voronkov (Eds.). 2001. *Handbook of Automated Reasoning (in 2 volumes).* Elsevier and MIT Press.

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

[29] Dan Schrimpsher, Zhiqiang Wu, Anthony M. Orme, and Letha H. Etzkorn. 2010. Dynamic ontology version control. In *Proc. ACMse'10.* ACM, 25.

[30] Kent A. Spackman. 2000. SNOMED RT and SNOMED CT. Promise of an international clinical ontology. *M.D. Computing 17* (2000).

[31] Heiner Stuckenschmidt, Christine Parent, and Stefano Spaccapietra (Eds.). 2009. *Modular Ontologies: Concepts, Theories and Techniques for Knowledge Modularization.* Lecture Notes in Computer Science, Vol. 5445. Springer.

[32] Nicolas Troquard, Roberto Confalonieri, Pietro Galliani, Rafael Peñaloza, Daniele Porello, and Oliver Kutz. 2018. Repairing Ontologies via Axiom Weakening. In *Proc. AAAI'18.* AAAI Press, 1981–1988.

[33] K. Wang, G. Antoniou, R. W. Topor, and A. Sattar. 2005. Merging and Aligning Ontologies in DL-Programs. In *RuleML (Lecture Notes in Computer Science)*, Vol. 3791. Springer, 160–171.

[34] Yizheng Zhao, Ghadah Alghamdi, Renate A. Schmidt, Hao Feng, Giorgos Stoilos, Damir Juric, and Mohammad Khodadadi. 2019. Tracking Logical Difference in Large-Scale Ontologies: A Forgetting-Based Approach. In *Proc. AAAI'19.* AAAI Press, 3116–3124.

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

[36] Yizheng Zhao and Renate A. Schmidt. 2018. FAME: An Automated Tool for Semantic Forgetting in Expressive Description Logics. In *Proc. IJCAR'18 (Lecture Notes in Computer Science)*, Vol. 10900. Springer, 19–27.