A Practical Approach to Forgetting in Description Logics with Nominals

Yizheng Zhao,[1] Renate A. Schmidt,[2] Yuejie Wang,[3] Xuanming Zhang,[4] Hao Feng[5]
[1]National Key Laboratory for Novel Software Technology, Nanjing University, China
[2]Department of Computer Science, The University of Manchester, UK
[3]School of Electronics Engineering and Computer Science, Peking University, China
[4]School of Computer Science, University of Nottingham Ningbo China, China
[5]School of Knowledge Engineering, North China University of Science and Technology, China
zhaoyz@nju.edu.cn, renate.schmidt@manchester.ac.uk, {kathywangyuejie, hao.feng0429}@gmail.com
zy21855@nottingham.edu.cn

Abstract

This paper investigates the problem of forgetting in description logics with nominals. In particular, we develop a practical method for forgetting concept and role names from ontologies specified in the description logic $\mathcal{ALCO}$, extending the basic $\mathcal{ALC}$ with nominals. The method always terminates, and is sound in the sense that the forgetting solution computed by the method has the same logical consequences with the original ontology. The method is so far the only approach to deductive forgetting in description logics with nominals. An evaluation of a prototype implementation shows that the method achieves a significant speed-up and notably better success rates than the LETHE tool which performs deductive forgetting for $\mathcal{ALC}$-ontologies. Compared to FAME, a semantic forgetting tool for $\mathcal{ALCOIH}$-ontologies, better success rates are attained. From the perspective of ontology engineering this is very useful, as it provides ontology curators with a powerful tool to produce views of ontologies.

Introduction

Forgetting is an ontology engineering technique that seeks to produce *views* of ontologies. This is achieved by eliminating from ontologie a subset of their signature, namely the *forgetting signature*, in such a way that all logical consequences up to the remaining signature are preserved. Forgetting is useful for many ontology engineering tasks such as reuse (Wang et al. 2014), alignment and merging (Wang et al. 2005), versioning (Klein and Fensel 2001), debugging (Ribeiro and Wassermann 2009), repair (Troquard et al. 2018), logical difference (Konev, Walther, and Wolter 2008; 2009; Ludwig and Konev 2014; Zhao et al. 2019), and related tasks (Bicarregui et al. 2001; Lang, Liberatore, and Marquis 2003; Ghilardi, Lutz, and Wolter 2006; Eiter et al. 2006; Grau and Motik 2012; Ludwig and Konev 2013).

Forgetting is basically a non-standard reasoning problem which can be defined deductively as the dual of *uniform interpolation* (Visser 1996; Lutz and Wolter 2011) or model-theoretically as *semantic forgetting* (Wang et al. 2014). Uniform interpolation preserves logical consequences and semantic forgetting preserves semantic equivalence over a particular signature. Uniform interpolation is thus a weaker notion of forgetting than semantic forgetting; the results of uniform interpolation (*uniform interpolants*), are in general weaker than those of semantic forgetting (*semantic solutions*). This means for a specific language semantic solutions are always uniform interpolants, but not the other way round.

Practical methods for computing uniform interpolants include the approach of LETHE (Koopmann and Schmidt 2013a; 2013b; 2014; 2015), the one developed by (Ludwig and Konev 2014), and the one by (Zhao et al. 2019). LETHE handles $\mathcal{ALCH}$, $\mathcal{SIF}$, $\mathcal{SHQ}$-TBoxes, and $\mathcal{ALC}$-ontologies. The latter two handle $\mathcal{ALC}$-TBoxes.

FAME (Zhao and Schmidt 2018) is presently the only semantic forgetting tool for ontologies. It is also the only semantic forgetting tool for description logics with nominals; it handles $\mathcal{ALCOIH}$-ontologies (Zhao and Schmidt 2016). In this approach, the target language is not the same as the source language, in particular, it is more expressive in order to capture semantic solutions. This is however not satisfactory for users like SNOMED CT (Spackman 2000) 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. Tracking logical difference between two ontology versions is one application where the target language should coincide with the source language (Zhao et al. 2019).

In this paper we consider catering for the situation where the description logic $\mathcal{ALCO}$ serves as both source and target languages. One option is to attempt to approximate the semantic solutions for $\mathcal{ALCO}$ inputs to $\mathcal{ALCO}$-ontologies. Actually there is current research concerned with reduction of expressiveness by approximation (Brandt et al. 2002). However, rather than going down that route, we develop a novel practical forgetting approach for $\mathcal{ALCO}$-ontologies. The approach is deductive but incorporates also techniques from semantic forgetting, called Ackermann's Lemma. The method always terminates, and is sound in the sense that the forgetting solution computed by the method has the same logical consequences with the original ontology. It is so far the only approach to *deductive forgetting* for description logics with nominals. An evaluation of the method over $\mathcal{ALCO}$-ontologies shows that it has similar speed as semantic forgetting with FAME, but with better success rates. An evaluation over $\mathcal{ALC}$ ontologies shows that the method is both significantly faster than deductive forgetting of LETHE and exhibits better success rates. From the perspective of ontology engineering this is very useful, as it provides ontology curators with a powerful tool to produce views of ontologies.

Proofs of all theorems and lemmas can be found in a long version of this paper, which can be downloaded via http://www.cs.man.ac.uk/~schmidt/publications/aaai20/.

$\mathcal{ALCO}$-Ontologies and Forgetting

Let $\mathsf{N_C}$, $\mathsf{N_R}$ and $\mathsf{N_I}$ be pairwise disjoint and countably infinite sets of respectively *concept names*, *role names* and *individual names* (*nominals*). *Concepts* in $\mathcal{ALCO}$ 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 $a \in \mathsf{N_I}$, $A \in \mathsf{N_C}$, $r \in \mathsf{N_R}$, and both C and D denote arbitrary concepts in $\mathcal{ALCO}$.

An $\mathcal{ALCO}$-ontology consists of a TBox and an ABox. A TBox is a finite set of axioms of the form $C \sqsubseteq D$ (*concept inclusions*), where C and D are concepts. An ABox is a finite set of axioms of the form $C(a)$ (*concept assertions*) and the form $r(a, b)$ (role assertions), where $a, b \in \mathsf{N_I}$, $r \in \mathsf{N_R}$, and C is a concept. In description logics with nominals, ABox assertions are superfluous, as they can be internalized as concept inclusions via nominals, namely $C(a)$ as $a \sqsubseteq C$ and $r(a, b)$ as $a \sqsubseteq \exists r.b$. Hence in this paper, we assume that an $\mathcal{ALCO}$-ontology is a finite set of concept inclusions.

The semantics of $\mathcal{ALCO}$ is defined in terms of an *interpretation* $\mathcal{I} = \langle \Delta^{\mathcal{I}}, \cdot^{\mathcal{I}} \rangle$, where $\Delta^{\mathcal{I}}$ denotes the *domain of the interpretation*, which is 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:

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

Let $\mathcal{I}$ be an interpretation. 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 $\mathcal{I} \models \mathcal{O}$.

Next, we formalize the notion of deductive forgetting. By $\mathsf{sig_C}(X)$ and $\mathsf{sig_R}(X)$ we denote respectively the sets of the concept names and the role names occurring in X, where X ranges over concepts, axioms, clauses, a set of axioms, and a set of clauses. We define $\mathsf{sig}(X) = \mathsf{sig_C}(X) \cup \mathsf{sig_R}(X)$.

Definition 1 (**Forgetting for $\mathcal{ALCO}$-Ontologies**). *Let $\mathcal{O}$ be an $\mathcal{ALCO}$-ontology and let $\mathcal{F} \subseteq \mathsf{sig}(\mathcal{O})$ be a set of concept names and role names. We say that an $\mathcal{ALCO}$-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 the forgetting solution $\mathcal{V}$ has exactly the same logical consequences with the original ontology $\mathcal{O}$ in the remaining signature $\mathsf{sig}(\mathcal{O}) \backslash \mathcal{F}$. $\mathcal{F}$ is called the *forgetting signature*. $\mathcal{V}$ can be seen as a *view* of $\mathcal{O}$ for $\mathsf{sig}(\mathcal{O}) \backslash \mathcal{F}$. Forgetting solutions (views) are unique up to logical equivalence, that is, if both $\mathcal{V}$ and $\mathcal{V}'$ are solutions of forgetting $\mathcal{F}$ from $\mathcal{O}$, then they are logically equivalent, though their representations may not be identical.

Normalization of $\mathcal{ALCO}$-Ontologies

Our method works with $\mathcal{ALCO}$-ontologies in *clausal normal form*. Clauses are obtained from corresponding axioms using the standard transformations based on logical equivalences such as $\neg\exists r.C \equiv \forall r.\neg C$, $\neg\forall r.C \equiv \exists r.\neg C$, and $\neg\neg C \equiv C$. By incrementally applying the standard transformations, any $\mathcal{ALCO}$-ontology can be transformed into a set of clauses (Zhao 2018).

Definition 2 (**Clausal Normal Form**). *A* literal *in $\mathcal{ALCO}$ is a concept of the form a, $\neg a$, A, $\neg A$, $\exists r.C$ and $\forall r.C$, where $a \in \mathsf{N_I}$, $A \in \mathsf{N_C}$, $r \in \mathsf{N_R}$, and C is an arbitrary concept. A* clause *in $\mathcal{ALCO}$ is a finite disjunction of literals. A clause is called an $\mathcal{S}$-*clause *if it contains an $\mathcal{S} \in \mathsf{N_C} \cup \mathsf{N_R}$.*

In the following we introduce two specialized normal form notions (based on clausal normal form), namely *A-reduced form* and *r-reduced form*. These two notions are important because they are used in the calculi of our method for concept name and role name elimination, respectively, described in detail in the next section.

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

The A-reduced form generalizes all basic forms of an A-clause in which a concept name could occur; a concept name could occur (either positively or negatively) at the top level of a clause, or under an $\exists$- or $\forall$-restriction.

Definition 4 (**r-Reduced Form**). *Let $\mathcal{N}$ be a set of clauses, and let $r \in \mathsf{sig_R}(\mathcal{N})$. A clause is in r-*reduced form *if it has 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. $\mathcal{N}$ is in r-*reduced form *iff all r-clause in $\mathcal{N}$ are in r-reduced form.*

The r-reduced form generalizes all basic forms of an r-clause in which a role name could occur; a role name could occur immediately under an $\exists$- or $\forall$-restriction.

Given a set $\mathcal{N}$ of clauses, not every clause in $\mathcal{N}$ is initially in appropriate reduced form. A/r-clauses not in A/r-reduced form can be transformed into the form by introducing *definer names* (or *definers* for short). Definers are auxiliary concept names externally introduced to facilitate the ontology normalization (Koopmann and Schmidt 2013b): let $\mathsf{N_D} \subset \mathsf{N_C}$ be a set of definers disjoint from $\mathsf{sig_C}(\mathcal{N})$. Definers are introduced as substitutes, incrementally replacing 'C' and 'D' until neither of them contains A/r. In addition, for each concept replacement, a new clause $\neg \mathsf{D} \sqcup C$ (D) is added to the set $\mathcal{N}$, where $\mathsf{D} \in \mathsf{N_D}$ is a fresh definer.

Theorem 1. *With definer introduction, any $\mathcal{ALCO}$-ontology can be transformed into A/r-reduced form. The transformation can be done in polynomial time, and preserves semantic equivalence up to the names in the original ontology.*

Example 1. *Consider the following clause set $\mathcal{N}$:*

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

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 clauses:

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

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

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

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

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

Definer reuse is an important feature of our method, i.e., definers are supposed not to be present in the forgetting solutions, and they must be eliminated from the solutions once the names in $\mathcal{F}$ have been eliminated, so introducing as few definers as possible is good for the efficiency of our method.

The Forgetting Method

Our method is basically a non-standard reasoning procedure consisting of two independent calculi: (i) a calculus for concept name elimination in $\mathcal{ALCO}$-ontologies and (ii) a calculus for role name elimination in $\mathcal{ALCO}$-ontologies. They are *non-trivially* extended from the calculi develop in the work of (Zhao et al. 2019) for respectively concept name and role name elimination in $\mathcal{ALC}$-TBoxes. The extended calculi can handle additionally nominals and ABox axioms.

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

Let $\mathcal{N}$ denote a set of $\mathcal{ALCO}$-clauses. Let $\mathcal{F} \subseteq \mathsf{sig}(\mathcal{N})$ denote the forgetting signature. The name in $\mathcal{F}$ under current consideration for forgetting is called the *pivot*.

The Calculus for Concept Name Elimination

The calculus for eliminating a concept name from a set $\mathcal{N}$ of clauses includes three rules: (i) two purify rules, and (ii) one combination rule. Each rule is applicable to specific cases.

The purify rules are applicable to cases where the pivot occurs only positively or only negatively in $\mathcal{N}$, i.e., the pivot is *pure* in $\mathcal{N}$. For the positive (negative) cases, the pivot is eliminated from $\mathcal{N}$ by substitution with $\top$ $(\bot)$. This is referred to as *purification*. The purify rules preserve semantic equivalence up to the remaining names (Ackermann 1935; Zhao 2018), and thus the clause set obtained from purification is a solution of forgetting the pivot from $\mathcal{N}$.

For other cases, i.e., cases where the pivot occurs both positively and negatively in $\mathcal{N}$, one can apply the combination rule, shown in Figure 2. Compared to the combination rule for concept name elimination in $\mathcal{ALC}$-TBoxes (Zhao et al. 2019), this rule (for $\mathcal{ALCO}$-ontologies) accommodates nominals together with regular concepts; this means that the concepts C_i, D_j, E_k, $F_{i'}$, $G_{j'}$ and $H_{k'}$ in the rule can be any regular concepts and nominals. The combination rule is applicable to $\mathcal{N}$ to eliminate A iff $\mathcal{N}$ is in *A-reduced form*; see Definition 3. Clauses in A-reduced form have six distinct forms, classified into two types, namely *positive premises* and *negative premises*, denoted respectively by $\mathcal{P}^+(A)$ and $\mathcal{P}^-(A)$; see Table 1. Positive premises are premises where the pivot occurs only positively, and negative premises are premises where the pivot occurs only negatively.

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

Figure 1: Different forms of clauses in A-reduced form

The fundamental idea of the combination rule, in a nutshell, is to *combine* all positive premises with every negative one, or to *combine* all negative premises with every positive one; this is how the name 'combination' comes from. The result of each combination is a finite set of clauses, denoted by $\mathsf{Block}(\mathcal{P}^+(A), \alpha)$ when $\alpha \in \mathcal{P}^-(A)$, or $\mathsf{Block}(\mathcal{P}^-(A), \alpha)$ when $\alpha \in \mathcal{P}^+(A)$. The *conclusion* of the rule, which is the

$$\frac{\begin{array}{c}\mathcal{N}^{-\mathsf{A}}, \overbrace{C_1 \sqcup \mathsf{A}, \ldots, C_l \sqcup \mathsf{A}}^{\mathcal{P}^+_{\mho}(\mathsf{A})}, \overbrace{D_1 \sqcup \exists r_1.\mathsf{A}, \ldots, D_m \sqcup \exists r_m.\mathsf{A}}^{\mathcal{P}^+_{\exists}(\mathsf{A})}, \overbrace{E_1 \sqcup \forall s_1.(\mathsf{A} \sqcup U_1), \ldots, E_n \sqcup \forall s_n.(\mathsf{A} \sqcup U_n)}^{\mathcal{P}^+_{\forall}(\mathsf{A})} \\ \overbrace{F_1 \sqcup \neg\mathsf{A}, \ldots, F_{l'} \sqcup \neg\mathsf{A}}^{\mathcal{P}^-_{\mho}(\mathsf{A})}, \overbrace{G_1 \sqcup \exists t_1.\neg\mathsf{A}, \ldots, G_{m'} \sqcup \exists t_{m'}.\neg\mathsf{A}}^{\mathcal{P}^-_{\exists}(\mathsf{A})}, \overbrace{H_1 \sqcup \forall q_1.(\neg\mathsf{A} \sqcup W_1), \ldots, H_{n'} \sqcup \forall q_{n'}.(\neg\mathsf{A} \sqcup W_{n'})}^{\mathcal{P}^-_{\forall}(\mathsf{A})}\end{array}}{\begin{array}{c}\mathcal{N}^{-\mathsf{A}}, \mathsf{Block}(\mathcal{P}^+_{\mho}(\mathsf{A}), \mathcal{P}^-_{\mho}(\mathsf{A})), \mathsf{Block}(\mathcal{P}^+_{\mho}(\mathsf{A}), G_1 \sqcup \exists t_1.\neg\mathsf{A}), ..., \mathsf{Block}(\mathcal{P}^+_{\mho}(\mathsf{A}), G_{m'} \sqcup \exists t_{m'}.\neg\mathsf{A}), \\ \mathsf{Block}(\mathcal{P}^+_{\mho}(\mathsf{A}), H_1 \sqcup \forall q_1.(\neg\mathsf{A} \sqcup W_1)), ..., \mathsf{Block}(\mathcal{P}^+_{\mho}(\mathsf{A}), H_{n'} \sqcup \forall q_{n'}.(\neg\mathsf{A} \sqcup W_{n'})), \mathsf{Block}(\mathcal{P}^-(\mathsf{A}), D_1 \sqcup \exists r_1.\mathsf{A}), \ldots, \\ \mathsf{Block}(\mathcal{P}^-(\mathsf{A}), D_m \sqcup \exists r_m.\mathsf{A}), \mathsf{Block}(\mathcal{P}^-(\mathsf{A}), E_1 \sqcup \forall s_1.(\mathsf{A} \sqcup U_1)), ..., \mathsf{Block}(\mathcal{P}^-(\mathsf{A}), E_n \sqcup \forall s_n.(\mathsf{A} \sqcup U_n))\end{array}}$$

Notation in the combination rule ($1 \le i \le l, 1 \le j \le m, 1 \le k \le n, 1 \le i' \le l', 1 \le j' \le m', 1 \le k' \le n'$):
C_i, D_j, E_k, $F_{i'}$, $G_{j'}$ and $H_{k'}$ denote arbitrary concepts; r_j, s_k, $t_{j'}$ and $q_{k'}$ denote arbitrary role names.

$\mathsf{Block}(\mathcal{P}^-(\mathsf{A}), \alpha) = \mathsf{Block}(\mathcal{P}^-_{\mho}(\mathsf{A}), \alpha) \cup \mathsf{Block}(\mathcal{P}^-_{\exists}(\mathsf{A}), \alpha) \cup \mathsf{Block}(\mathcal{P}^-_{\forall}(\mathsf{A}), \alpha)$, where $\alpha \in \mathcal{P}^+_{\exists}(\mathsf{A}) \cup \mathcal{P}^+_{\forall}(\mathsf{A})$

Case 1: $\mathsf{Block}(\mathcal{P}^+_{\mho}(\mathsf{A}), \mathcal{P}^-_{\mho}(\mathsf{A}))$ 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: $\mathsf{Block}(\mathcal{P}^+_{\mho}(\mathsf{A}), G_{j'} \sqcup \exists t_{j'}.\neg\mathsf{A})$ denotes the set $\{G_{j'} \sqcup \exists t_{j'}.(C_1 \sqcap \ldots \sqcap C_l)\}$.

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

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

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

Case 6: $\mathsf{Block}(\mathcal{P}^-_{\exists}(\mathsf{A}), D_j \sqcup \exists r_j.\mathsf{A})$ 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: $\mathsf{Block}(\mathcal{P}^-_{\exists}(\mathsf{A}), E_k \sqcup \forall s_k.(\mathsf{A} \sqcup U_k))$ denotes the sets $\bigcup\limits_{1 \le j' \le m'} \{G_{j'} \sqcup \exists t_{j'}.\top\}$ and

$\bigcup\limits_{1 \le j' \le m'} \{G_{j'} \sqcup E_k \sqcup \exists s_k.U_k\}$ for any $t_{j'} = s_k$.

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

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

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

solution of the forgetting $\{A\}$ from $\mathcal{N}$, is the union of the results obtained from each combination.

Lemma 1. *The combination rule in Figure 2 is sound.*

Proof (sketch). The idea of the combination in Cases 1, 2, 3, 4 and 5 is basically that of Ackermann's Lemma which preserves semantic equivalence. The conclusion of Case 6 follows directly from the premises. We prove Case 7. Assume a domain element d has a t-successor not satisfying A and a domain element d' has all its s-successors satisfying A or U, if $t = s$ and $\mathsf{d} = \mathsf{d}'$, then d (d') must have a s-successor satisfying U. Case 8 can be proved similarly. We prove Case 9. Assume a domain element d has all its s-successors satisfying A or U, and has all its q-successors not satisfying A, if $s = q$, then all its s-successors satisfy U.

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

Proof. The calculus is sound in the sense that the output clause set $\mathcal{N}^{-\mathsf{A}}$ returned by the calculus has the same logical consequences with the input clause set $\mathcal{N}$ in the remaining signature $\mathsf{sig}(\mathcal{N})\backslash\{\mathsf{A}\}$ This follows from Theorem 1, Lemma 1, and soundness of the purify rules. □

Example 2. 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 $\mathcal{N}$. Since $\mathcal{N}$ is not pure, we apply the combination rule. 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 third step is to apply the combination rule (Case 4) to Clauses 2 and 3 to eliminate the introduced definer D, yielding the forgetting solution $\{1.C \sqcup \exists r.\top, 2.\neg a \sqcup \exists s.C\}$.

Compared to the calculus of (Zhao et al. 2019) for $\mathcal{ALC}$-TBoxes, this calculus can handle additionally nominals and ABoxes; it handles $\mathcal{ALCO}$-ontologies. However, we have found that even for $\mathcal{ALC}$-TBoxes only, there are cases that, at present, can only be solved by our calculus, but not by the calculus of (Zhao et al. 2019); see the following example.

Example 3. Let $\mathcal{N} = \{1.C_1 \sqcup \exists r.A, 2.C_2 \sqcup \forall r.(\neg A \sqcup B)\}$. We consider the case of forgetting $\{A\}$ from $\mathcal{N}$. Following the calculus of (Zhao et al. 2019), the first step is to transform $\mathcal{N}$ into A-reduced form, wherein a fresh definer $\mathsf{D} \in \mathsf{N_D}$ is introduced to replace the concept $\neg A \sqcup B$, and a new clause $\neg D \sqcup \neg A \sqcup B$ is added to $\mathcal{N}$. Then we apply the combination rule in the calculus of (Zhao et al. 2019) (Case 4) to $\mathcal{N} = \{1.C_1 \sqcup \exists r.A, 2.C_2 \sqcup \forall r.\mathsf{D}, 3.\neg\mathsf{D} \sqcup \neg A \sqcup B\}$ to eliminate A, yielding an intermediate result $\mathcal{N}' = \{1.C_1 \sqcup$

$\exists r.\mathsf{D}, 2.C_2 \sqcup \forall r.(\neg\mathsf{D} \sqcup B)\}$. Observe that the clauses in $\mathcal{N}'$ have the same syntactic patterns as those in the original $\mathcal{N}$. This means eliminating the introduced definer D from $\mathcal{N}'$ would introduce a fresher definer, and would similarly yield a set of clauses that have the same syntactic patterns as those in $\mathcal{N}'$; definers would thus be infinitely introduced and the forgetting process would never terminate. With the present calculus however, the problem can be solved by directly applying the combination rule (Case 7) to $\mathcal{N}$, yielding a forgetting solution $\mathcal{N}^{-\mathsf{A}} = \{1.C_1 \sqcup \exists r.\top, 2.C_1 \sqcup C_2 \sqcup \exists r.B\}$.

Usage of the Forgetting Method

The input to our method are an $\mathcal{ALCO}$-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 $\mathcal{ALCO}$, only the $\mathcal{ALCO}$-fragment is taken. This can be done by removing from $\mathcal{O}$ axioms not expressible in $\mathcal{ALCO}$. Which names are to be forgotten is determined by the user and their application demands. Our method can eliminate names in any 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 of the method consists of five main phases: (i) transformation of $\mathcal{O}$ into a set $\mathcal{N}$ of clauses, (ii) role name elimination, (iii) concept name elimination, (iv) definer name elimination, and (v) transformation of the resulting set $\mathcal{N}'$ into an ontology $\mathcal{V}$.

(i) and (v) can be done using the standard transformations. (ii) is an iteration of several rounds in which the role names in $\mathcal{F}$ are eliminated using the calculus for role name elimination. (iii) includes two iterations that are executed in sequence. The first iteration consists of several rounds in which the concept names in $\mathcal{F}$ are eliminated using *only* the purify rules. This means in this iteration, only those 'pure' concept names are eliminated. This allows concept names to be eliminated *cheaply* (because definers are not introduced during purification). The second iteration consists of several rounds in which the remaining concept names in $\mathcal{F}$ are eliminated using not only the purify rules, but also the combination rule. In this iteration, all remaining concept names in $\mathcal{F}$ are eliminated, but definers may be introduced.

The forgetting solutions should not contain definers, since these are not part of the desired signature. Our method eliminates definers (if they were introduced) basically in the same manner as regular concept names. The differences are that: (i) when eliminating the introduced definers, the method may need to introduce new definers (if we used the combination rule to eliminate existing ones), this is however done in a *restricted* manner: the method only introduces fewer definers than the existing ones; otherwise, the method will only use the purify rules to eliminate existing definers. (ii) Definers are not guaranteed to be all eliminated; the elimination may fail when the original ontology contains cyclic dependencies over the names in $\mathcal{F}$.

For example, the solution of forgetting $\{A\}$ from the ontology $\{\neg A \sqcup \exists r.A\}$ (cyclic w.r.t. A) is $\{\neg\mathsf{D}_1 \sqcup \exists r.\mathsf{D}_1\}$, where $\mathsf{D}_1 \in \mathsf{N_D}$ is a fresh definer. It is easy to see that eliminating D_1 from the intermediary solution would yield a new clause with the same form, i.e., $\{\neg\mathsf{D}_2 \sqcup \exists r.\mathsf{D}_2\}$, where $\mathsf{D}_2 \in \mathsf{N_D}$ is a fresher definer. Definers would thus be infinitely introduced, and the forgetting process would never terminate. Our method guarantees termination of forgetting by imposing the restriction mentioned above. Hence, for this example, the method does not use the combination rule to eliminate D_1, because this would bring in a new definer D_2. Since the purify rules are not applicable to this case, our method leaves the definer in the resulting ontology. This means that the forgetting does not succeed and the method is incomplete. Cyclic cases can be solved with fixpoints, but since fixpoints are not supported by the OWL API, for practicality of the method, we do not include them in the target language — we do not trade practicality for completeness.

A number of standard simplification rules are applied 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) the applicability of the purify and the combination rules can be detected quickly.

Theorem 3. *Given any $\mathcal{ALCO}$-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 introduced definer names, then our method is* successful *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.

Evaluation of the Forgetting Method

We evaluated the applicability of the method on a large corpus of real-world ontologies. To this end, we implemented a prototype of the method in Java using the OWL API. The corpus for the evaluation was based on a snapshot of the NCBO BioPortal repository taken in March 2017 (Matentzoglu and Parsia 2017), containing 396 ontologies. In 83 of these ontologies, nominals were present in their TBoxes.

	Max.	Min.	Mean	Median	90th PCTL
$\#(\mathcal{O})$	1.8M	100	4.6K	1.1K	12.6K
$\#\mathsf{sig_C}(\mathcal{O})$	847.8K	36	2.1K	502	5.6K
$\#\mathsf{sig_R}(\mathcal{O})$	1.4K	0	54	12	144
$\#\mathsf{sig_I}(\mathcal{O})$	87.9K	0	216	0	206

Table 1: Statistical information about the test ontologies

Table 1 summarizes statistical information about the test ontologies. By $\#(\mathcal{O})$, $\#\mathsf{sig_C}(\mathcal{O})$, $\#\mathsf{sig_R}(\mathcal{O})$ and $\#\mathsf{sig_I}(\mathcal{O})$, we denote respectively the average numbers of the axioms, concept names, role names and nominals in the ontologies.

The prototype was evaluated for three settings: forgetting 10%, 40% and 70% of the concept and role names from the signature of each ontology. The experimental results provide insights into the applicability of the method for various real-world applications where the expected tasks were to forget a small, a moderate or a large number of concept and role names (in line with the three settings).

We compared the results computed by our prototype with those by LETHE and FAME, respectively. As LETHE cannot handle nominals, only the $\mathcal{ALC}$-fragments of the test ontologies were used for the comparison with LETHE. For the

Tool	$\mathcal{F}\%$	Time	Timeouts	Success	Extra
PROTOTYPE $\mathcal{ALC}$	10%	1.2s	1.8%	**95.7%**	2.5%
	40%	3.3s	3.5%	**91.4%**	5.1%
	70%	6.4s	6.8%	**84.4%**	8.8%
LETHE $\mathcal{ALC}$	10%	26.1s	9.1%	79.5%	11.4%
	40%	83.3s	17.2%	63.1%	16.4%
	70%	136.9s	25.3%	53.2%	21.5%
PROTOTYPE $\mathcal{ALCO}$	10%	1.5s	2.3%	**94.7%**	3.0%
	40%	3.9s	4.3%	**89.9%**	5.8%
	70%	7.9s	7.6%	**82.7%**	9.7%
FAME $\mathcal{ALCO}$	10%	1.3s	2.0%	86.6%	11.4%
	40%	3.4s	4.0%	74.6%	21.4%
	70%	7.1s	7.1%	67.4%	25.5%

Table 2: Results computed by prototype, LETHE and FAME

comparison with FAME, the $\mathcal{ALCO}$-fragments were used, since FAME can handle $\mathcal{ALCOIH}$-ontologies.

The names to be forgotten were randomly chosen. 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. A timeout of 15 minutes was used on each run.

The results are shown in Table 2. The most striking results are: (i) the prototype was successful in more than 87% of the test cases, and (ii) in more than 90% of the successful cases, the forgetting solutions were computed within seconds. Part of the failures were due to the timeout while others due to definers being unable to be completely eliminated and therefore some remaining in the returned ontologies.

The results show our prototype was on average faster than LETHE on the $\mathcal{ALC}$-fragments. An important reason is that our method introduces definers in a *conservative* manner (it introduces definers only when really necessary), while LETHE introduces them in a *systematic* and *exhaustive* manner. This can be illustrated with the following example.

Example 4. Let $\mathcal{N} = \{C \sqcup \exists r.A, E \sqcup \forall r.\neg A\}$ and $\mathcal{F} = \{A\}$. Our method applies directly the combination rule (Case 8) to $\mathcal{N}$ to eliminate A, yielding the solution $\{C \sqcup \exists r.\top, C \sqcup E\}$. LETHE computes the same solution as our method does, but the derivation is more complicated, involving these steps:
Step 1: Normalization ($\mathsf{D}_1, \mathsf{D}_2 \in \mathsf{N_D}$):
$\{1.C \sqcup \exists r.\mathsf{D}_1, 2.\neg\mathsf{D}_1 \sqcup A, 3.E \sqcup \forall r.\mathsf{D}_2, 4.\neg\mathsf{D}_2 \sqcup \neg A\}$.
Step 2: Role propagation ($\mathsf{D}_3 \in \mathsf{N_D}$):
$\{5.C \sqcup E \sqcup \exists r.\mathsf{D}_3$ (from 1, 3), $6.\neg\mathsf{D}_3 \sqcup \mathsf{D}_1, 7.\neg\mathsf{D}_3 \sqcup \mathsf{D}_2\}$.
Step 3: Classical resolution:
$\{8.\neg\mathsf{D}_3 \sqcup A$ (2, 6), $9.\neg\mathsf{D}_3 \sqcup \neg A$ (4, 7), $10.\neg\mathsf{D}_3$ (8, 9)$\}$
Step 4: Existential role elimination: $\{11.C \sqcup E$ (5, 10)$\}$.
Step 5: At this point, $\mathcal{N}$ is saturated w.r.t. A, and no further inferences can be performed. LETHE removes all clauses that contain A; Clauses 2, 4, 8 and 9 are thus removed.
Step 6: Clause 5 is redundant because of 11. Clauses 6 and 7 are redundant because of 10. Hence, 5, 6 and 7 are removed.
Step 7: Only Clauses 1, 3, 10, 11 remain. LETHE eliminates the definers in Clauses 1, 3 and 10 by purification, yielding: $\{12.C \sqcup \exists r.\top, 13.E \sqcup \forall r.\top, 14.E \sqcup F\}$. As 13 is a tautology, the forgetting solution computed by LETHE is $\{12, 14\}$. In contract, in two inference steps, FAME computes the semantic solution $\{C \sqcup \exists r.\forall r^-.E\}$, which is stronger than the solution computed by LETHE and our prototype, but involves the inverse role r^- not expressible in $\mathcal{ALCO}$.

Compared to FAME, similar speed was observed for our prototype, but it fared considerably better w.r.t. success rates (89% as opposed to 65.3%). This is because FAME computes semantic solutions which are often expressed in a more expressive target language.[1] The column headed 'Extra' shows the percentage of the test cases with extra expressivity in the solutions, namely, our prototype used definers, and LETHE and FAME used fixpoints to capture cyclic dependencies, and LETHE used additionally disjunctive ABox assertions to represent solutions of role forgetting, FAME used additionally inverse roles, the universal role, and role conjunction to represent semantic solutions.

An executable version of the prototype, together with the test datasets, can be downloaded for review and use via http://www.cs.man.ac.uk/~schmidt/publications/aaai20/.

Conclusion and Future Work

We developed a practical method for forgetting concept and role names from $\mathcal{ALCO}$-ontologies. The method is terminating and sound, and is so far the only approach to deductive forgetting in description logics with nominals. An evaluation of a prototype implementation shows that the method achieves a significant speed-up and notably better success rates than the LETHE tool which performs deductive forgetting for $\mathcal{ALC}$-ontologies. Compared to FAME, a semantic forgetting tool for $\mathcal{ALCOIH}$-ontologies, better success rates are attained. From the perspective of ontology engineering this is very useful, as it provides ontology curators with a powerful tool to produce 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.

References

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

Bicarregui, J.; Dimitrakos, T.; Gabbay, D. M.; and Maibaum, T. S. E. 2001. Interpolation in practical formal development. *Logic Journal of the IGPL* 9(2):231–244.

Brandt, S.; Küsters, R.; Turhan, A.; and sss. 2002. Approximation and Difference in Description Logics. In *Proc. KR'02*, 203–214. Morgan Kaufmann.

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

Eiter, T.; Ianni, G.; Schindlauer, R.; Tompits, H.; and Wang, K. 2006. Forgetting in managing rules and ontologies. In *Web Intelligence*, 411–419. IEEE Computer Society.

Ghilardi, S.; Lutz, C.; and Wolter, F. 2006. Did I Damage My Ontology? A Case for Conservative Extensions in Description Logics. In *Proc. KR'06*, 187–197. AAAI Press.

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

Klein, M. C. A., and Fensel, D. 2001. Ontology versioning on the Semantic Web. In *Proc. SWWS'01*, 75–91.

Konev, B.; Walther, D.; and Wolter, F. 2008. The Logical Difference Problem for Description Logic Terminologies. In *IJCAR*, volume 5195 of *Lecture Notes in Computer Science*, 259–274. Springer.

Konev, B.; Walther, D.; and Wolter, F. 2009. Forgetting and Uniform Interpolation in Large-Scale Description Logic Terminologies. In *Proc. IJCAI'09*, 830–835. IJCAI/AAAI Press.

Koopmann, P., and Schmidt, R. A. 2013a. Forgetting Concept and Role Symbols in $\mathcal{ALCH}$-Ontologies. In *Proc. LPAR'13*, volume 8312 of *LNCS*, 552–567. Springer.

Koopmann, P., and Schmidt, R. A. 2013b. Uniform Interpolation of $\mathcal{ALC}$-Ontologies Using Fixpoints. In *Proc. FroCos'13*, volume 8152 of *Lecture Notes in Computer Science*, 87–102. Springer.

Koopmann, P., and Schmidt, R. A. 2014. Count and Forget: Uniform Interpolation of $\mathcal{SHQ}$-Ontologies. In *Proc. IJCAR'14*, volume 8562 of *Lecture Notes in Computer Science*, 434–448. Springer.

Koopmann, P., and Schmidt, R. A. 2015. Uniform Interpolation and Forgetting for $\mathcal{ALC}$-Ontologies with ABoxes. In *Proc. AAAI'15*, 175–181. AAAI Press.

Lang, J.; Liberatore, P.; and Marquis, P. 2003. Propositional independence: Formula-variable independence and forgetting. *J. Artif. Intell. Res.* 18:391–443.

Ludwig, M., and Konev, B. 2013. Towards Practical Uniform Interpolation and Forgetting for ALC Tboxes. In *Proc. DL'13*, volume 1014 of *CEUR Workshop Proceedings*, 377–389. CEUR-WS.org.

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

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

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

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

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

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

Visser, A. 1996. *Bisimulations, Model Descriptions and Propositional Quantifiers*. Logic Group Preprint Series. Utrecht University.

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

Wang, K.; Wang, Z.; Topor, R. W.; Pan, J. Z.; and Antoniou, G. 2014. Eliminating Concepts and Roles from Ontologies in Expressive Descriptive Logics. *Computational Intelligence* 30(2):205–232.

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

Zhao, Y., and Schmidt, R. A. 2018. FAME: An Automated Tool for Semantic Forgetting in Expressive Description Logics. In *Proc. IJCAR'18*, volume 10900 of *Lecture Notes in Computer Science*, 19–27. Springer.

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

Zhao, Y. 2018. *Automated Semantic Forgetting for Expressive Description Logics*. Ph.D. Dissertation, The University of Manchester, UK.