Tracking Logical Difference in Large-Scale Ontologies: A Forgetting-Based Approach

Yizheng Zhao
The University of Manchester

Ghadah Alghamdi
The University of Manchester

Renate A. Schmidt
The University of Manchester

Hao Feng
North China University of Science & Technology

Giorgos Stoilos
Babylon Health

Damir Juric
Babylon Health

Mohammad Khodadadi
Babylon Health

Abstract

This paper explores how the logical difference between two ontologies can be tracked using a forgetting-based or uniform interpolation (UI)-based approach. The idea is that rather than computing all entailments of one ontology not entailed by the other ontology, which would be computationally infeasible, only the strongest entailments not entailed in the other ontology are computed. To overcome drawbacks of existing forgetting/uniform interpolation tools we introduce a new forgetting method designed for the task of computing the logical difference between different versions of large-scale ontologies. The method is sound and terminating, and can compute uniform interpolants for $\mathcal{ALC}$-ontologies as large as SNOMED CT and NCIt. Our evaluation shows that the method can achieve considerably better success rates ($>90\%$) and provides a feasible approach to computing the logical difference in large-scale ontologies, as a case study on different versions of SNOMED CT and NCIt ontologies shows.

Introduction

Ontologies are widely employed to give structured representations of domain knowledge suitable for AI reasoning and have become increasingly popular across numerous industry sectors, including medical, digital biology and energy sectors. In this paper we are concerned with the problem of tracking the changes between different versions of an ontology, which can be done based on the notion of *logical difference*. The logical difference between two ontologies are the axioms in one that are not entailed by the other, reflecting the information gained and information lost between the ontologies (Konev *et al.* 2008; 2012).

From the perspective of ontology engineers, tracking the differences of large-scale ontologies is a critical task for the purpose of maintaining them: to track what has changed in a new version of an ontology, to ensure the extension is safe in the sense of being a conservative extension (Lutz *et al.* 2007), and to identify gained and lost information (Klein *et al.* 2002). This provides a means of discovering issues in the ontologies and enhance quality control, which is important for ontology curators. Being able to compute the logical difference between two ontologies is also important when merging/aligning ontologies or integrating different ontologies (Jiménez-Ruiz *et al.* 2011; Solimando *et al.* 2017; Stoilos *et al.* 2018).

At present most existing ontology difference tools and ontology alignment/merging tools are only able to compute the *structural differences* (Noy and Musen 2002; Gonçalves *et al.* 2012) or to *approximate* the logical difference (Konev *et al.* 2012; Jiménez-Ruiz *et al.* 2011; Kremen *et al.* 2011; Solimando *et al.* 2017). A particular challenge is the size of ontologies used in real-world applications. Our target are SNOMED CT and NCIt ontologies. Being the most comprehensive, multi-lingual medical ontology in the world, the core SNOMED CT ontology contains over 335K concepts definitions (Spackman 2000). The NCIt ontologies define terminologies in the biomedical domain and include more than 60K concept definitions (Hartel *et al.* 2005).

In the present paper we explore how the logical difference between two ontologies can be tracked using a forgetting-based or Uniform Interpolation (UI)-based approach, as proposed in (Ludwig and Konev 2014). Forgetting is an abstraction technique which preserves the underlying logical model of the concepts and relationships (Lin and Reiter 1994; Konev *et al.* 2009b; Lutz and Wolter 2011; Wang *et al.* 2014). However, currently, there are no good forgetting tools that allow us to track logical differences efficiently. To address this shortcoming we introduce a new forgetting method designed for the task of computing the logical difference of large-scale ontologies. The method is sound and terminating, and can compute uniform interpolants for $\mathcal{ALC}$-ontologies as large as SNOMED CT and NCIt. Our evaluation shows the method can achieve considerably better success rates ($>90\%$). Based on this new forgetting method we have developed a new logical difference analysis tool to analyze the differences between the Australia and Canada country extensions of the base SNOMED CT International ontology and recent versions of the NCIt ontologies. While the Canada extension was found to be a conservative extension of SNOMED CT International, the diversion to the Australia version is significant, which is reflected in large logical difference sets.

Some Basics of Description Logic $\mathcal{ALC}$

In this paper the focus is on the description logic $\mathcal{ALC}$. *Concepts* in $\mathcal{ALC}$ 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 $A \in \mathsf{N_C}$ denote *concept names*, $r \in \mathsf{N_R}$ denote *role names*, and C and D denote arbitrary concepts. We assume an $\mathcal{ALC}$-ontology contains only a TBox, i.e., a set of axioms of the form $C \sqsubseteq D$ (*concept inclusions*), where C and D are concepts. The semantics of $\mathcal{ALC}$ is defined as usual.

Let us fix some commonly used notation. By $\mathsf{sig_C}(X)$ and $\mathsf{sig_R}(X)$ we denote respectively the sets of the concept names and role names occurring in X, where X ranges over concepts, clauses, axioms, sets of clauses and sets of axioms (ontologies). We let $\mathsf{sig}(X) = \mathsf{sig_C}(X) \cup \mathsf{sig_R}(X)$. A clause is called an $\mathcal{S}$*-clause* if it contains $\mathcal{S}$, where $\mathcal{S} \in \mathsf{N_C} \cup \mathsf{N_R}$.

Logical Difference via Forgetting

The notion of logical difference used in this paper was introduced by (Konev *et al.* 2008) as a means of capturing the difference in the meaning of terms that is independent of the representation of ontologies.

Definition 1 (Logical Difference for $\mathcal{ALC}$). *Let $\mathcal{O}_1$ and $\mathcal{O}_2$ be two $\mathcal{ALC}$-ontologies. Let $\Sigma = \mathsf{sig}(\mathcal{O}_1) \cap \mathsf{sig}(\mathcal{O}_2)$ be the common signature of $\mathcal{O}_1$ and $\mathcal{O}_2$. The* logical difference between $\mathcal{O}_1$ and $\mathcal{O}_2$ *is the set $\mathsf{Diff}(\mathcal{O}_1, \mathcal{O}_2)$ of all $\mathcal{ALC}$-axioms α such that (i) $\mathsf{sig}(\alpha) \subseteq \Sigma$, (ii) $\mathcal{O}_2 \models \alpha$, but (iii) $\mathcal{O}_1 \not\models \alpha$. An axiom α satisfying these conditions is a* witness *of a change in $\mathcal{O}_2$ with respect to $\mathcal{O}_1$. If $\alpha \in \mathcal{O}_2$ we call it an* explicit witness*, else it is an* implicit witness*.*

We notice that $\mathsf{Diff}(\mathcal{O}_1, \mathcal{O}_2)$ computes *information gain* from $\mathcal{O}_1$ to $\mathcal{O}_2$, or *information loss* from $\mathcal{O}_2$ to $\mathcal{O}_1$.

To compute all witnesses in $\mathsf{Diff}(\mathcal{O}_1, \mathcal{O}_2)$ it is necessary to derive all entailments of the second ontology (satisfying the signature restriction) which are not entailed by the first ontology. In general, this set is however infinite. Existing logical difference tools compute therefore only approximations of the difference set. Many methods compute only witnesses that are simple inclusions of the form $A \sqsubseteq B$ (Jiménez-Ruiz *et al.* 2011; Gonçalves *et al.* 2012; Kremen *et al.* 2011) or also inclusions of a form $A \sqsubseteq \exists r.B$ (Stoilos *et al.* 2018), where A and B denote concept names and r is a role name in Σ. Tools like ECCO (Gonçalves *et al.* 2012) can only find explicit witnesses, i.e., witnesses α such that $\alpha \in \mathcal{O}_2$ and $\mathcal{O}_1 \not\models \alpha$, but not implicit ones.

A finite representation of $\mathsf{Diff}(\mathcal{O}_1, \mathcal{O}_2)$ can be computed using a uniform interpolation-based approach.

Definition 2 (Uniform Interpolation for $\mathcal{ALC}$). *Let $\mathcal{O}$ be an $\mathcal{ALC}$-ontology and $\Sigma \subseteq \mathsf{sig}(\mathcal{O})$ be a set of concept and role names. An ontology $\mathcal{V}$ is an* $\mathcal{ALC}$-uniform interpolant *of $\mathcal{O}$ for Σ iff the following conditions hold: (i) $\mathsf{sig}(\mathcal{V}) \subseteq \Sigma$ and (ii) for any $\mathcal{ALC}$-axiom α with $\mathsf{sig}(\alpha) \subseteq \Sigma$, $\mathcal{V} \models \alpha$ iff $\mathcal{O} \models \alpha$. Σ is called the* interpolation signature*.*

Uniform interpolants are strongest entailments. By definition, $\mathcal{V}$ is a strongest entailment of $\mathcal{O}$, if $\mathcal{O} \models \mathcal{V}$ and for any ontology $\mathcal{V}'$ such that $\mathcal{O} \models \mathcal{V}'$ and $\mathsf{sig}(\mathcal{V}') \subseteq \Sigma$ then $\mathcal{V} \models \mathcal{V}'$. In general it can be shown that: $\mathcal{V}$ is a uniform interpolant of an ontology $\mathcal{O}$ for Σ iff $\mathcal{V}$ is a strongest entailment of $\mathcal{O}$ in Σ. Uniform interpolants are unique up to logical equivalence.

Logical difference can therefore be related to uniform interpolation as follows: $\mathsf{Diff}(\mathcal{O}_1, \mathcal{O}_2) = \emptyset$ iff $\mathcal{O}_1 \models \mathcal{V}_2$, where $\mathcal{V}_2$ is a uniform interpolant of $\mathcal{O}_2$ for $\Sigma = \mathsf{sig}(\mathcal{O}_1) \cap \mathsf{sig}(\mathcal{O}_2)$, the common signature of $\mathcal{O}_1$ and $\mathcal{O}_2$. If $\mathcal{O}_1 \not\models \mathcal{V}_2$, this means that $\mathsf{Diff}(\mathcal{O}_1, \mathcal{O}_2)$ is non-empty. Then every axiom $\alpha \in \mathcal{V}_2$ with $\mathcal{O}_1 \not\models \alpha$ is a witness of $\mathsf{Diff}(\mathcal{O}_1, \mathcal{O}_2)$.

The idea of our method is to compute a uniform interpolant $\mathcal{V}_2$ of $\mathcal{O}_2$ for the common symbols in Σ and take the witnesses from this (Ludwig and Konev 2014).

Definition 3 (UI Difference for $\mathcal{ALC}$). *Let $\mathcal{O}_1$ and $\mathcal{O}_2$ be two $\mathcal{ALC}$-ontologies. Let $\Sigma = \mathsf{sig}(\mathcal{O}_1) \cap \mathsf{sig}(\mathcal{O}_2)$ be the common signature of $\mathcal{O}_1$ and $\mathcal{O}_2$. The* UI difference between $\mathcal{O}_1$ and $\mathcal{O}_2$ *is the set $\mathsf{UI\text{-}Diff}(\mathcal{O}_1, \mathcal{O}_2)$ of all $\mathcal{ALC}$-axiom α such that (i) $\mathsf{sig}(\alpha) \subseteq \Sigma$, (ii) $\alpha \in \mathcal{V}_2$ and (iii) $\mathcal{O}_1 \not\models \alpha$, where $\mathcal{V}_2$ is a uniform interpolant of $\mathcal{O}_2$ for Σ. α is referred to a* UI-witness *of a change in $\mathcal{O}_2$ with respect to $\mathcal{O}_1$.*

Since any $\alpha \in \mathcal{V}_2$ is an entailment of $\mathcal{O}_2$, every UI-witness is a witness and $\mathsf{UI\text{-}Diff}(\mathcal{O}_1, \mathcal{O}_2) \subseteq \mathsf{Diff}(\mathcal{O}_1, \mathcal{O}_2)$. Since all the witnesses can in principle be computed from the deductive closure of a Σ-uniform interpolant $\mathcal{V}_2$ of $\mathcal{O}_2$, we can think of $\mathsf{UI\text{-}Diff}(\mathcal{O}_1, \mathcal{O}_2)$ as a representation of the set $\mathsf{Diff}(\mathcal{O}_1, \mathcal{O}_2)$. For $\mathcal{ALC}$, $\mathsf{UI\text{-}Diff}(\mathcal{O}_1, \mathcal{O}_2)$ is in fact a *finite* representation of the witnesses, which is crucial (possibly involving auxiliary symbols).

From these considerations it follows that the set of UI-witnesses can be computed by this two step algorithm:

Step (1): compute the uniform interpolant $\mathcal{V}_2$ of $\mathcal{O}_2$ for $\Sigma = \mathsf{sig}(\mathcal{O}_1) \cap \mathsf{sig}(\mathcal{O}_2)$, and then

Step (2): collect the axioms $\alpha \in \mathcal{V}_2$ not entailed by $\mathcal{O}_1$.

The first step can be performed using an uniform interpolation tool, the second step can be done using an external DL reasoner. As a third step it may be useful to partition the (UI-)witnesses into explicit and implicit (UI-)witnesses.

Interpolating an ontology $\mathcal{O}$ for a signature Σ amounts to capturing the information in $\mathcal{O}$ that involves only the names in Σ. This is realized by *forgetting* from $\mathcal{O}$ the names that do not belong to Σ (Koopmann 2015; Ludwig and Konev 2014). Uniform interpolation is therefore also referred to as *(deductive) forgetting* of the names in $\mathcal{F} = \mathsf{sig}(\mathcal{O}) \backslash \Sigma$. This means the uniform interpolant does not contain names in $\mathcal{F}$. In the case of logical difference, the signature to be forgotten is the set $\mathcal{F} = \mathsf{sig}(\mathcal{O}_2) \backslash \mathsf{sig}(\mathcal{O}_1)$.

The following very simple example illustrates the main notions in our approach. Consider the ontologies

$$\mathcal{O}_1 = \{A \sqsubseteq B, C \sqsubseteq E\}, \quad \text{and}$$
$$\mathcal{O}_2 = \{A \sqsubseteq B, C \sqsubseteq E, B \sqsubseteq C, F \sqsubseteq C, A \sqsubseteq F\}$$

with common symbols $\Sigma = \{A, B, C, E\}$. The logical difference is the set

$$\mathsf{Diff}(\mathcal{O}_1, \mathcal{O}_2) = \{B \sqsubseteq C, A \sqsubseteq C, B \sqsubseteq E, A \sqsubseteq E\}.$$

To compute the UI difference we forget $\mathcal{F} = \{F\}$ from $\mathcal{O}_2$ to get the uniform interpolant $\mathcal{V}_2 = \{A \sqsubseteq B, C \sqsubseteq E, B \sqsubseteq C, A \sqsubseteq C\}$. Eliminating the axioms inferred by $\mathcal{O}_1$ we get

$$\mathsf{UI\text{-}Diff}(\mathcal{O}_1, \mathcal{O}_2) = \{B \sqsubseteq C, A \sqsubseteq C\}.$$

We can view these as the strongest witnesses gained from $\mathcal{O}_1$ to $\mathcal{O}_2$. $B \sqsubseteq C$ is an explicit witness because it is an axiom

in $\mathcal{O}_2$, while $A \sqsubseteq C$ is an implicit (inferred) witness. Since the problem is Boolean, the logical difference has a finite representation, and other methods using a generate-and-test approach can compute the difference set. Our method guarantees a finite UI-based logical difference is found for all $\mathcal{ALC}$-ontologies, thus presenting a general method to track the difference of $\mathcal{ALC}$-ontologies.

Drawbacks of Existing Forgetting Tools

Practical tools for forgetting include FAME 1.0 (Zhao and Schmidt 2018a), LETHE (Koopmann 2015) and the tool developed by (Ludwig and Konev 2014). For several reasons these tools, in their present form, have not been found suited for the logical difference task.

FAME 1.0 is a Java-based implementation of a forgetting method developed in the work of (Zhao and Schmidt 2015; 2016). The method computes *semantic solutions* of forgetting for the description logic $\mathcal{ALCOIH}$. While the tool has proved to be very fast and thus might have been a good option to realize Step (1) of the logical difference algorithm — using FAME 1.0, forgetting solutions could be computed within seconds in more than 70% of the test cases (Zhao and Schmidt 2018a) — experimental results have shown that there are at least two reasons why FAME 1.0 is not best suited for the task. One reason is that extra expressivity may be required to capture semantic solutions of forgetting; often the witnesses are expressed using operators not in the language of the given ontologies, and were found to be syntactically complex and intuitively unreadable. In particular, our evaluation of FAME 1.0 on different versions of the SNOMED CT and NCIt ontologies showed that the computed forgetting solutions contained a large number of deeply nested axioms of the form $C \sqsubseteq \forall r^-.\exists s.D$, involving the inverse role operator, which is not part of $\mathcal{ALC}$.

A more critical reason is that semantic solutions of forgetting do not always exist for $\mathcal{ALC}$-ontologies, even for only acyclic cases or with extended expressivity (Konev *et al.* 2013). This is different from uniform interpolation, where using fixpoints or auxiliary concept definers, uniform interpolants can always be found for $\mathcal{ALC}$ (Koopmann and Schmidt 2013b), whereas semantic solutions do not always exist for $\mathcal{ALC}$, even with extra expressivity in the target language. This means that FAME 1.0 is incomplete, and there are forgetting problems where FAME 1.0 is unable to find a solution. For these cases, the 'witnesses' identified by an external DL reasoner from the forgetting solution are not those as desired, as they may contain newly introduced definer names that do not belong to the common signature of the given two ontologies.

LETHE and the tool developed by (Ludwig and Konev 2014) use resolution-based methods to compute uniform interpolants for $\mathcal{ALC}$ TBoxes, and in the case of LETHE, also several extensions of $\mathcal{ALC}$ TBoxes (Koopmann and Schmidt 2013b; 2014; Koopmann 2015). Since $\mathcal{ALC}$ does not have the uniform interpolation property (Lutz and Wolter 2011), LETHE incorporates fixpoints in the target language to ensure that the uniform interpolants can always be finitely represented, and the tool developed by (Ludwig and Konev 2014) solves the problem by adapting a depth-bounded version of its core algorithm. While these make the tools (uniform interpolation) complete for $\mathcal{ALC}$-ontologies, it has been found that neither of them can be used for the logical difference problem. A current drawback of LETHE is lack of speed and inability to handle very large ontologies. A preliminary comparison of LETHE and a previous version of FAME 1.0 showed that LETHE was considerably slower than FAME 1.0 (Alassaf and Schmidt 2017). A casual test showed that LETHE could not handle ontologies as large as SNOMED CT and NCIt, getting stuck in the forgetting loop. We conjecture that this is because LETHE introduces definer names[1] in a systematic and exhaustive manner, which often leads to an exponential explosion of axioms in the forgetting process; see (Koopmann 2015) for a comprehensive analysis of this plus examples.

Problems with the tool of (Ludwig and Konev 2014), besides not being as fast as FAME 1.0 either, include that the tool can only eliminate concept names, but not role names, and only depth-bounded uniform interpolants are computed although this did not seem to have been a restriction in the authors' evaluation.

Our Method for Computing Uniform Interpolants for $\mathcal{ALC}$-Ontologies

In this section, we introduce a new method designed to compute uniform interpolants for very large $\mathcal{ALC}$-ontologies. It contains two main ingredients: a calculus for concept name elimination and a calculus for role name elimination. Both operate on axioms in clausal normal form.[2]

Calculus for Concept Name Elimination

The calculus for eliminating a concept name $A \in \mathsf{sig_C}(\mathcal{N})$ from a set $\mathcal{N}$ of clauses includes three types of rules: a pair of purify rules, a pair of Ackermann rules, and a combination rule. The *purify rules* state that, if $\mathcal{N}$ is positive (negative) w.r.t. A, then A is eliminated by replacing every occurrence of A in $\mathcal{N}$ by $\top$ ($\bot$). This is referred to as *purification*.

Let the *pivot* be the name in $\mathcal{F}$ under current consideration for forgetting. The *Ackermann rules* are:

$$\frac{\mathcal{N}(\mathcal{S}^-), \alpha_1 \sqcup \mathcal{S}, ..., \alpha_n \sqcup \mathcal{S} \quad \text{(premises)}}{\mathcal{N}(\mathcal{S}^-)^{\mathcal{S}}_{\neg\alpha_1 \sqcup ... \sqcup \neg\alpha_n} \quad \text{(conclusion)}} \tag{1}$$

$$\frac{\mathcal{N}(\mathcal{S}^+), \neg\mathcal{S} \sqcup \alpha_1, ..., \neg\mathcal{S} \sqcup \alpha_n \quad \text{(premises)}}{\mathcal{N}(\mathcal{S}^+)^{\mathcal{S}}_{\alpha_1 \sqcap ... \sqcap \alpha_n} \quad \text{(conclusion)}} \tag{2}$$

where $\mathcal{S} \in \mathsf{N_C}$ is the pivot, the α_i ($1 \leq i \leq n$) are concepts that do not contain $\mathcal{S}$, $\mathcal{N}(\mathcal{S}^+)$ ($\mathcal{N}(\mathcal{S}^-)$) denotes the clause set $\mathcal{N}$ being positive (negative) w.r.t. $\mathcal{S}$, and $\mathcal{N}^{\mathcal{S}}_{\alpha}$ denotes the clause set obtained from $\mathcal{N}$ by substituting α for every occurrence of $\mathcal{S}$ in $\mathcal{N}$. The Ackermann rules replace

[1] *Definer names* are fresh concept names introduced to facilitate the normalization of an ontology (Koopmann and Schmidt 2013b).

[2] A *clause* in $\mathcal{ALC}$ is a disjunction of literals, which are concepts of the form A, $\neg A$, $\exists r.C$ and $\forall r.C$. Clauses are obtained from axioms using the standard clausal form transformations.

the axioms above the line, namely the *premises*, by those under the line, namely the *conclusion*. The rules are applicable (to $\mathcal{N}$ to eliminate A) iff $\mathcal{N}$ has the form as above, which, however, is not always the case.

The purify rules and the Ackermann rules preserve semantic equivalence up to the remaining names, details and a proof of this can be found in (Zhao 2017). This means the conclusions of these rules are uniform interpolants of $\mathcal{O}$ for $\mathsf{sig}(\mathcal{O})\backslash\{A\}$.

LETHE and the tool of (Ludwig and Konev 2014) do not use the Ackermann rules for concept name elimination, but it has been found in many cases that, in order to eliminate a concept name, LETHE needs to introduce many auxiliary definer names, whereas our method can apply directly the Ackermann rules.

If A is not *purifiable* (i.e., cannot be eliminated by purification) or cannot be eliminated using the Ackermann rules, it is attempted to be eliminated using the *combination rule*, shown in Figure 1. The combination rule is applicable (to $\mathcal{N}$ to eliminate A) iff $\mathcal{N}$ is in A-reduced form.

Definition 4 (A**-Reduced Form).** *Let $A \in \mathsf{N_C}$ be the pivot. A clause is in* A-reduced form *if it is of the form $C \sqcup A$, $C \sqcup \neg A$, $C \sqcup \exists r.A$, $C \sqcup \exists r.\neg A$, $C \sqcup \forall r.A$ or $C \sqcup \forall r.\neg A$, where $r \in \mathsf{N_R}$ is any role name and C is a clause that does not contain A. A set $\mathcal{N}$ of clauses is in* A-reduced form *if every A-clause in $\mathcal{N}$ is in A-reduced form.*

The *A-reduced form* generalizes all basic forms in which a concept name $A \in \mathsf{N_C}$ could occur; A could occur (either positively or negatively) at the top level of a clause, or under a $\exists$-restriction or a $\forall$-restriction. Given a set $\mathcal{N}$ of clauses, not every A-clause in $\mathcal{N}$ is in A-reduced form; an A-clause not in A-reduced form has the form $C \sqcup \exists r.D$ or $C \sqcup \forall r.D$, where $r \in \mathsf{N_R}$ is a role name, (i) C is a clause that contains A, or (ii) $D \neq A$ is a concept that contains A. An A-clause (not in A-reduced form) can be transformed into A-reduced form using definer names: Let $\mathsf{N_D} \subset \mathsf{N_C}$ be a set of definer names disjoint from $\mathsf{sig_C}(\mathcal{N})$. Definer names are introduced as substitutes, incrementally replacing 'C' and 'D' in every A-clause not in A-reduced form until (i) and (ii) do not hold. A new clause $\neg\mathsf{D^d} \sqcup C$ ($\neg\mathsf{D^d} \sqcup D$) is added to $\mathcal{N}$ for each replaced C (D), where $\mathsf{D^d} \in \mathsf{N_D}$ is a fresh definer.

Let $\mathcal{N}$ be a set of clauses in A-reduced form. We use the notation $\mathcal{P}^+_\bigstar(A)$ and $\mathcal{P}^-_\bigstar(A)$ to denote the sets of the clauses of the form $C \sqcup A$ and $C \sqcup \neg A$, respectively. We use the notation $\mathcal{P}^+_\exists(A)$ and $\mathcal{P}^-_\exists(A)$ to denote the sets of the clauses of the form $C \sqcup \exists r.A$ and $C \sqcup \exists r.\neg A$, respectively. We use the notation $\mathcal{P}^+_\forall(A)$ and $\mathcal{P}^-_\forall(A)$ to denote the sets of the clauses of the form $C \sqcup \forall r.A$ and $C \sqcup \forall r.\neg A$, respectively. By $\mathcal{N}^{-A(-r)}$ we denote the set of clauses that do not contain A (r). By $\mathcal{P}^+(A)$ we denote the union of $\mathcal{P}^+_\bigstar(A)$, $\mathcal{P}^+_\exists(A)$ and $\mathcal{P}^+_\forall(A)$ (*positive premises*), and by $\mathcal{P}^-(A)$ the union of $\mathcal{P}^-_\bigstar(A)$, $\mathcal{P}^-_\exists(A)$ and $\mathcal{P}^-_\forall(A)$ (*negative premises*).

As with the Ackermann rules, the combination rule is a replacement rule that replaces the premises by its conclusion. The idea is to combine all positive premises $\mathcal{P}^+(A)$ with every negative premise $\alpha \in \mathcal{P}^-(A)$ (or to combine all negative premises $\mathcal{P}^-(A)$ with every positive premise $\alpha \in \mathcal{P}^+(A)$). The result of each combination is a set of clauses that does not contain A, denoted by $\textbf{BLOCK}(\mathcal{P}^+(A), \alpha)$ (or $\textbf{BLOCK}(\mathcal{P}^-(A), \alpha)$). The conclusion of the rule, which is also the solution of forgetting $\{A\}$ from $\mathcal{N}$, is the union of the results obtained from each combination.

Lemma 1. *The combination rule in Figure 1 preserves all logical consequences up to the names in $\mathsf{sig}(\mathcal{N})\backslash\{A\}$.*

Proof. The idea of the combination in Cases 1, 2, 3, 4 and 5 is basically that of Rules (1) and (2). The result in Case 6 follows directly from the premises. We prove Case 7 (Case 8 can be proved similarly). Assume a domain element d has a t-successor not satisfying A and a domain element d' has all its s-successors satisfying A, if $t = s$, then d and d' must be different elements. We prove Case 9. Assume a domain element d has all its s-successors satisfying A and has all its q-successors not satisfying A, if $s = q$, then it has all its s-successors satisfying $\bot$. □

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

This follows from soundness of the purify and the Ackermann rules, Lemma 1 and Lemma 2 stated below.

Calculus for Role Name Elimination

The calculus used by our method for eliminating a role name $r \in \mathsf{sig_R}(\mathcal{N})$ from a clause set $\mathcal{N}$ includes one *combination rule*, shown in Figure 2. The combination rule is applicable (to $\mathcal{N}$ to eliminate r) iff $\mathcal{N}$ is in r-reduced form.

Definition 5 (r**-Reduced Form).** *Let $r \in \mathsf{N_R}$ be the pivot. A clause is in* r-reduced form *if it is of the form $C \sqcup \exists r.D$ or $C \sqcup \forall r.D$, where C (D) is a clause (concept) that does not contain r. A set $\mathcal{N}$ of clauses is in* r-reduced form *if every r-clause in $\mathcal{N}$ is in r-reduced form.*

The *r-reduced form* generalizes all basic forms in which a role name $r \in \mathsf{N_R}$ could occur; r could occur under a $\exists$-restriction or a $\forall$-restriction. An r-clause is not in r-reduced form if 'C' or 'D' contains r. It can be transformed into r-reduced form using definer names as well: definer names are used as substitutes, incrementally replacing 'C' and 'D' in every r-clause not in r-reduced form neither of them contains r. A new clause $\neg\mathsf{D^d} \sqcup C$ ($\neg\mathsf{D^d} \sqcup D$) is added to $\mathcal{N}$ for each replaced C (D), where $\mathsf{D^d} \in \mathsf{N_D}$ is a fresh definer.

Lemma 2. *Using the definer name introduction, any set $\mathcal{N}$ of $\mathcal{ALC}$-clauses can be transformed into A- or r-reduced form. The transformation is polynomial and preserves all logical consequences up to $\mathsf{sig}(\mathcal{N})\backslash\{A\}$ or $\mathsf{sig}(\mathcal{N})\backslash\{r\}$.*

Proof. Definer name introduction is basically the standard structural transformation, which is polynomial. □

The idea of the combination rule for role elimination is analogous to that of the rule for concept elimination.

Lemma 3. *The combination rule in Figure 2 preserves all logical consequences up to the names in $\mathsf{sig}(\mathcal{N})\backslash\{r\}$.*

Proof. Same proof as for Cases 7 and 8 in Lemma 1. □

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

Proof. This follows from Lemmas 2 and 3. □

$$\mathcal{N}^{-\mathsf{A}}, \overbrace{C_1 \sqcup \mathsf{A}, \ldots, C_l \sqcup \mathsf{A}}^{\mathcal{P}^+_\bigstar(\mathsf{A})}, \overbrace{D_1 \sqcup \exists r_1.\mathsf{A}, \ldots, D_m \sqcup \exists r_m.\mathsf{A}}^{\mathcal{P}^+_\exists(\mathsf{A})}, \overbrace{E_1 \sqcup \forall s_1.\mathsf{A}, \ldots, E_n \sqcup \forall s_n.\mathsf{A}}^{\mathcal{P}^+_\forall(\mathsf{A})} \quad \text{(positive premises)}$$

$$\overbrace{F_1 \sqcup \neg\mathsf{A}, \ldots, F_{l'} \sqcup \neg\mathsf{A}}^{\mathcal{P}^-_\bigstar(\mathsf{A})}, \overbrace{G_1 \sqcup \exists t_1.\neg\mathsf{A}, \ldots, G_{m'} \sqcup \exists t_{m'}.\neg\mathsf{A}}^{\mathcal{P}^-_\exists(\mathsf{A})}, \overbrace{H_1 \sqcup \forall q_1.\neg\mathsf{A}, \ldots, H_{n'} \sqcup \forall q_{n'}.\neg\mathsf{A}}^{\mathcal{P}^-_\forall(\mathsf{A})} \quad \text{(negative premises)}$$

$$\begin{aligned}\mathcal{N}^{-\mathsf{A}}, &\textsc{Block}(\mathcal{P}^+_\bigstar(\mathsf{A}), \mathcal{P}^-_\bigstar(\mathsf{A})), \textsc{Block}(\mathcal{P}^+_\bigstar(\mathsf{A}), G_1 \sqcup \exists t_1.\neg\mathsf{A}), ..., \textsc{Block}(\mathcal{P}^+_\bigstar(\mathsf{A}), G_{m'} \sqcup \exists t_{m'}.\neg\mathsf{A}),\\ &\textsc{Block}(\mathcal{P}^+_\bigstar(\mathsf{A}), H_1 \sqcup \forall q_1.\neg\mathsf{A}), ..., \textsc{Block}(\mathcal{P}^+_\bigstar(\mathsf{A}), H_{n'} \sqcup \forall q_{n'}.\neg\mathsf{A}), \textsc{Block}(\mathcal{P}^-(\mathsf{A}), D_1 \sqcup \exists r_1.\mathsf{A}), \ldots, \quad \text{(conclusion)}\\ &\textsc{Block}(\mathcal{P}^-(\mathsf{A}), D_m \sqcup \exists r_m.\mathsf{A}), \textsc{Block}(\mathcal{P}^-(\mathsf{A}), E_1 \sqcup \forall s_1.\mathsf{A}), ..., \textsc{Block}(\mathcal{P}^-(\mathsf{A}), E_n \sqcup \forall s_n.\mathsf{A})\end{aligned}$$

Notation in the combination rule ($1 \leq j \leq m$, $1 \leq k \leq n$, $1 \leq j' \leq m'$, $1 \leq k' \leq n'$):

$\textsc{Block}(\mathcal{P}^-(\mathsf{A}), \alpha) = \textsc{Block}(\mathcal{P}^-_\bigstar(\mathsf{A}), \alpha) \cup \textsc{Block}(\mathcal{P}^-_\geq(\mathsf{A}), \alpha) \cup \textsc{Block}(\mathcal{P}^-_\leq(\mathsf{A}), \alpha)$, where $\alpha \in \mathcal{P}^+_\geq(\mathsf{A}) \cup \mathcal{P}^+_\leq(\mathsf{A})$

Case 1: $\textsc{Block}(\mathcal{P}^+_\bigstar(\mathsf{A}), \mathcal{P}^-_\bigstar(\mathsf{A}))$ denotes the set $\{C_1 \sqcup (F_1 \sqcap \ldots \sqcap F_{l'}), \ldots, C_l \sqcup (F_1 \sqcap \ldots \sqcap F_{l'})\}$.

Case 2: $\textsc{Block}(\mathcal{P}^+_\bigstar(\mathsf{A}), G_{j'} \sqcup \exists t_{j'}.\neg\mathsf{A})$ denotes the set $\{G_{j'} \sqcup \exists t_{j'}.(C_1 \sqcap \ldots \sqcap C_l)\}$.

Case 3: $\textsc{Block}(\mathcal{P}^+_\bigstar(\mathsf{A}), H_{k'} \sqcup \forall q_{k'}.\neg\mathsf{A}\}$ denotes the set $\{H_{k'} \sqcup \forall q_{k'}.(C_1 \sqcap \ldots \sqcap C_l)\}$.

Case 4: $\textsc{Block}(\mathcal{P}^-_\bigstar(\mathsf{A}), D_j \sqcup \exists r_j.\mathsf{A})$ denotes the set $\{D_j \sqcup \exists r_j.(F_1 \sqcap \ldots \sqcap F_{l'})\}$.

Case 5: $\textsc{Block}(\mathcal{P}^-_\bigstar(\mathsf{A}), E_k \sqcup \forall s_k.\mathsf{A})$ denotes the set $\{E_k \sqcup \forall s_k.(F_1 \sqcap \ldots \sqcap F_{l'})\}$.

Case 6: $\textsc{Block}(\mathcal{P}^-_\exists(\mathsf{A}), D_j \sqcup \exists r_j.\mathsf{A})$ denotes the sets $\{D_j \sqcup \exists r_j.\top\}$ and $\{G_1 \sqcup \exists t_1.\top, \ldots, G_{m'} \sqcup \exists t_{m'}.\top\}$.

Case 7: $\textsc{Block}(\mathcal{P}^-_\exists(\mathsf{A}), E_k \sqcup \forall s_k.\neg\mathsf{A})$ denotes the sets $\bigcup_{1 \leq j' \leq m'} \{G_{j'} \sqcup \exists t_{j'}.\top\}$ and $\bigcup_{1 \leq j' \leq m'} \{G_{j'} \sqcup E_k\}$ for any $t_{j'} = s_k$.

Case 8: $\textsc{Block}(\mathcal{P}^-_\forall(\mathsf{A}), D_j \sqcup \exists r_j.\mathsf{A})$ denotes the sets $\{D_j \sqcup \exists r_j.\top\}$ and $\bigcup_{1 \leq k' \leq n'} \{D_j \sqcup H_{k'}\}$ for any $q_{k'} = r_j$.

Case 9: $\textsc{Block}(\mathcal{P}^-_\forall(\mathsf{A}), E_k \sqcup \forall s_k.\mathsf{A})$ denotes the sets $\bigcup_{1 \leq k' \leq n'} \{E_k \sqcup H_{k'} \sqcup \forall s_k.\bot\}$ for any $q_{k'} = s_k$.

Figure 1: The combination rule for eliminating $\mathsf{A} \in \mathsf{sig_C}(\mathcal{N})$ from a set $\mathcal{N}$ of clauses in A-reduced form

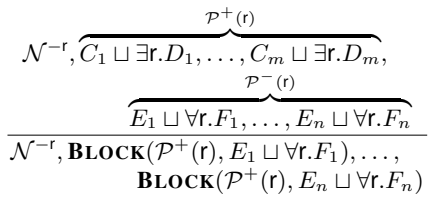
The Forgetting Process

The *input* to the forgetting method are a set $\mathcal{F}_\mathsf{R}$ of role names to be forgotten, a set $\mathcal{F}_\mathsf{C}$ of concept names to be forgotten, and a set cls of $\mathcal{ALC}$-clauses from which the names in $\mathcal{F}_\mathsf{R}$ and $\mathcal{F}_\mathsf{C}$ are to be eliminated. If a concept (role) name is successfully eliminated from cls, immediately it is removed from $\mathcal{F}_\mathsf{C}$ ($\mathcal{F}_\mathsf{R}$); otherwise it remains in $\mathcal{F}_\mathsf{C}$ ($\mathcal{F}_\mathsf{R}$).

The *forgetting process* in the method consists of the concept forgetting process and the role forgetting process. Our method defaults to performing role forgetting first, because then the definer names introduced during the role forgetting process can be eliminated as part of subsequent concept forgetting; otherwise the method may need to perform concept forgetting again (to eliminate the definer names possibly introduced in the previous role forgetting). The *role forgetting process* is an iteration of several elimination rounds in each of which a role name in $\mathcal{F}_\mathsf{R}$ is eliminated using the calculus for role name elimination, as described above.

The *concept forgetting process* consists of two iterations which are executed in sequence. The first iteration contains several elimination rounds in each of which a concept name in $\mathcal{F}_\mathsf{C}$ is attempted to be eliminated using purely the purify or the Ackermann rules. This iteration is intended to eliminate those concept names that currently can be eliminated without the help of definer names. In contrast to LETHE, our method introduces definer names in a *conservative* manner (introduces as few definer names as is possible). This makes our method considerably faster than LETHE, and allows our method to use less memory than LETHE does during the forgetting process. The second iteration contains several elimination rounds in which all remaining names in $\mathcal{F}_\mathsf{C}$ are eliminated using not only the purify and the Ackermann rules, but also the combination rule for concept elimination that may involve the introduction of definer names. It has been found that a name that could not be eliminated by the method may become eliminable once another name has been eliminated; see (Zhao 2017) for examples. We therefore impose a do-while loop on the iterations with the break condition checking if there were concept names eliminated in the previous loop. If so, the method repeats the iteration until there were no concept names eliminated in the previous loop or $\mathcal{F}_\mathsf{C}$ has become empty, i.e., all concept names in $\mathcal{F}_\mathsf{C}$ have been eliminated from cls.

Definer names, if introduced, are eliminated in exactly the same way as concept names in $\mathcal{F}_\mathsf{C}$ except that definer elimination is not always successful; it may fail when the original ontology contains cyclic dependencies over the forgetting signature; see (Zhao 2017) for an example. Cyclic cases can however be solved using fixpoints, but since fixpoints are not compatible with the OWL API and are at present not



$$\frac{\mathcal{N}^{-\mathsf{r}}, \overbrace{C_1 \sqcup \exists \mathsf{r}.D_1, \ldots, C_m \sqcup \exists \mathsf{r}.D_m}^{\mathcal{P}^+(\mathsf{r})}, \overbrace{E_1 \sqcup \forall \mathsf{r}.F_1, \ldots, E_n \sqcup \forall \mathsf{r}.F_n}^{\mathcal{P}^-(\mathsf{r})}}{\mathcal{N}^{-\mathsf{r}}, \mathbf{BLOCK}(\mathcal{P}^+(\mathsf{r}), E_1 \sqcup \forall \mathsf{r}.F_1), \ldots, \mathbf{BLOCK}(\mathcal{P}^+(\mathsf{r}), E_n \sqcup \forall \mathsf{r}.F_n)}$$

$\mathbf{BLOCK}(\mathcal{P}^+(\mathsf{r}), E_j \sqcup \forall \mathsf{r}.F_j)$ denotes the union of the following sets ($1 \leq j \leq n$):

1st-tier: $\bigcup_{1 \leq i \leq m} \{C_i \sqcup E_j\}$, for any D_i s.t. $D_i \sqcap F_j \sqsubseteq \bot$.

2nd-tier: $\bigcup_{1 \leq i_1 \leq i_2 \leq m} \{C_{i_1} \sqcup C_{i_2} \sqcup E_j\}$, for any D_{i_1}, D_{i_2} s.t. $D_{i_1} \sqcap D_{i_2} \sqcap F_j \sqsubseteq \bot$.

...

mth-tier: $\{C_1 \sqcup \ldots \sqcup C_m \sqcup E_j\}$, if $D_1 \sqcap \ldots \sqcap D_n \sqcap F_j \sqsubseteq \bot$.

Figure 2: The combination rule for eliminating $\mathsf{r} \in \mathsf{sig}_{\mathsf{R}}(\mathcal{N})$ from a set $\mathcal{N}$ of clauses in r-reduced form

supported by mainstream tools, we do not include them in the target language for practicality of our method. Without fixpoints, definer names would be infinitely introduced. Our method guarantees termination of cyclic cases by examining whether the axioms in the intermediary resulting set have identical syntactic patterns as those in the given one. If so, the method terminates immediately. The following theorem states termination and soundness of the method (as a consequence of Theorems 1 and 2).

Theorem 3. *Given any $\mathcal{ALC}$-ontology $\mathcal{O}$ and any forgetting signature $\mathcal{F} \subseteq \mathsf{sig}(\mathcal{O})$, our method always terminates and returns a finite set $\mathcal{O}'$ of axioms. If $\mathcal{O}'$ does not contain any introduced definer names, then our method is* successful *and $\mathcal{O}'$ is an $\mathcal{ALC}$-*uniform interpolant *of $\mathcal{O}$ for $\mathsf{sig}(\mathcal{O}) \backslash \mathcal{F}$.*

Crucial to the method are a set of equivalence-preserving simplification rules. These rules are applied throughout the forgetting process, ensuring that the clauses are always simpler representations, and no redundant clauses would be present in the forgetting solutions.

Evaluation of the Method

In order to investigate the practicality of our method, we implemented a prototype in Java using the OWL API Version 3.5.6[3] and evaluated it on a snapshot of the NCBO BioPortal repository[4] taken in March 2017 (Matentzoglu and Parsia 2017), containing 396 OWL API compatible ontologies. The corpus was previously used for the evaluation of FAME 1.0 in the work of (Zhao and Schmidt 2018a) where statistical information about these ontologies can be found.

The prototype was evaluated for three settings: forgetting 10% (199), 30% (597) and 50% (995) of concept and role names in the signature of each ontology. We compared the obtained results with those computed by LETHE and FAME 1.0. The names to be forgotten were randomly selected. The experiments were run on a desktop computer with an Intel Core i7-4790 processor, four cores running at up to 3.60 GHz, and 8 GB of DDR3-1600 MHz RAM. The experiments were run 10 times on each ontology and we averaged the results in order to verify the accuracy of our findings. A timeout of 1000 seconds was imposed on each run.

[3]https://github.com/owlcs/owlapi

[4]https://bioportal.bioontology.org/

Tool	$\%\mathcal{F}$	Time	T.O.	S. Rate	Extra
PROTOTYPE	10%	0.7s	1.8%	**96.4%**	1.8%
	30%	1.5s	2.5%	**93.7%**	3.8%
	50%	2.4s	4.8%	**90.1%**	5.1%
LETHE	10%	25.6s	8.8%	81.4%	9.8%
	30%	75.3s	19.0%	65.0%	16.0%
	50%	127.1s	29.3%	50.5%	20.2%
FAME 1.0	10%	0.6s	1.8%	87.2%	11.0%
	30%	1.3s	2.5%	73.5%	24.0%
	50%	2.0s	4.8%	66.7%	28.5%

Table 1: Results computed by our prototype, LETHE and FAME 1.0. (T.O.: Timeout, S. Rate: Success Rate)

The results obtained from the experiments are shown in Table 1. What is striking about the numbers in the table is that our prototype had considerably better success rates (>90%) over LETHE and FAME 1.0, whose success rates stayed between 50.5%–81.4% and 66.7%–87.2%, respectively.[5] As for speed, it can be seen that our prototype was comparable to FAME 1.0. LETHE was however struggling with speed; it was on average 50 times slower than other tools. LETHE's success rates were significantly affected by its speed; a large portion of its failures was due to the timeout. The column headed 'Extra' shows the percentages of cases where forgetting solutions contained undesired names or extra expressivity, i.e., FAME 1.0 introduced nominals to express semantic solutions, LETHE used fixpoint operators and our prototype used definer names to capture cyclic dependencies. What was unexpected is that forgetting solutions containing undesired names occurred rarely for our prototype, yet it occurred fairly frequently for LETHE, but these tools are supposed to show similar behaviour in this respect. This remains to be further investigated.

Logical Difference Case Study

In this section, we study how our prototype performs in practice for the forgetting task in the logical difference problem. Specifically, we used the prototype to compute the uniform

[5]It was defined in (Zhao and Schmidt 2018a) that FAME 1.0 is successful if it has eliminated all names in $\mathcal{F}$. We slightly adjust this in this paper: FAME 1.0 is successful if it has eliminated all names in $\mathcal{F}$, while not introducing extra expressivity.

interpolant $\mathcal{V}_2$ of $\mathcal{O}_2$ for the common signature of two given ontologies $\mathcal{O}_1$ and $\mathcal{O}_2$ that are being compared, and we used the DL reasoner Hermit (Glimm *et al.* 2014) to check if an axiom $\alpha \in \mathcal{V}$ is a witness of $\mathsf{UI\text{-}Diff}(\mathcal{O}_1, \mathcal{O}_2)$. To simplify the entire process, we implemented a tailor-made tool that integrated both our prototype for forgetting and HermiT for entailment checking. The *input* of the tool are two ontologies to be compared which must be specified as OWL/XML files, or as URLs pointing to OWL/XML files, and a file path specifying the location where the output returned by the tool is going to be saved. The *output* are a set $\mathcal{W}$ of UI-witnesses, a set $\mathcal{EW}$ of explicit UI-witnesses, and a set $\mathcal{IW}$ of implicit UI-witnesses, where $\mathcal{W} = \mathcal{EW} \cup \mathcal{IW}$. The witness sets are saved as standard ontologies (OWL/XML files) that can be used for in depth analysis or further processing.

We evaluated our difference tool on four versions of the SNOMED CT ontology and eight versions of the NCIt ontology. Hardware configurations remained unchanged.

The tool, along with the prototype for forgetting and the test data sets, can be downloaded via `http://www.cs.man.ac.uk/~schmidt/publications/aaai19/`.

Tracking Logical Difference between Different Versions of SNOMED CT Ontology

The four versions of the SNOMED CT ontology used in the evaluation were the following:

- SNOMED CT January 2017 International edition
- SNOMED CT July 2017 International edition
- SNOMED CT Australia edition
- SNOMED CT Canada edition

The SNOMED CT July 2017 International edition is an update of the January 2017 edition. The Australia and Canada editions are local extensions of the SNOMED CT July 2017 International edition.

Case	$\mathsf{UI\text{-}Diff}(\mathcal{O}_1, \mathcal{O}_2)$	$\#\mathcal{F}_\mathsf{C}$	$\#\mathcal{F}_\mathsf{R}$	$\#\mathsf{cls}$
1	UI-Diff(January, July)	10696	17	648080
2	UI-Diff(July, January)	614	0	630483
3	UI-Diff(July, Australia)	102880	15	1435778
4	UI-Diff(Australia, July)	6	0	648080
5	UI-Diff(July, Canada)	1700	0	650341
6	UI-Diff(Canada, July)	0	0	648080

Table 2: Logical difference and forgetting tasks for SNOMED CT

We consider the cases of computing the logical difference between six pairs of different versions of SNOMED CT; see Table 2. The first step is to compute a uniform interpolant from which the set of UI-witnesses is obtained. This requires to eliminate $\#\mathcal{F}_\mathsf{C}$ number of concept names and $\#\mathcal{F}_\mathsf{R}$ number of role names from $\#\mathsf{cls}$ number of clauses.

The results obtained from the forgetting are shown in Table 3, where Succ indicates whether or not the forgetting was successful (i.e., eliminated all names in $\mathcal{F}$, and the introduced definer names), FD denotes the forgetting duration (in seconds), $\#\mathcal{V}$ denotes the number of axioms in the computed uniform interpolant, $\#\mathsf{N_D}$ denotes the number of definer names introduced during the forgetting process, #PRF denotes the number of cases where a concept or definer name was eliminated by purification, #ACK denotes the number of cases where a concept or definer name was eliminated using the Ackermann rules, #CR denotes the number of cases where a concept or definer name was eliminated using the combination rule in Figure 1. An interesting case is that a concept or definer name had been eliminated when the prototype proceeded to its elimination round (it had been eliminated in the round of eliminating other names); #AE denotes the number of such cases. Observe that only in a small number of cases definer names were introduced; in more than 98.8% of the elimination rounds, a concept or definer name could be eliminated by purification or using the Ackermann rules, and without introducing any definer names. This explains why our prototype, introducing definer names in a conservative manner, is much faster than LETHE, which introduces definer names in a systematic and exhaustive manner.

Case	1	2	3	4	5	6
Succ	Yes	Yes	Yes	Yes	Yes	N/A
FD	68.8	7.6	3994	5.0	7.8	N/A
$\#\mathcal{V}$	636K	630K	928K	648K	648K	648K
$\#\mathsf{N_D}$	5991	1182	65	0	0	N/A
#PRF	8594	928	63158	3	1482	N/A
#ACK	3326	580	35329	3	215	N/A
#CR	1219	154	37	0	0	N/A
#AE	3548	134	4421	0	3	N/A

Table 3: Statistics of forgetting process and forgetting solutions

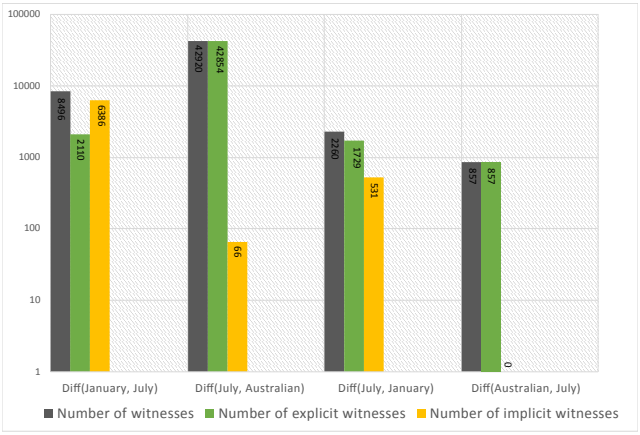


Figure 3: Logical difference between versions of SNOMED CT

The next step is to find from each computed uniform interpolant all witnesses of $\mathsf{UI\text{-}Diff}(\mathcal{O}_1, \mathcal{O}_2)$ using HermiT. Since the main focus of this paper is forgetting, we do not describe the entailment checking process. The logical difference between the four versions of the SNOMED CT ontology computed by the tool is shown in Figure 3. UI-Diff(July, Cannadian) and UI-Diff(Canada, July) disappear from the figure because there were no witnesses found in them, though the Canada edition had introduced 1700 new terms over the International July 2017 edition. Since the introduction of new

terms does not interfere with the meanings of existing terms, the Canada edition can be seen as a *safe extension* of the base edition. The Australia edition in contrast has had many term definitions of the base edition changed in the extension; there were 42920 witnesses found in UI-Diff(July, Australia). In this case, the involvement of a domain expert may be useful to validate the changes.

Tracking Logical Difference between Different Versions of NCIt Ontology

In order to make our findings and claims more convincing, we conducted another evaluation, where we used our logical difference tool to compute the logical difference between six pairs of different versions of the NCIt ontology. We used the most recent eight versions, consecutively released in 2018, as our test data. It was noticed from the SNOMED CT case that computing information gain was more challenging than computing information loss, as there were more names to be forgotten. Hence, in this test, we only considered the 'forward direction', computing the logical difference between an NCIt version and its most recent new release. The logical difference tasks performed in this test, together with the corresponding forgetting tasks, are shown in Figure 4. We omitted the case of Diff(18.05d, 18.06d) because these two versions were identical.

Case	UI-Diff($\mathcal{O}_1$, $\mathcal{O}_2$)	$\#\mathcal{F}_\mathsf{C}$	$\#\mathcal{F}_\mathsf{R}$	#cls
1	UI-Diff(18.01e, 18.02d)	3719	0	283326
2	UI-Diff(18.02d, 18.03d)	963	0	284806
3	UI-Diff(18.03d, 18.04e)	1294	0	286861
4	UI-Diff(18.04e, 18.05d)	2404	0	291462
5	UI-Diff(18.06d, 18.07e)	299	0	292091
6	UI-Diff(18.07e, 18.08e)	778	0	293426

Table 4: Logical difference and forgetting tasks for NCIt

Case	1	2	3	4	5	6
Succ	Yes	Yes	Yes	Yes	Yes	Yes
FD	3.7	2.0	2.6	11.0	2.0	2.3
$\#\mathcal{V}$	280K	284K	286K	290K	295K	293K
$\#\mathsf{N_D}$	253	28	23	111	2	13
#PRF	3002	872	1153	2106	224	668
#ACK	799	77	130	277	72	110
#CR	117	38	32	111	5	11
#AE	54	4	2	21	0	2

Table 5: Statistics of forgetting process and forgetting solutions

The results obtained from the forgetting are shown in Table 5. The most encouraging result is that our prototype was successful in all cases and uniform interpolants were computed within a few seconds. In more than 97% of the cases, a concept or definer name could be eliminated by purification or using the Ackermann rules, and without introducing definer names. This shows once again our method is practical.

The logical difference between versions of the NCIt ontology computed by our tool is shown in Figure 4. It can be observed that the numbers of witnesses in the NCIt cases

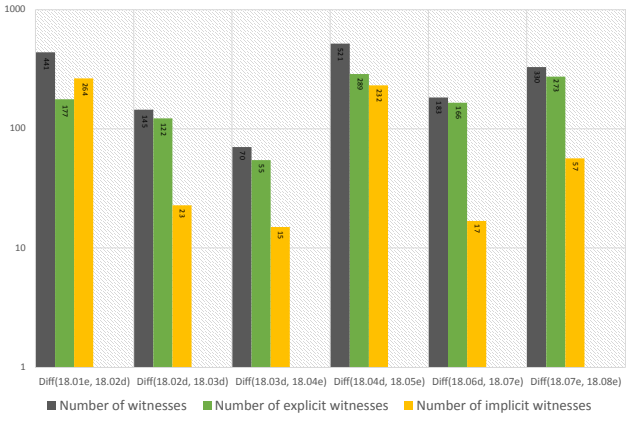


Figure 4: Logical difference between versions of NCIt

were notably smaller than those of the witnesses in the SNOMED CT cases (i.e., NCIt is evolving more gradually). This is probably because SNOMED CT is updated on a half-yearly basis, whereas NCIt is updated on a monthly basis (more frequent changes implies fewer changes), and also because SNOMED CT contains country specific extensions where significant adjustments can be expected.

Conclusion and Future Work

In this paper we have introduced a logical difference analysis tool able to track the changes in very large ontologies. As the forgetting step in the algorithm was a bottleneck with existing tools, we have developed a highly optimized deductive forgetting method and tool. For the first time it is possible to compute the logical difference between two ontologies of the size of SNOMED CT and NCIt in feasible time. This provides ontology engineers with a practical tool to analyze and maintain changes in their ontologies.

As the external reasoners have performance issues for entailment checking, a new approach to computing logical difference that obviates the need for a reasoner will be investigated in future work.

Although there is no restriction on the method, the tool handles only ontologies with TBoxes. To handle also ABoxes, the forgetting solutions may contain disjunctive concept assertions, e.g., axioms of the form $C(a) \sqcup D(b)$, where a, b are nominals (Koopmann and Schmidt 2015b). Such assertions cannot be encoded using the OWL API and thus cannot be saved as OWL/XML files. This also means subsequent entailment checking can only be performed with DL reasoners allowing disjunctive concept assertions. In these cases, a feasible solution would be to approximate the uniform interpolants and witness sets.

Acknowledgments

We would like to thank the reviewers for useful comments and good suggestions. This research was funded by EPSRC IAA Project 204 and Babylon Health.

References

R. Alassaf and R. A. Schmidt. A Preliminary Comparison of the Forgetting Solutions Computed using SCAN, LETHE and FAME In *Proc. SOQE'17*, volume 2013 of *CEUR Workshop Proceedings*, pages 21–26. CEUR-WS.org, 2017.

B. Glimm, I. Horrocks, B. Motik, G. Stoilos and Z. Wang. HermiT: An OWL 2 Reasoner. *J. Autom. Reasoning*, 53(3):245–269, 2014.

R. S. Gonçalves, B. Parsia and U. Sattler. Ecco: A Hybrid Diff Tool for OWL 2 ontologies In *Proc. OWLED'12*, CEUR-WS.org, 2012.

F. W. Hartel and S. de Coronado and R. Dionne and G. Fragoso and J. Golbeck. Modeling a description logic vocabulary for cancer research. *Journal of Biomedical Informatics*, 38(2):114–129, 2005.

E. Jiménez-Ruiz and B. C. Grau and I. Horrocks and R. B. Llavori. Supporting concurrent ontology development: Framework, algorithms and tool *Data Knowl. Eng.*, 70(1):146–164, 2011.

M. C. A. Klein and D. Fensel and A. Kiryakov and D. Ognyanov. Ontology Versioning and Change Detection on the Web In *Proc. EKAW'02*, volume 2473 of *LNCS*, pages 197–212. Springer, 2002.

B. Konev, D. Walther, and F. Wolter. The logical difference problem for description logic terminologies. In *Proc. IJCAR'08*, volume 5195 of *LNCS*, pages 259–274. Springer, 2008.

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.

B. Konev, D. Walther, and F. Wolter. Forgetting and Uniform Interpolation in Extensions of the Description Logic EL In *Proc. DL'09*, volume 477 of CEUR Workshop Proceedings, CEUR-WS.org, 2009.

B. Konev and M. Ludwig and D. Walther and F. Wolter The Logical Difference for the Lightweight Description Logic $\mathcal{EL}$. *J. Artif. Intell. Res.*, 44:633–708, 2012.

B. Konev, C. Lutz, D. Walther, and F. Wolter. Model-theoretic inseparability and modularity of description logic ontologies. *Artif. Intell.*, 203:66–103, 2013.

P. Koopmann and R. A. Schmidt. Uniform interpolation of $\mathcal{ALC}$-ontologies using fixpoints. In *Proc. FroCoS'13*, volume 8152 of *LNCS*, pages 87–102. Springer, 2013.

P. Koopmann and R. A. Schmidt. Count and forget: Uniform interpolation of $\mathcal{SHQ}$-ontologies. In *Proc. IJCAR'14*, volume 8562 of *LNCS*, pages 434–448. Springer, 2014.

P. Koopmann and R. A. Schmidt. Uniform interpolation and forgetting for $\mathcal{ALC}$ ontologies with ABoxes. In *Proc. AAAI'15*, pages 175–181. AAAI Press, 2015.

P. Koopmann. *Practical Uniform Interpolation for expressive Description Logics*. PhD thesis, University of Manchester, UK, 2015.

P. Kremen and M. Smid and Z. Kouba OWLDiff: A Practical Tool for Comparison and Merge of OWL Ontologies In *Proc. DEXA'11*, pages 229–233. IEEE Computer Society, 2011.

F. Lin and R. Reiter. Forget It! In *Proc. AAAI Fall Symposium on Relevance*, pages 154–159. AAAI Press, 1994.

M. Ludwig and B. Konev. Practical uniform interpolation and forgetting for $\mathcal{ALC}$ TBoxes with applications to logical difference. In *Proc. KR'14*, pages 318–327. AAAI Press, 2014.

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.

C. Lutz, D. Walther, and F. Wolter. Conservative extensions in expressive description logics. In *Proc. IJCAI'07*, pages 453–458, 2007.

N. Matentzoglu and B. Parsia. BioPortal Snapshot 30.03.2017, March 2017.

N. F. Noy and M. A. Musen PROMPTDIFF: A Fixed-Point Algorithm for Comparing Ontology Versions. In *Proc. AAAI'02*. pages 744–750, AAAI Press, 2002.

A. Solimando and E. Jiménez-Ruiz and G. Guerrini Minimizing conservativity violations in ontology alignments: algorithms and evaluation. *Knowl. Inf. Syst.*, 51(3):775–819, 2017.

K. Spackman. SNOMED RT and SNOMED CT. Promise of an international clinical ontology *M.D. Computing 17*, 2000.

G. Stoilos, D. Geleta, S. Wartak, S. Hall, M. Khodadadi, Y. Zhao, G. Alghamdi and R. A. Schmidt Methods and Metrics for Knowledge Base Engineering and Integration In *Proc. WOP'18*, pages 72–86. CEUR-WS.org, 2018.

K. Wang, Z. Wang, R. W. Topor, J. Z. Pan, and G. Antoniou. Eliminating concepts and roles from ontologies in expressive descriptive logics. *Computational Intelligence*, 30(2):205–232, 2014.

Y. Zhang and Y. Zhou. Forgetting Revisited. In *Proc. KR'10*, pages 602–604. AAAI Press, 2010.

Y. Zhao and R. A. Schmidt. Concept forgetting in $\mathcal{ALCOI}$-ontologies using an Ackermann approach. In *Proc. ISWC'15*, volume 9366 of *LNCS*, pages 587–602. Springer, 2015.

Y. Zhao and R. A. Schmidt. Forgetting concept and role symbols in $\mathcal{ALCOIH}\mu^+(\triangledown, \sqcap)$ ontologies. In *Proc. IJCAI'16*, pages 1345–1352. IJCAI/AAAI Press, 2016.

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

Y. Zhao. *Automated Semantic Forgetting For Expressive Description Logics*. PhD thesis, University of Manchester, UK, 2017.