Implementation and Evaluation of Forgetting In $\mathcal{ALC}$-Ontologies

Patrick Koopmann and Renate A. Schmidt

The University of Manchester, UK
{koopmanp, schmidt}@cs.man.ac.uk

Abstract. We implement and evaluate a recently introduced method to compute uniform interpolants for ontologies specified in the description logic $\mathcal{ALC}$. The aim of uniform interpolation is to reformulate an ontology such that it only uses a specified set of symbols, while preserving consequences that involve these symbols. Uniform interpolation is useful to applications in ontology engineering and modular ontologies. It is known that uniform interpolants of ontologies in $\mathcal{ALC}$ cannot always be presented in a finite way, and that their size can in the worst case be triple exponential in the size of the original ontology. These properties leave the question on how practical computing uniform interpolants is. The aim of this paper is to approach this question by implementing our recently presented method that always computes a finite representation of the uniform interpolant – either by using fixpoint logics or by extending the signature – and by undertaking an experimental evaluation of the method on a larger set of real-life ontologies.

1 Introduction

Ontologies represent information about concepts and relations (roles) using description logics, fragments of first-order logic, to allow reasoning systems to derive implicit information automatically. The signature of an ontology is the set of symbols used by the ontology. In forgetting, the aim is to remove concept or role symbols from an ontology in such a way that all logical consequences over the remaining symbols are preserved. The result of forgetting is a uniform interpolant, the original ontology restricted to a smaller signature, such that all consequences over that signature are preserved.

Uniform interpolation and forgetting have several potential applications that are interesting in the context of ontology engineering and modular ontologies. For example, an ontology to be published contains confidential parts that should not be accessible by the public. A solution to this problem is *predicate hiding* [4], which can be performed by forgetting the confidential concepts from the ontology. A related application is *ontology obfuscation* [7]. Here again, the aim is to share an ontology for re-use by other parties without giving away all of its information. Obfuscation is a technique known in the context of software engineering which transforms a program into a functionally equivalent progam that is difficult by human users to read and understand, to prevent reverse engineering.

Often, ontologies contain terms whose main function is to give structure and make the ontology accessible. By forgetting these terms, one can create an ontology whose structure is destroyed and which is not accessible by human users, while it can still be used for deriving logical entailments over the remaining concepts.

Other applications aim at analysing ontologies or ontology changes. One such application is *exhibiting hidden relations*. Often relations between different concepts are not stated explicitly but are only deducible with the help of reasoners. To get a better understanding how certain concepts relate to each other, one can compute the uniform interpolant over a signature of interest. Uniform interpolation can also be used to compute the *logical difference* between two versions of an ontology. Extending or modifying an ontology can lead to unintended results. Checking whether consequences over a specified signature are preserved in a new version can be performed by computing its uniform interpolant and testing whether it is entailed by the original ontology.

Despite these applications, there has not been much work yet to develop practical algorithms for uniform interpolation on real-life ontologies in expressive description logics. A reason for this might be that the known theoretical properties of uniform interpolation cast doubt on whether such practical methods even exist: it is known that for ontologies expressed in $\mathcal{ALC}$, uniform interpolants are not always expressible in a finite way, if $\mathcal{ALC}$ is also used to represent the uniform interpolant. Also, in the worst case, the size of the uniform interpolant can be triple exponential in the size of the original ontology [8]. These properties already hold for general ontologies expressed in $\mathcal{EL}$ [10,9]. The method presented in [7] is a first approach towards practical uniform interpolation for $\mathcal{ALC}$-ontologies, but it only ensures termination if the uniform interpolant is approximated by a given bound.

In [6], we present a method for uniform interpolation on $\mathcal{ALC}$-ontologies that always computes finite representations of uniform interpolants with the help of fixpoint operators. The target language $\mathcal{ALC}\mu$, which is $\mathcal{ALC}$ enriched with fixpoint operators, has the same complexity properties on the common reasoning tasks as $\mathcal{ALC}$ [2], but is currently not supported by most description logic reasoners. Fixpoint operators are also not supported by OWL, the standard language for representing web ontologies. The method presented in [6] gives a solution to this by simulating fixpoints using 'helper concept symbols' in the forgetting result. This way, the uniform interpolant is approximated signature-wise using a finite representation, and still preserves all consequences over the desired signature. If helper concept symbols are used in the result, the approximated interpolant is not entailed by the original ontology anymore, which limits the application of our method for computing the logical difference between ontologies. Our experimental results suggest however that this only happens for specific combinations of ontologies and signatures. For the other mentioned applications, these helper-concepts do not pose a major problem.

In order to approach the question as to whether the method is also practical for the mentioned applications, we present an experimental evaluation of the

method on real life ontologies. The results suggests that, while for some ontologies uniform interpolants are still hard to compute, there are a lot real-world cases for which the method can be used.

2 Preliminaries

Let N_c, N_r be two disjoint sets of *concept symbols* and *role symbols*. *Concepts* in $\mathcal{ALC}$ are of the following form:

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

where $A \in N_c$, $r \in N_r$ and C and D are arbitrary concepts. $\top$, $C \sqcap D$ and $\forall r.C$ are defined as abbreviations: $\top$ stands for $\neg\bot$, $C \sqcap D$ for $\neg(\neg C \sqcup \neg D)$ and $\forall r.C$ for $\neg\exists r.\neg C$.

A TBox is a set of *axioms* of the forms $C \sqsubseteq D$ and $C \equiv D$, where C and D are concepts. $C \equiv D$ is a short-hand for the two axioms $C \sqsubseteq D$ and $D \sqsubseteq C$. Since we are only dealing with the TBox part of an ontology, we will use the terms 'ontology' and 'TBox' interchangeably.

We write $C[A]$ to denote a concept that contains a concept symbol A, and denote the result of replacing A by a different expression E by $C[E]$. For a TBox $\mathcal{T}$, $\mathcal{T}^{[A \mapsto C]}$ denotes the result of replacing every A in $\mathcal{T}$ by C.

The semantics of $\mathcal{ALC}$ is defined as follows. An *interpretation* is a pair $\mathcal{I} = \langle \Delta^{\mathcal{I}}, \cdot^{\mathcal{I}} \rangle$, where the *domain* $\Delta^{\mathcal{I}}$ is a nonempty set and the *interpretation function* $\cdot^{\mathcal{I}}$ assigns to each concept symbol $A \in N_c$ a subset of $\Delta^{\mathcal{I}}$ and to each role symbol $r \in N_r$ a subset of $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$. The interpretation function is extended to concepts as follows:

$$\bot^{\mathcal{I}} := \emptyset \qquad (\neg C)^{\mathcal{I}} := \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}} \qquad (C \sqcup D)^{\mathcal{I}} := C^{\mathcal{I}} \cup D^{\mathcal{I}}$$
$$(\exists r.C)^{\mathcal{I}} := \{x \in \Delta^{\mathcal{I}} \mid \exists y : (x, y) \in r^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}.$$

$C \sqsubseteq D$ is *true* in an interpretation $\mathcal{I}$ iff $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$. $\mathcal{I}$ is model of a TBox $\mathcal{T}$ if all axioms in $\mathcal{T}$ are true in $\mathcal{I}$. A TBox $\mathcal{T}$ is *satisfiable* if there exists a model for $\mathcal{T}$, otherwise it is *unsatisfiable*. $\mathcal{T} \models C \sqsubseteq D$ holds iff in every model of $\mathcal{T}$ we have $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$.

In order to define $\mathcal{ALC}\mu$, we extend the language with a set N_v of *concept variables*. $\mathcal{ALC}\mu$ extends $\mathcal{ALC}$ with concepts of the form $\mu X.C$ and $\nu X.C$, where $X \in N_v$, and C is a concept in which X occurs as a concept symbol only positively (under an even number of negations). $\mu X.C$ is the *least fixpoint* of C on X, $\nu X.C$ the *greatest fixpoint*.

A concept variable X is *bound* if it occurs in the scope C of a fixpoint expression $\mu X.C$ or $\nu X.C$. Otherwise it is *free*. A concept is *closed* if it does not contain any free variables. Axioms in $\mathcal{ALC}\mu$ are of the form $C \sqsubseteq D$ and $C \equiv D$, where C and D are closed concepts.

Following [3], we define the semantics of fixpoint expressions. Let $\mathcal{V}$ be an *assignment function* that maps concept variables to subsets of $\Delta^{\mathcal{I}}$. $\mathcal{V}[X \mapsto W]$ denotes $\mathcal{V}$ modified by setting $\mathcal{V}(X) = W$. $C^{\mathcal{I},\mathcal{V}}$ is the interpretation of C

taking into account this assignment, and when $\mathcal{V}$ is defined for all variables in C, $C^{\mathcal{I},\mathcal{V}} = C^{\mathcal{I}}$. The semantics of fixpoint concepts is defined as follows:

$$(\mu X.C)^{\mathcal{I},\mathcal{V}} := \bigcap\{W \subseteq \Delta^{\mathcal{I}} \mid C^{\mathcal{I},\mathcal{V}[X\mapsto W]} \subseteq W\}$$
$$(\nu X.C)^{\mathcal{I},\mathcal{V}} := \bigcup\{W \subseteq \Delta^{\mathcal{I}} \mid W \subseteq C^{\mathcal{I},\mathcal{V}[X\mapsto W]}\}.$$

A *signature* Σ is a subset of $N_s \cup N_r$. $sig(E)$ denotes the concept and role symbols occurring in E, where E ranges over concept descriptions, axioms and TBoxes. Given two TBoxes $\mathcal{T}_1$, $\mathcal{T}_2$ and a signature Σ, we say $\mathcal{T}_1$ and $\mathcal{T}_2$ are *Σ-inseparable*, in symbols $\mathcal{T}_1 \equiv_\Sigma \mathcal{T}_2$, iff for every concept inclusion α with $sig(\alpha) \subseteq \Sigma$, $\mathcal{T}_1 \models \alpha$ implies $\mathcal{T}_2 \models \alpha$ and vice versa. Given a TBox $\mathcal{T}$ and a signature Σ, $\mathcal{T}'$ is a *uniform interpolant of* $\mathcal{T}$ if $sig(\mathcal{T}') \subseteq \Sigma$ and $\mathcal{T} \equiv_\Sigma \mathcal{T}'$(Note that in contrast to conservative extensions and deductive modules no syntactical constraints are given[12]). From this definition follows that uniform interpolants for a given TBox and signature are unique modulo logical equivalence. For a given TBox and signature, we will therefore speak of *the* uniform interpolant and denote it by $\mathcal{T}^\Sigma$. Given a TBox $\mathcal{T}$ and a concept symbol A, the result of *forgetting* A *in* $\mathcal{T}$, denoted by $\mathcal{T}^{-A}$, is the uniform interpolant $\mathcal{T}^\Sigma$, where $\Sigma = sig(\mathcal{T})\backslash\{A\}$. Since $\mathcal{T}^{-A}$ entails exactly the same consequences as $\mathcal{T}$ that are not using A, it is easy to verify that $(\mathcal{T}^{-A})^{-B} \equiv (\mathcal{T}^{-B})^{-A}$. In other words forgetting a set of concept symbols one after the other always yields an equivalent TBox, regardless of the order in which symbols are processed.

3 The Method

In the following we give a brief overview of our method for computing uniform interpolants. For a more detailed description see [6]. We reduce computing of uniform interpolants to the problem of forgetting single concept symbols. In order to compute the uniform interpolant for a generic signature Σ, we forget the symbols which are not in Σ one after the other.

Given a TBox $\mathcal{T}$, the clausal form of $\mathcal{T}$, denoted by $clauses(\mathcal{T})$, is a TBox $\mathcal{T}'$ with $\mathcal{T} \equiv_{sig(\mathcal{T})} \mathcal{T}'$, such that every axiom is of the form $\top \sqsubseteq L_0 \sqcup ... \sqcup L_n$, where every L_i is of the form A,$\neg A$, $\exists r.D$ or $\forall r.D$, with $A \in N_c$, $r \in N_r$ and $D \in N_D$. $N_D \subseteq N_c \setminus sig(\mathcal{T})$ is a set of designated concept symbols called *definer symbols*. Any TBox can be transformed into its clausal form using standard structural transformation and conjunctive normal form transformation techniques. We will refer to axioms of a clausal form TBox as *clauses* and just write $L_0 \sqcup ... \sqcup L_n$ omitting the leading $\top \sqsubseteq$. We also assume that clauses are represented as sets (that is, no disjunct occurs twice in a clause and the order of the disjuncts does not matter).

Our method to compute $\mathcal{T}^{-A}$ consists of five phases:

1. Set N=$clauses(\mathcal{T})$.
2. Saturate N using the rules in Figure 1.
3. Filter out unnecessary clauses and group clauses of the form $\neg D \sqcup C_i$, where $D \in N_D$, into concept inclusions $D \sqsubseteq \bigsqcap C_i$.

Resolution:

$$\frac{C_1 \sqcup B \qquad C_2 \sqcup \neg B}{C_1 \sqcup C_2}$$

where B is the symbol A we want to forget or a definer symbol and provided $C_1 \sqcup C_2$ does not contain more than one negative definer literal.

Role Propagation:

$$\frac{C_1 \sqcup \forall r.D_1 \qquad C_2 \sqcup \mathsf{Q}r.D_2}{C_1 \sqcup C_2 \sqcup \mathsf{Q}r.D_3}$$

where $\mathsf{Q} \in \{\exists, \forall\}$ and D_3 is a (possibly new) definer symbol representing $D_1 \sqcap D_2$ and provided $C_1 \sqcup C_2$ does not contain more than one negative definer literal.

Fig. 1. Rules for forgetting concept symbol A

Non-cyclic definer elimination:

$$\frac{\mathcal{T} \cup \{D \sqsubseteq C\}}{\mathcal{T}^{[D \mapsto C]}} \quad \text{provided } D \notin \mathit{sig}(C)$$

Definer purification:

$$\frac{\mathcal{T}}{\mathcal{T}^{[D \mapsto \top]}} \quad \text{provided } D \text{ occurs only positively in } \mathcal{T}$$

Cyclic definer elimination:

$$\frac{\mathcal{T} \cup \{D \sqsubseteq C[D]\}}{\mathcal{T}^{[D \mapsto \nu X.C[X]]}} \quad \text{provided } D \in \mathit{sig}(C[D])$$

Fig. 2. Rules for eliminating definer concept symbols

4. Apply the rules in Figure 2 exhaustively to eliminate introduced symbols.
5. Apply simplifications and represent clauses as proper concept inclusions.

The rules in Figure 1 derive all consequences based on the selected concept symbol A we want to eliminate, rendering clauses containing A superfluous for the uniform interpolant. The role propagation rule is special since it may involve the introduction of new definer symbols. Because we want to preserve the clausal form in Phase 2, in order to represent a concept conjunction $D_1 \sqcap D_2$, we introduce a new definer symbol D_3 and add two clauses $\neg D_3 \sqcup D_1$ and $\neg D_3 \sqcup D_2$ to the current clause set. In order to restrict the introduction of new definer symbols, we keep track of each introduced definer symbol and reuse them as much as possible. By doing this wisely it is possible to restrict the number of introduced definer symbols to maximally $2^{|N_D|}$.

It can be shown that if a set of clauses is saturated using the rules in Figure 1, all clauses containing the selected concept symbol A and all clauses containing

positive definer symbols that do not occur under a role restriction can be removed, and the resulting set is still Σ-inseparable with the original TBox [6]. The new definer symbols that are introduced in Phase 1 and 2 are eliminated in Phase 4 using the rules in Figure 2. These rules are motivated by Ackermann's Lemma and its generalised form, first published in [1] and [11], respectively.

If the desired target language is $\mathcal{ALC}$, the cyclic definer elimination rule cannot be applied, since it introduces fixpoint operators. In this case the cyclic definers remain in the result, which means the resulting TBox is not a uniform interpolant. It does, however, not contain A and preserves all consequences not containing A. The remaining cyclic definers can be seen as 'helper concept symbols' that help keep the result finite without using fixpoint operators. The result of applying only non-cyclic definer elimination and definer purification can be viewed as *signature-wise approximation* of the uniform interpolant. It should be noted though that the existence of cyclic definers in the returned result does not necessarily imply that there is no finite representation of the uniform interpolant in $\mathcal{ALC}$.

In [6] is it proven that our method always terminates and computes the uniform interpolant in $\mathcal{ALC}\mu$, or a signature-wise approximation.

4 Implementation

We implemented our forgetting method in Scala[1] using the OWL API.[2] Since fixpoint operators are not supported by most standards and reasoners, for practical applications it is of interest to compute only results that are expressible in $\mathcal{ALC}$. For this reason, our method does not eliminate definer symbols where this would lead to a fixpoint operator in the result. In order to make the method practical, we implemented several optimisations.

Restricting the Role Propagation Rule. Though in its presented form the calculus works correctly, in order to make the method practical, it is neccessary to apply further restrictions on the role propagation rule. The main role of the role propagation rule is to derive new clauses between which resolution on the symbol we want to forget is applicable. In order to avoid the unnecessary introduction of new clauses and definer symbols, we check beforehand whether applying role propagation contributes to any further resolution rule applications. If not, we omit its application.

Redundancy Elimination. From the proofs in [6] one can see that standard redundancy elimination techniques like tautology and subsumption deletion are compatible with our method. We also take into account subsumptions between introduced definer symbols: Note that $\neg D_1 \sqcup D_2$ implies $D_1 \sqsubseteq D_2$. With every newly introduced definer symbol we build up a subsumption hierarchy for definer symbols, which enables us to check for subsumption between literals of the forms $\exists r.D_1$ and $\forall r.D_2$. On the basis of this extended subsumption notion, we implement eager subsumption deletion and condensation as in classical

[1] `http://www.scala-lang.org`
[2] `http://owlapi.sourceforge.net`

resolution-based theorem provers. The correctness of these simplifications can be proven by adaptions of the proofs for the original method in [6].

Structural Transformation. Since the resolution rule and the role propagation rule only apply to a restricted subset of literals in the clause set, the number of clauses can be significantly reduced by using further structural transformations. For a clause C, let C^A denote the literals on which our rules apply, and $C^{\overline{A}}$ the remaining literals. We replace each set of clauses $\{C_0, ..., C_n\}$, such that $C_i^A = C_j^A$ for all $i, j < n$, by a single clause $X \sqcup C_0^A$, where X is a new concept symbol, and store the information that $X \equiv C_0^{\overline{A}} \sqcap ... \sqcap C_n^{\overline{A}}$. As soon as a clause is added to the result set, we undo this transformation and apply eager subsumption deletion on the current result set. This optimisation is influenced by the uniform interpolation method presented in [7].

Simplifications. The simplifications performed in Phase 5 are the following. Following an arbitrary ordering defined on concept symbols, we select the maximal literal of the form $\neg A$, if existent, and transform the clause into an axiom of the form $A \sqsubseteq C$. We then group all concept inclusion axioms that have the same concept A on the right hand side into a single concept inclusion. We apply several replacement rules to remove tautological or unsatisfiable subexpressions. We also detect tautological fixpoint-expressions. For any fixpoint expression $\nu X.C[X]$, if $C[\top]$ is a tautology, $\top$ is the greatest fixpoint of $C[X]$, and we can replace $\nu X.C[X]$ by $\top$. Tautological and unsatisfiable sub-expressions are detected using sound but incomplete syntactic criteria. Since the number of introduced definer symbols can be exponential in the number of role restrictions of the input ontology, it is also wise to minimise their occurrences. This is accomplished in Phase 5 by transforming disjunctions of the form $\exists r.C_0 \sqcup ... \sqcup \exists r.C_n$ into single existential role restrictions $\exists r.(C_0 \sqcup ... \sqcup C_n)$ and conjunctions of the form $\forall r.C_0 \sqcap ... \sqcap \forall r.C_n$ into single universal role restrictions $\forall r.(C_0 \sqcap ... \sqcap C_n)$.

Module extraction. To restrict the number of symbols we have to forget, we first extract the syntactic locality based $\top\bot*$-module [12] for the selected signature. This module is a subset of the original ontology that preserves all consequences over the signature, but may still contain thousands of additional symbols.

Purification. Before applying our method, we compute the negation normal form $\mathcal{T}_{NNF}$ of the input ontology $\mathcal{T}$. If a concept symbol A occurs only positively in $\mathcal{T}_{NNF}$, then $\mathcal{T}^{-A} = \mathcal{T}^{[A \mapsto \top]}$. If A occurs only negatively in $\mathcal{T}_{NNF}$, then $\mathcal{T}^{-A} = \mathcal{T}^{[A \mapsto \bot]}$. We call this transformation *purification of A*. The soundness of purification follows from the fact that in these cases the resolution rule would never be applied, what effectively means we only remove clauses containing A. Purification of A leads to an equivalent result as removing clauses containing A, but can be performed much faster. When computing uniform interpolants for our experimental evaluation, we observed that in some cases already thousands of concept symbols could be eliminated using purification.

5 Experimental Evaluation

In order to evaluate how our implementation behaves on real-life ontologies, we selected a set of ontologies from the NCBO BioPortal ontology repository.[3] The ontologies of this corpus are known to be diverse in complexity, size and structure [5]. From this corpus we selected all ontologies for which it is possible to download uncorrupted files of ontologies that could be parsed using the OWL API. We further noticed that on some ontologies, extracting $\top\bot*$-modules using the OWL API caused a runtime exception. Ontologies for which this was the case were excluded from our corpus as well.

Since our method is designed for $\mathcal{ALC}$-ontologies, we restricted the ontologies to their $\mathcal{ALC}$-fragments in the following way. Axioms that can be rewritten into $\mathcal{ALC}$ axioms in a unified way (equivalent concepts, disjoint concepts, disjoint union axioms, property range axioms and property domain axioms) were rewritten, the remaining axioms that are not in $\mathcal{ALC}$ were removed from the TBox. We further removed all ontologies where the $\mathcal{ALC}$-fragment of the TBox contained less than 5 concept symbols or consisted only of axioms of the form $A \sqsubseteq B$ and $A \equiv B$, where A and B are concept symbols. This way, we extracted a corpus of 207 ontologies for our experiments.

In these ontologies, on average 5.75% of the TBox axioms had to be removed in order to generate an $\mathcal{ALC}$-TBox, while 54 ontologies were completely expressible in $\mathcal{ALC}$.

The ontologies of the resulting corpus contain between 2 and 187,514 concept symbols (on average 5,728). The average number of axioms per ontology is 21,821.20 and the average axiom size is 4.61. The size of an axiom is defined recursively as follows: $\mathit{size}(A) = 1$, where A is a concept symbol, $\mathit{size}(\neg C) = \mathit{size}(C) + 1$, $\mathit{size}(\exists r.C) = \mathit{size}(\forall r.C) = \mathit{size}(C) + 2$, $\mathit{size}(C \sqcup D) = \mathit{size}(C \sqcap D) = \mathit{size}(C) + \mathit{size}(D) + 1$, and $\mathit{size}(C \sqsubseteq D) = \mathit{size}(C \equiv D) = \mathit{size}(C) + \mathit{size}(D) + 1$.

The experiments were run on an Intel Core i5-2400 CPU with four cores running at 3.10 GHz and 8 GB of RAM. Since our implementation does not make use of multi-threading, we ran several experiments in parallel in order to make full use of the multiple processors.

Depending on the application, it might either be interesting to forget a small set of concept symbols from the ontology (predicate hiding, ontology obfuscation, logical difference), or to restrict the ontology to a small signature (exhibit hidden relations, sharing restricted parts of an ontology). We first considered how our method performed on forgetting small sets of concept symbols. For this we selected random subsets of 5, 10, 50 and 150 concept symbols, 10 subsets in each case, from the signature of each ontology, for which we applied our method. Since the average number of concept symbols per ontology is 5,728, in most cases this represented a small subset of the overall signature. If, however, the signature of an ontology contained less than the selected number of concept symbols, we omitted the corresponding experiments. This was the case for 4, 21 and 60 ontologies for the signature sizes 10, 50 and 150, respectively.

[3] `http://bioportal.bioontology.org`

Ontologies	$\lvert sig(\mathcal{T}) \setminus \Sigma \rvert$	Timeouts	Definers Left	Average Nr. of Axioms	Average Size of Axioms	Average Duration
All	5	0.3%	1.6%	18,772.61	12.81	1.1 sec.
	10	1.0%	2.2%	19,233.38	9.21	1.1 sec.
	50	1.1%	11.3%	20,577.60	23.14	6.0 sec.
	150	3.6%	17.5%	24,627.58	60.29	18.5 sec.
NCI	50	0%	0%	138,216.99	5.37	23.5 sec.
	100	2%	0%	138,170.44	6.22	117.9 sec.
	150	3%	0%	138,127.78	6.26	121.5 sec.

Table 1. Results for forgetting small sets of concept symbols.

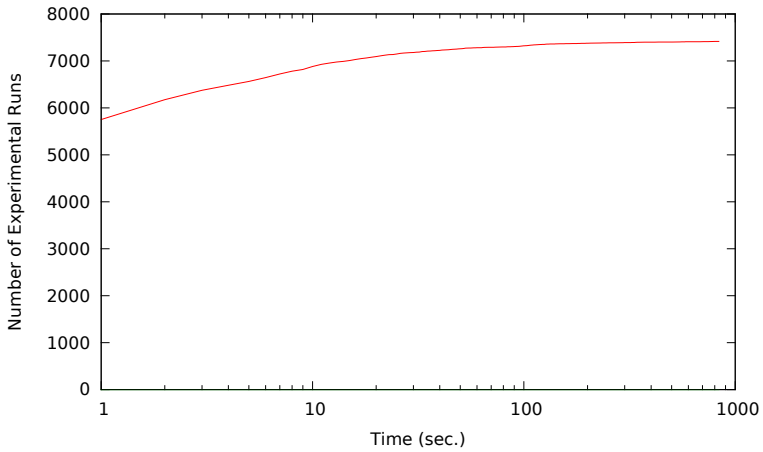

Fig. 3. Cumulative distribution of the duration of each forgetting experimental run.

We used a timeout of 1,000 seconds for each experimental run. In order to evaluate how our method performed on larger ontologies, we applied the same procedure on the $\mathcal{ALC}$-fragment of Version 13.05d of the *National Cancer Institute Thesaurus* (NCI), which was part of our corpus. The $\mathcal{ALC}$-fragment of this ontology, represented only using the operators presented in the Preliminaries Section, has 138,260 axioms of average size 5. Here, we set a higher timeout of an hour, as well as higher numbers of concept symbols, and performed 100 runs for each number.

Table 1 summarises the results of these experiments. It shows the percentage of experimental runs that could not be completed within the given time limit, the percentage of successful runs in which cyclic definer symbols remained in the results, the average number of axioms in the resulting ontology, the average size of the axioms in the result and the average duration per experimental run.

Ontologies	$\lvert\Sigma \setminus N_r\rvert$	Timeouts	Definers Left	Average Nr. of Axioms	Average Size of Axioms	Average Duration
All	5	3.5%	17.1%	3.70	627.13	5.4 sec.
	10	4.6%	20.1%	7.98	623.88	7.7 sec.
	50	8.8%	22.7%	54.84	180.48	11.8 sec.
	150	12.7%	23.1%	336.83	216.09	31.8 sec.
NCI	50	0%	15%	141.66	3,115.43	594.5 sec.
	100	4%	12%	335.28	1,876.51	927.0 sec.
	150	7%	15%	568.69	1,751.58	1,389.2 sec.

Table 2. Results for computing uniform interpolants over small signatures.

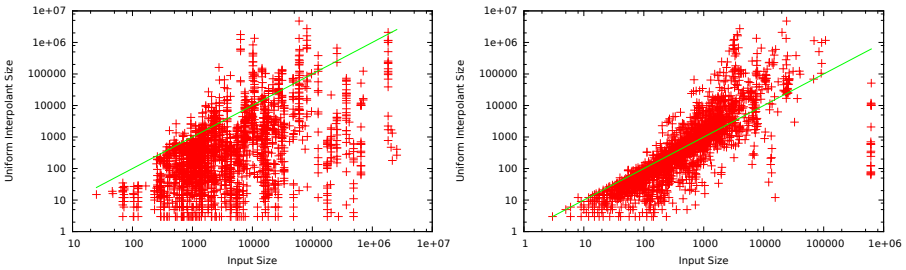

Fig. 4. Sizes of extracted modules and corresponding uniform interpolants.

The size of the ontologies remained mostly unchanged, which was due to the fact that a major part of the ontology was not touched by the method if the concept symbols were only used in a small subset.

With an increasing number of forgotten concept symbols the number of cases in which definer symbols are left in the result rose slightly, but in 93% of the cases the result could be represented finitely without definer symbols. In 99% of the cases our method was able to compute the forgetting result in the set time limit. The average duration suggests that a much smaller timeout could already have led to similar results. Figure 3 shows the cumulative distribution of the durations of each experimental run. It shows that nearly 6,000 out of 7,426 experimental runs could be performed within less than one second, which suggests that for most cases, forgetting small sets of concepts is actually a cheap operation.

Next we wanted to evaluate how good our method performed on restricting the signature of an ontology to a small set of concept symbols. Since computing uniform interpolants for small signatures is much more computationally expensive as forgetting small sets of concept symbols, we performed the experiments only on a subset of the original corpus, for which we randomly selected 170 ontologies, and performed 5 experimental runs for each ontology and sample size. The results are summarised in Table 2.

The effect of uniform interpolation was more apparent in these cases. In 20.1% of the cases, the computed uniform interpolant would have used fixpoint

operators. Even if only 5 concept symbols were used in the result, the average axiom size was 627. In case of the NCI ontology, the average size of an axiom was even higher. The main reason for this is that much more information about the role structure of the ontology and disjointnesses between concepts had to be represented in fewer axioms.

Figure 4 plots the sizes of input ontologies and the sizes of the extracted modules against the sizes of the signature-wise approximated uniform interpolants. Interestingly, in most cases the computed uniform interpolant was of similar size or smaller than the corresponding module. In 90.0% of the cases, the resulting ontology was smaller than the input ontology, and in 75.9% of the cases, it was smaller than the corresponding $\top\bot*$-module. In the most extreme case the uniform interpolant was however 559 times bigger than the corresponding $\top\bot*$-module.

The performance on our method strongly depended on how distributed the concept symbols to be forgotten are in the ontology, and how many additional symbols remained in the module. The biggest effect on computation time and output size was caused if the concept symbols to be forgotten occurred in high numbers nested under role restrictions, since the role propagation rule had to be applied more often in these cases. This lead to a high number of clauses and seemed to be the main cause for timeouts.

The corpora used for the experiments, as well as the implementation, can be found under `http://www.cs.man.ac.uk/~koopmanp/womo_experiments`.

6 Conclusion

We implemented and evaluated a recently presented method to compute uniform interpolants of $\mathcal{ALC}$-ontologies. Uniform interpolation has a lot of potential applications in ontology engineering and modular ontologies. It is known that uniform interpolants of $\mathcal{ALC}$-ontologies cannot always be represented in a finite way in $\mathcal{ALC}$, and their size is in the worst case triple exponential in the size of the input ontology. We evaluated an implementation of uniform interpolation to investigate how these theoretical properties affect uniform interpolation of $\mathcal{ALC}$-fragments of real-life ontologies. Our method computes uniform interpolants for $\mathcal{ALC}\mu$, which is $\mathcal{ALC}$ extended with fixpoint operators, to enable the finite representation of uniform interpolants in all cases. Since fixpoint operators are not supported by most standards and reasoners, our implementation uses helper concept symbols in the result, which means the computed ontologies approximate the uniform interpolant signature-wise. Our experiments showed however, that in a majority of cases this was not needed, since the uniform interpolant could be represented without fixpoint operators. Our experiments suggest that, even though the worst case complexity of the size of uniform interpolants is triple exponential, in reality, the situation where the interpolant is exponential rarely occurs. In fact, in most cases uniform interpolants could be computed in a few seconds, and were even smaller than the input ontologies. These results suggest

that, even though computing uniform interpolation for complex ontologies can be expensive, there are a lot of applications where it is practical.

In contrast to the earlier approaches on uniform interpolation of $\mathcal{ALC}$-ontologies presented [13,8], our method proceeds in a focused way in the sense that only derivations on the currently selected symbol to be forgotten are computed. This enables our method to perform efficiently on larger ontologies, but a trade-off is that our method will not always compute an interpolant in $\mathcal{ALC}$ without fixpoint operators if it exists. To illustrate the problem, consider the TBox $\mathcal{T} = \{A \sqsubseteq \exists r.A \sqcup B, B \sqsubseteq \exists r.B\}$. When forgetting B, our method computes the TBox $\mathcal{T}^{-B} = \{A \sqsubseteq \exists r.A \sqcup \nu X.\exists r.X\}$, since it only considers derivations on B. The fixpoint expression in this ontology is however redundant, since $A \sqsubseteq \exists r.A$ already entails all consequences of the form $A \sqsubseteq \exists r^n.\top$. Note that while in this example the redundancy is quite obvious, in general it will be more hidden. In future it would be desirable to find an efficient way to deal with these kind of situations.

References

1. Ackermann, W.: Untersuchungen über das Eliminationsproblem der mathematischen Logik. Mathematische Annalen 110(1), 390–413 (1935)
2. Bradfield, J., Stirling, C.: Modal mu-calculi. In: Handbook of Modal Logic, Studies in Logic and Practical Reasoning, vol. 3, pp. 721–756. Elsevier (2007)
3. Calvanese, D., Giacomo, G.D., Lenzerini, M.: Reasoning in expressive description logics with fixpoints based on automata on infinite trees. In: Proc. IJCAI '99. pp. 84–89. Morgan Kaufmann (1999)
4. Grau, B.C., Motik, B.: Reasoning over ontologies with hidden content: The import-by-query approach. J. of Artificial Intelligence Research 45, 197–255 (2012)
5. Horridge, M., Parsia, B., Sattler, U.: The state of bio-medical ontologies. Bio-Ontologies 2011 (2011)
6. Koopmann, P., Schmidt, R.A.: Uniform Interpolation of $\mathcal{ALC}$-Ontologies Using Fixpoints. In: Proc. FroCoS'13. Springer (2013), to appear.
7. Ludwig, M., Konev, B.: Towards Practical Uniform Interpolation and Forgetting for $\mathcal{ALC}$ TBoxes. http://lat.inf.tu-dresden.de/research/papers/2013/LuKo-DL-2013.pdf
8. Lutz, C., Wolter, F.: Foundations for uniform interpolation and forgetting in expressive description logics. In: Proc. IJCAI '11. pp. 989–995. AAAI Press (2011)
9. Nikitina, N.: Forgetting in General $\mathcal{EL}$ Terminologies. Proc. DL '11, CEUR-WS.org (2011)
10. Nikitina, N., Rudolph, S.: ExpExpExplosion: Uniform interpolation in general $\mathcal{EL}$ terminologies. In: Proc. ECAI'12. pp. 618–623. IOS Press (2012)
11. Nonnengart, A., Szałas, A.: A fixpoint approach to second-order quantifier elimination with applications to correspondence theory. In: Logic at Work, pp. 307–328. Springer (1999)
12. Sattler, U., Schneider, T., Zakharyaschev, M.: Which Kind of Module Should I Extract? In: Proc. DL'09. CEUR-WS.org (2009)
13. Wang, Z., Wang, K., Topor, R., Zhang, X.: Tableau-based forgetting in $\mathcal{ALC}$ ontologies. In: Proc. ECAI '10. pp. 47–52. IOS Press (2010)