UI-FAME: A Deductive Forgetting System for Creating Views of ALC-TBoxes*

Xuan Wu[1,2], Chang Lu[1,3], Yizheng Zhao[1,2,5], Hao Feng[4], Renate A. Schmidt[5], Yiwei Dai[1]

[1] National Key Laboratory for Novel Software Technology, Nanjing Univeristy, China
[2] School of Artificial Intelligence, Nanjing University, China
[3] School of Physics, Nanjing University, China
[4] North China University of Science and Technology, China
[5] Department of Computer Science, The University of Manchester, UK

Abstract. UI-FAME is a Java-based forgetting system for creating views of $\mathcal{ALC}$-TBoxes. The system implements the method developed in our previous work for forgetting concept and role names from $\mathcal{ALC}$-TBoxes. In this paper, we introduce UI-FAME and compare it with LETHE, a peer forgetting system for $\mathcal{ALC}$-ontologies and many its extensions, over the $\mathcal{ALC}$-TBox fragment of 494 realistic ontologies taken from the Oxford Ontology Repository. The comparison considers success rates, speed, memory consumption as the principal indicators to assess the performance of the two systems. For validation purpose, we also explore the semantic relationships between the views computed by the two systems. The experimental results showed that UI-FAME attained in general better success rates and performance results than LETHE. We found that in 97.08% cases UI-FAME and LETHE computed logically equivalent views, in 2.06% cases LETHE's solution entailed UI-FAME's solution but not the other way round, in 0.79% cases UI-FAME's solution entailed LETHE's solution but not the other way round, and in 0.07% cases they had no mutual entailment relationship. This was somewhat undesirable and thus needs further investigation.

1 Introduction

In Computer Science and Artificial Intelligence, *ontology* is a technical term denoting an artifact that is designed for a specific purpose to enable the modeling of knowledge about a domain of discourse. More specifically, an *ontology* is a formal representation of the knowledge within a domain of discourse using a set of representational primitives. These representational primitives are *classes* (*concepts*), *class members* (*individuals*), and *properties* (*attributes* of class members or *relationships* among class members).

Ontologies model domain knowledge of applications rooted in numerous industry sectors, including energy, laws, biology, medical, and healthcare sectors.

* Copyright © 2020 for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

Modern ontologies are specified in the Web Ontology Language (OWL) that has a formal semantics based on description logics (DLs) [2]; OWL is presently the most prevalent textual language for developing ontologies. Using a logic-based, well-structured language such as DLs has two notable advantages: (i) they have unambiguous formal semantics — the meaning of terms is specified in an unambiguous way, thereby enabling shared understanding of domain knowledge, and (ii) one can make use of the reasoning services of DL reasoners for ontology engineering and related tasks.

1.1 Computing Views of Ontologies

With the growing usage of ontologies in real-world applications, not only has the number of available ontologies increased considerably, but also they are becoming large in size, complex in structure, and thus 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 automated 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 developed from scratch, which is costly and error-prone. Computing 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 ontology views is useful for many ontology re-engineering and related tasks. These include, but are not limited to the following ones.

i. **Ontology Reuse**: Knowledge modelled in ontologies is often rich, heterogeneous, and multi-topic related, while applications are interested in or focused on specific parts. Compared to exploiting existing ontologies or building new ontologies from scratch, extracting fragments w.r.t. specific topics from existing ontologies and reusing them in a specialized context is simpler, cheaper, and thus more interesting to the ontology engineers.
ii. **Information Hiding** Medical and military ontologies may contain sensitive information that must be kept confidential to the public and the correspondences when the ontologies are published, shared, or disseminated. The confidentiality and protection can be achieved through the removal of concept and role name relative to sensitive information.
iii. **Ontology-Based Query Answering**: Taking ontological knowledge into account when retrieving data from relational databases has been widely acknowledged. It has been found in many cases [5] however that querying a large knowledge base often involves massive reasoning, which, due to high computational complexity of reasoning in DLs, can be very expensive both in terms of time and space. Instead, querying a view of the knowledge base which contains full information about the query seems a good solution.

Computing ontology views is also useful for many other tasks such as ontology alignment and merging [17,22,12], versioning [6,7,18,20], debugging and repair [19,21], and logical difference computation [9,10,13,24].

1.2 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, after the elimination, all logical consequences are preserved up to the remaining signature. Forgetting in this sense can be used as a means of computing views of ontologies: the ontology obtained from forgetting, namely the *forgetting solution*, is the view of the original ontology for the target signature, which corresponds to the remaining signature in forgetting.

Related notions of forgetting are inseparability and conservative extensions [4]. All these notions can be formalized deductively or model-theoretically. In particular, forgetting can be formalized as *deductive forgetting* (*weak forgetting* [23], *consequence-based forgetting*), and formalized as *model-theoretic forgetting* [10,25]. As their names indicate, deductive notion has the property that deductive solutions retain all logical consequences up to the terms in the remaining signature, while model-theoretic notion has the property that model-theoretic solutions require equivalence to be preserved on the model level — the terms in the remaining signature must be interpreted in the same way as in the original ontology. Hence, the model-theoretic notion is a stronger notion of forgetting than the deductive one, and model-theoretic solutions are in general stronger than deductive ones — the former always entails the latter, but the converse does not hold. Computing model-theoretic solutions often requires the target language to be extended with extra expressivity. For example, for $\mathcal{ALC}$, the model-theoretic solutions often involve nominals, inverse roles and the universal role, while the deductive ones are expressed in $\mathcal{ALC}$ [26].

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. Previous studies have shown that: (i) deductive or model-theoretic solutions of forgetting do not always exist for $\mathcal{EL}$ and $\mathcal{ALC}$ [9,15,8], (ii) deciding the existence of deductive solutions is ExpTime-complete for $\mathcal{EL}$ [14] and 2ExpTime-complete for $\mathcal{ALC}$ [15], (iii) the existence of model-theoretic solutions of forgetting is undecidable for $\mathcal{EL}$ and $\mathcal{ALC}$ [8,4], and (iv) deductive solutions of forgetting can be triple exponential in size w.r.t. the input ontologies for $\mathcal{EL}$ and $\mathcal{ALC}$ [15,16].

Although forgetting is a challenging problem, there is however general consensus on its potential for ontology-based knowledge processing, and there has been continuous efforts dedicated into the development and automation of practical methods for forgetting. A few such methods have thus been developed and automated for various description logics. These methods include LETHE [11], the method developed by [13], and the method by [24]. In particular, LETHE uses a *resolution*-based approach [3], and can eliminate concept and role names from $\mathcal{ALCH}$-TBoxes and eliminate concept names from $\mathcal{SHQ}$-TBoxes. The one of [13] is based on resolution as well; it can eliminate concept names from $\mathcal{ALC}$-TBoxes. UI-FAME is a hybrid approach using both resolution and a monotonicity property called Ackermann's Lemma [1]; it can eliminate concept and role names from $\mathcal{ALC}$-TBoxes. All these methods have prototypical implementations, among which, only an early version of LETHE is currently publicly accessible.[6]

1.3 Contribution

In this paper, we introduce UI-FAME, a Java implementation of the forgetting method developed in our previous work [24]. We compared UI-FAME with the peer LETHE system on the $\mathcal{ALC}$-TBox fragment of 494 ontologies taken from the Oxford Ontology Repository. The comparison considered success rates, speed, memory consumption as the principal indicators to assess the performance of the systems. The experimental results showed that UI-FAME had in general better success rates and performance results than LETHE. We found that in 97.08% cases UI-FAME and LETHE computed logically equivalent forgetting solutions, in 2.06% cases LETHE's solution entailed UI-FAME's solution but not the other way round, in 0.79% cases UI-FAME'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.

2 Preliminaries

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

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

where $A \in \mathsf{N_C}$, $r \in \mathsf{N_R}$, and C and D are arbitrary concepts.

An *$\mathcal{ALC}$-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. In the remainder of this paper, the terms *TBox* and *ontology* are used interchangeably.

The semantics of $\mathcal{ALC}$-TBox 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 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:

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

[6] http://www.cs.man.ac.uk/ koopmanp/lethe/index.html

Let $\mathcal{I}$ be an interpretation. A concept equivalence $C \equiv D$ is *true* in $\mathcal{I}$ (or $\mathcal{I}$ satisfies $C \equiv D$) iff $C^{\mathcal{I}} \equiv D^{\mathcal{I}}$. A concept inclusion $C \sqsubseteq D$ is *true* in $\mathcal{I}$ (or $\mathcal{I}$ satisfies $C \sqsubseteq D$) iff $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$. $\mathcal{I}$ 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 $\mathsf{sig}_{\mathsf{C}}(X)$ and $\mathsf{sig}_{\mathsf{R}}(X)$ we denote respectively the sets of the concept names and role names occurring in X, where X ranges over concepts, axioms, and a set of axioms (ontologies). We define $\mathsf{sig}(X) = \mathsf{sig}_{\mathsf{C}}(X) \cup \mathsf{sig}_{\mathsf{R}}(X)$.

Definition 1 (Deductive Forgetting). *Let $\mathcal{O}$ be an $\mathcal{ALC}$-TBox and let $\mathcal{F} \subseteq \mathsf{sig}(\mathcal{O})$ be a set of concept and role names. An $\mathcal{ALC}$-TBox $\mathcal{V}$ is a* solution of deductively 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 means 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 $\mathcal{V}$ of an ontology $\mathcal{O}$ is the strongest entailment of $\mathcal{O}$ 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 Implementation of UI-FAME

UI-FAME is implemented in Java using the OWL API,[7] a Java API for creating, parsing, manipulating, and serializing OWL ontologies, and is released under the open source licenses LGPL[8] and Apache[9]. UI-FAME uses the OWL API Version 3.4.7 for the aforementioned tasks.

Figure 1 depicts the general design of UI-FAME. Given as input to UI-FAME are an $\mathcal{ALC}$-TBox $\mathcal{O}$, a set $\mathcal{F}_{\mathsf{C}} \subseteq \mathsf{sig}_{\mathsf{C}}(\mathcal{O})$ of concept names to be forgotten, and

[7] http://owlcs.github.io/owlapi/
[8] https://www.gnu.org/licenses/lgpl-3.0.html
[9] https://www.apache.org/licenses/LICENSE-2.0

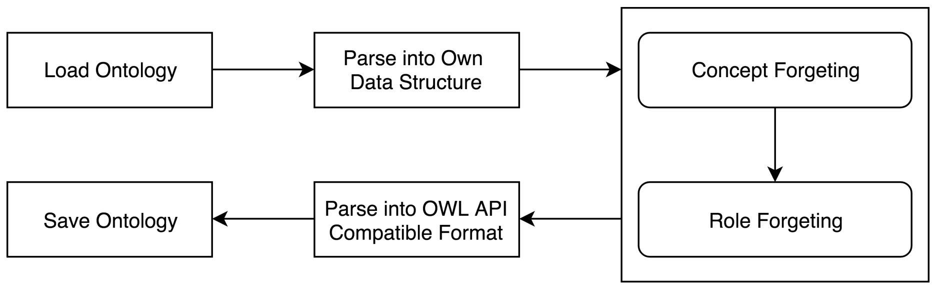
Load Ontology
Parse into Own Data Structure
Concept Forgeting
Role Forgeting
Parse into OWL API Compatible Format
Save Ontology

Fig. 1. General Design of UI-FAME

a set of $\mathcal{F}_{\mathsf{R}} \subseteq \mathsf{sig}_{\mathsf{R}}(\mathcal{O})$ of role names to be forgotten. Together, $\mathcal{F}_{\mathsf{C}}$ and $\mathcal{F}_{\mathsf{R}}$ make up the forgetting signature $\mathcal{F}$. The input ontology must be given as a text file in XML, OWL, RDF, or TURTLE format, or a URL pointing to the file. UI-FAME takes only $\mathcal{ALC}$-axioms; axioms not expressible in $\mathcal{ALC}$ are removed. Figures 2 lists the types of axioms handled by UI-FAME, which, via simple reformulations, can be represented as SubClassOf axioms.

Types of Axioms	**SubClassOf Representation**
SubClassOf(C1 C2)	SubClassOf(C1 C2)
EquivalentClasses(C1 C2)	SubClassOf(C1 C2), SubClassOf(C2 C1)
DisjointClasses(C1 C2)	SubClassOf(C1 ObjectComplementOf(C2))
ObjectPropertyDomain(R C)	SubClassOf(ObjectSomeValuesFrom(R owl:Thing), C)
ObjectPropertyRange(R C)	SubClassOf(owl:Thing ObjectAllValuesFrom(R C))

Fig. 2. Types of axioms handled by UI-FAME

UI-FAME computes a solution of forgetting $\mathcal{F}$ from $\mathcal{O}$ by eliminating single concept and role names in $\mathcal{F}$. Concept and role names are eliminated using two mutually independent calculi for respectively concept and role name elimination for $\mathcal{ALC}$-TBoxes [24]. The resulting ontology is returned in OWL/XML format. The source code and an executable .jar file of UI-FAME can be found at `https://github.com/anonymous-ai-researcher/uifame`. Another access to try out UI-FAME is via the online platform `http://www.forgettingshow.info/`.

4 Comparison of UI-FAME and LETHE

We compared UI-FAME with LETHE to gain an insight into the overall performance of the two systems considering success rate, speed, size of resulting ontology and memory consumption.[10]

4.1 Comparison Framework

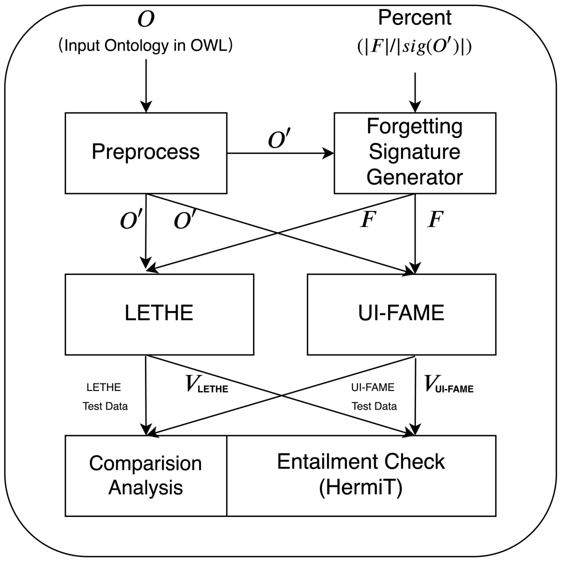

Fig. 3. Comparison Framework of UI-FAME and LETHE

Figure 3 depicts the comparison framework. Given $\mathcal{O}$ as the input ontology, an $\mathcal{ALC}$-TBox fragment $\mathcal{O}'$ is obtained via the preprocessing step by removing all axioms not expressible as $\mathcal{ALC}$-TBox axioms. Based on $\mathsf{sig}(\mathcal{O}')$, forgetting signatures, which are randomly generated in the Forgetting Signature Generator module, are delivered into UI-FAME and LETHE together with $\mathcal{O}'$. Here, we didn't pick symbols by their frequency of occurrence. On account of forgetting frequent symbols is not friendly to the case study, which is very important in this phase. But randomly symbol generation is not enough, for further investigations, we may use more sampling strategies.

$\mathcal{V}_{\text{LETHE}}$ and $\mathcal{V}_{\text{UI-FAME}}$ are the ontologies obtained from forgetting (not necessarily forgetting solutions) using respectively LETHE and UI-FAME. We used

[10] We did not bring the implementation of the forgetting method of [13] into the comparison because the method can only eliminate concept names, and moreover, is not currently publicly accessible.

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 UI-FAME and LETHE. The experiments were run on a laptop with an Intel Core i7-9750H processor, 6 cores running at up to 2.60 GHz, and 16 GB of DDR4-1330 MHz memory. Both UI-FAME and LETHE were allocated 8GB heap space for each test round.

4.2 Test Data

Table 1. 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
	$\lvert TBox \rvert$	0	990	162	262	658
II	$\lvert N_C \rvert$	200	5877	1665	1769	2801
	$\lvert N_R \rvert$	0	887	11	34	61
	$\lvert TBox \rvert$	1008	4976	2282	2416	3937
III	$\lvert N_C \rvert$	1162	9809	4042	5067	8758
	$\lvert N_R \rvert$	1	158	4	23	158
	$\lvert TBox \rvert$	5112	9783	7277	7195	9179

The ontologies used for the comparison were taken from the Oxford Ontology Repository (Oxford-ISG).[13] Oxford-ISG contained a large number of ontologies collected from multiple sources. In particular, Oxford-ISG contained 797 ontologies, and we took 494 of them with the number $|TBox|$ of TBox axioms in the ontology not exceeding 10000. We further split the entire corpus of 494 ontologies into three groups: Corpus I with $10 \leq |TBox| \leq 1000$, containing 356 ontologies, Corpus II with $1000 \leq |TBox| \leq 5000$, containing 108 ontologies, and Corpus III with $5000 \leq |TBox| \leq 10000$, containing 26 ontologies. This gives a clearer insight into how LETHE and UI-FAME perform forgetting for ontologies of different sizes. Table 1 shows statistical information about the selected ontologies, where $|N_C|$ and $|N_R|$ denote the average numbers of the concept names and role names in the selected ontologies.

4.3 Performance Comparison

To fit for real-world application scenarios, the standard of forgetting success was set to be: (1) forgetting all the terms in the forgetting signature $\mathcal{F}$; (2) without

[11] https://testng.org/doc/
[12] http://www.hermit-reasoner.com/
[13] https://www.cs.ox.ac.uk/isg/ontologies/

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 20 minutes and heap space to 8GB.

Both UI-FAME and LETHE were tested over the three Oxford-ISG snapshots. For each snapshot, we considered forgetting respectively 10% and 30% of the terms from the signature of each ontology. With a randomly generated forgetting signature, the test was repeated three times for UI-FAME and LETHE.

Table 2. Performance Results (Mem: memory consumption of successful cases, S-Rate: success rate, TO-Rate: timeout rate, MO-Rate: out of memory rate, Extra: extra expressivity percent)

			Duration(s)	Mem(MB)	S-Rate	TO-Rate	MO-Rate	Extra
UI-FAME	0.1	I	6.2	43	89.7	2.9	0.0	7.4
		II	13.3	349	81.7	12.5	0.0	5.7
		III	3.6	306	81.5	16.9	0.0	1.5
	0.3	I	4.8	106	85.4	13.3	0.0	1.3
		II	19.3	301	57.5	42.5	0.0	0.0
		III	26.9	424	68.6	31.4	0.0	0.0
LETHE	0.1	I	9.7	199	85.4	7.1	0.0	7.3
		II	10.2	1101	66.7	31.8	0.0	1.5
		III	48.1	1881	65.4	25.6	0.0	5.1
	0.3	I	21.7	322	73.9	18.2	0.0	7.7
		II	41.4	868	51.2	47.5	0.0	1.2
		III	69.7	1083	50.0	46.2	0.0	5.0

The results are shown in Table 2, where "Duration" denotes the average time consumption of the successful cases, and "Extra" denotes the percentage of cases introducing extra expressivity outside of $\mathcal{ALC}$. Such expressivity is not desired to be in forgetting solutions; if extra expressivity cannot be removed from the resulting ontologies, then the forgetting fails.

UI-FAME had better success rates than LETHE because of a less timeout rate attained by UI-FAME (TO-Rate). Apparently, UI-FAME was faster than LETHE when forgetting percentage is relatively big. This is partly because the elimination in LETHE is based on resolution, while UI-FAME uses both resolution and Ackermann's Lemma; the latter allows concept names to be eliminated more cheaply when forgetting task is big. In addition, we found that in almost 97% of the elimination rounds, a concept name could be eliminated using Ackermann's Lemma. Another fact that accounts for such performance results is that UI-FAME 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 1. Let $\mathcal{O} = \{C \sqcup \exists r.A, E \sqcup \forall r.\neg A\}$ *and* $\mathcal{F} = \{A\}$. $\mathsf{N_D}$ *is the definers set disjoint with the signature of* $\mathcal{O}$. *UI-FAME applies directly the combination*

rule (Case 8) to $\mathcal{O}$ 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, $\mathcal{O}$ 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* $\{12, 14\}$.

The results also show that UI-FAME had lower memory consumption during the forgetting process; see the "Mem" column. In particular, LETHE's memory consumption was about four times of UI-FAME. We believe that this could be attributed to the definer introduction mechanism employed by LETHE.

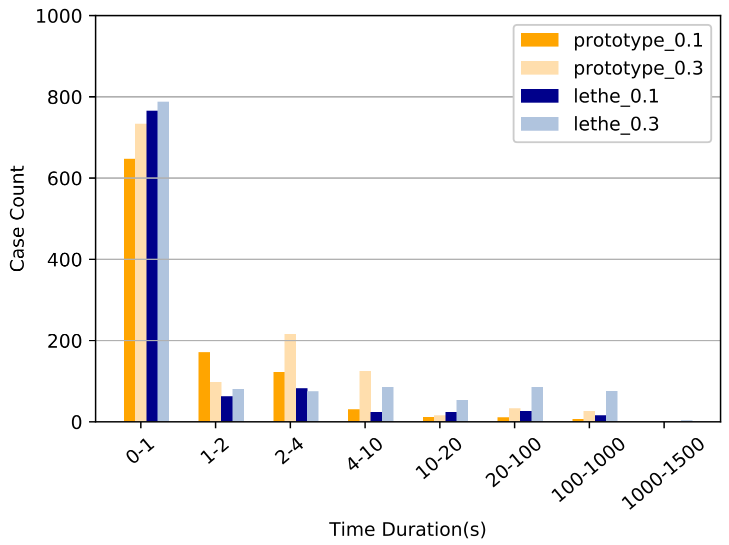

Fig. 4. Statistical graph of samples in each time period

Figure 4 depicts the cach time distribution for all the successful cases, where it can be seen that most successful cases were finished within 1 second for both LETHE and UI-FAME, while UI-FAME had more such cases. In most test cases, the variation in the sizes of the forgetting signatures did not affect the overall time consumption too much. This was an undesirable yet interesting observation which is worth a comprehensive study; we leave this as one of our future work.

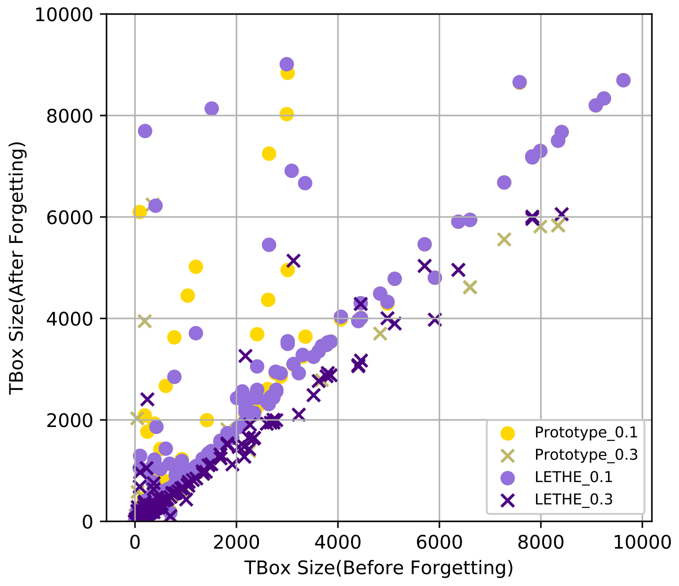

Fig. 5. TBox scatter plot before and after forgetting

Figure 5 exhibits the changes in the number of the axioms in the ontologies before and after forgetting in each group of experiments. In general, with the increase in the size of the original ontologies, the size of the forgetting solutions decreases; this seems against the theoretical result found in [15] that forgetting can lead to exponential space explosion in the worst cases. This means that, such worst cases rarely occur in real-world scenarios.

4.4 Entailment Relationship Checking

In principle, UI-FAME and LETHE 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). Based on the forgetting solutions obtained in the experiment, we compared the entailment relationship of these two systems for the successful cases. HermiT[14] was employed to compute the inequivalence between the forgetting solutions computed by UI-FAME and LETHE.

The results showed that in 97.08% cases UI-FAME and LETHE computed logically equivalent forgetting solutions, in 2.06% cases LETHE's solution entailed UI-FAME's solution but not the other way round, in 0.79% cases UI-FAME's solution entailed LETHE's solution but not the other way round, and in 0.07% cases they had no mutual entailment relationship. The inequivalent cases are caused by the data procession or maybe tool bugs. This is somewhat undesirable and thus needs further investigation.

5 Conclusion and Future Work

This paper presents an empirical comparison of two forgetting systems, namely UI-FAME and LETHE, showing better performance of UI-FAME.

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 *ü*ber das Eliminationsproblem der mathematischen Logik. *Mathematische Annalen*, 110(1):390–413, 1935.
2. F. Baader, I. Horrocks, C. Lutz, and U. Sattler. *An Introduction to Description Logic*. Cambridge University Press, 2017.
3. L. Bachmair, H. Ganzinger, D. A. McAllester, and C. Lynch. Resolution theorem proving. In J. A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning (in 2 volumes)*, pages 19–99. Elsevier and MIT Press, 2001.
4. E. Botoeva, B. Konev, C. Lutz, V. Ryzhikov, F. Wolter, and M. Zakharyaschev. Inseparability and Conservative Extensions of Description Logic Ontologies: A Survey. In *Proc. RW'16*, volume 9885 of *Lecture Notes in Computer Science*, pages 27–89. Springer, 2016.
5. 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.
6. M. C. A. Klein and D. Fensel. Ontology versioning on the Semantic Web. In *Proc. SWWS'01*, pages 75–91, 2001.
7. M. C. A. Klein, D. Fensel, A. Kiryakov, and D. Ognyanov. Ontology Versioning and Change Detection on the Web. In A. Gómez-Pérez and V. R. Benjamins, editors, *Proc. EKAW'02*, volume 2473 of *Lecture Notes in Computer Science*, pages 197–212. Springer, 2002.

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

8. B. Konev, C. Lutz, D. Walther, and F. Wolter. Model-theoretic inseparability and modularity of description logic ontologies. *Artif. Intell.*, 203:66–103, 2013.
9. B. Konev, D. Walther, and F. Wolter. The Logical Difference Problem for Description Logic Terminologies. In *IJCAR*, volume 5195 of *Lecture Notes in Computer Science*, pages 259–274. Springer, 2008.
10. B. Konev, D. Walther, and F. Wolter. Forgetting and Uniform Interpolation in Large-Scale Description Logic Terminologies. In *Proc. IJCAI'09*, pages 830–835. IJCAI/AAAI Press, 2009.
11. P. Koopmann. *Practical Uniform Interpolation for Expressive Description Logics*. PhD thesis, The University of Manchester, UK, 2015.
12. P. Lambrix and H. Tan. Ontology Alignment and Merging. In *Anatomy Ontologies for Bioinformatics, Principles and Practice*, volume 6 of *Computational Biology*, pages 133–149. Springer, 2008.
13. 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.
14. C. Lutz, I. Seylan, and F. Wolter. An Automata-Theoretic Approach to Uniform Interpolation and Approximation in the Description Logic $\mathcal{EL}$. In *Proc. KR'12*, pages 286–297. AAAI Press, 2012.
15. 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.
16. N. Nikitina and S. Rudolph. (Non-)Succinctness of uniform interpolants of general terminologies in the description logic $\mathcal{EL}$. *Artif. Intell.*, 215:120–140, 2014.
17. 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.
18. N. F. Noy and M. A. Musen. Ontology versioning in an ontology management framework. *IEEE Intelligent Systems*, 19(4):6–13, 2004.
19. M. M. Ribeiro and R. Wassermann. Base revision for ontology debugging. *J. Log. Comput.*, 19(5):721–743, 2009.
20. D. Schrimpsher, Z. Wu, A. M. Orme, and L. H. Etzkorn. Dynamic ontology version control. In *Proc. ACMse'10*, page 25. ACM, 2010.
21. 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.
22. K. Wang, G. Antoniou, R. W. Topor, and A. Sattar. Merging and aligning ontologies in DL-programs. In *RuleML*, volume 3791 of *Lecture Notes in Computer Science*, pages 160–171. Springer, 2005.
23. Y. Zhang and Y. Zhou. Forgetting revisited. In *Twelfth International Conference on the Principles of Knowledge Representation and Reasoning*, 2010.
24. 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.
25. Y. Zhao and R. A. Schmidt. Role forgetting for $\mathcal{ALCOQH}(\triangledown)$-ontologies using an ackermann-based approach. In *Proc. IJCAI'17*, pages 1354–1361. IJCAI/AAAI Press, 2017.
26. Y. Zhao and R. A. Schmidt. FAME: An Automated Tool for Semantic Forgetting in Expressive Description Logics. In *Proc. IJCAR'18*, volume 10900 of *Lecture Notes in Computer Science*, pages 19–27. Springer, 2018.