EPiC Series in Computing

Volume 65, 2019, Pages 127–138

GCAI 2019. Proceedings of the 5th Global Conference on Artificial Intelligence


# DLS-Forgetter: An Implementation of the DLS Forgetting Calculus for First-Order Logic

Ruba Alassaf and Renate Schmidt

The University of Manchester, UK

**Abstract**

DLS-Forgetter is a reasoning tool that aims to compute restricted views of a knowledge base of first-order logic formulae via semantic forgetting. Semantic forgetting achieves this by eliminating predicate symbols in an equivalence preserving way up to the remaining symbols. Forgetting has many applications such as information hiding, explanation generation and computing logical difference. DLS-Forgetter combines ideas from two Ackermann-based approaches: the DLS algorithm and a modal logic inference system inspired by the algorithm. The tool enhances the DLS algorithm by incorporating an ordering over the symbols in the forgetting signature. This allows more control over the forgetting process and the application of the elimination rules. The theory behind the tool is provided by a first-order Ackermann calculus, a first-order generalisation of the rules outlined by the modal logic inference system. The purpose of the tool is to provide the research community with an experimental tool to allow further research to be conducted in the area. This paper describes the DLS-Forgetter tool along with its underlying calculus, it outlines the forgetting process used in the implementation, and presents results of an empirical evaluation.

## 1 Introduction

This paper presents a nonstandard reasoning tool called DLS-Forgetter. The aim of DLS-Forgetter is to compute forgetting solutions for problems in first-order logic based on a result of Ackermann [1]. Forgetting is a reasoning technique that seeks to reduce the symbols used to express a given set of formulae provided that equivalence is preserved up to the remaining symbols.

The recent developments in the applications of forgetting have increased interest in the topic, particularly in the artificial intelligence community. Forgetting could be applied to diverse areas such as abductive reasoning [4], logical difference [10, 11] and correspondence theory [6, 14]. The following gives a formal definition of forgetting.

**Definition 1** (**Semantic/Strong Forgetting**)**.** Let F be a first-order logic formula and $\Sigma$ a set of predicate symbols. $F'$ is a solution of semantic forgetting $\Sigma$ from $F$ iff for any interpretation $I$, $I \models F'$ iff there is some interpretation $I'$ such that $I' \models F$, and $I$ and $I'$ coincide but differ possibly on how the predicate symbols in $\Sigma$ are interpreted.

Semantic forgetting amounts to the elimination of the second-order existential quantifier[6]. Other related notions are weak forgetting which is the dual to uniform interpolation [10].

Ackermann showed that the problem of second-order quantifier elimination is incomplete and undecidable for first-order logic [1, 6]. Nonetheless, there are practical approaches to compute forgetting. These fall within two categories: saturation approaches and rewrite-replacement[1] approaches. Saturation approaches rely on the idea of exhaustively generating consequences (e.g., using a resolution algorithm) and subsequently eliminating the consequences containing symbols from the forgetting signature. Exploiting Ackermann's Lemma, which is a monotonicity property, rewrite-replacement approaches attempt to rewrite the knowledge base using a specified set of rules until a definition of a forgetting symbol is found. The elimination of the symbol is achieved via a replacement with the definition.

The current version of DLS-Forgetter combines ideas from two rewrite replacement approaches: the DLS algorithm of [5] and the Ackermann-based inference system for modal logic described in [14]. Along with one other recent implementation [16], the system is currently the only Ackermann-based system for computing forgetting for first-order logic.

## 2 The DLS Inference System

This section describes the calculus that forms the basis of the reasoning approach used in the implementation. The language used in the calculus is that of first-order logic. We use $X$ to denote a predicate symbol in the forgetting signature and $\alpha, \beta$ to denote two possibly complex formulas. To begin with, we give some preliminary definitions.

**Definition 2** (Orderings)**.** An ordering over a set $S$ is a binary relation over $S$ that is irreflexive and transitive. An ordering $>$ over a set $S$ is said to be total if and only if for any two different elements $x$ and $y$ in $S$ either $x > y$ or $y > x$.

**Definition 3.** Let $\alpha$ contain at least one occurrence of the predicate symbol $X$. We say that $X$ occurs negatively in a formula $\alpha$, if it is preceded by a negation symbol in the negation normal form of $\alpha$. Otherwise, the predicate symbol $X$ is said to occur positively in $\alpha$. Note that it is possible that $X$ occurs both positively and negatively in $\alpha$.

*Example* 1. Consider the following formula $\neg(F \vee \neg X) \wedge \neg G$. Transforming the formula into negation normal form will give us $\neg F \wedge X \wedge \neg G$. Since the predicate symbol X is not proceeded by a negation symbol in the latter, it is said that it occurs positively in the former.

**Definition 4.** Let $\alpha$ and $\beta$ be first-order formulas, $X$ be a predicate symbol, and $\bar{x}$ be a sequence of distinct variables where the length of $\bar{x}$ is equal to the arity of $X$. The expression $\beta^{X(\bar{x})}_{\alpha(\bar{x},\bar{z})}$ denotes a first-order formula that is obtained by replacing each occurrence of $X$ of the form $X(\bar{x})$ in $\beta$ by $\alpha(\bar{x}, \bar{z})$.

*Example* 2. Consider the expression $(X(a,b) \vee \neg X(b,c))^{X(x,y)}_{(R(x,y,z)\wedge\neg Q(y,z))}$. It denotes the formula obtained by replacing each occurrence of $X(x,y)$ by $(R(x,y,z) \wedge \neg Q(y,z))$. Evaluation of the expression results in $(R(a,b,z) \wedge \neg Q(b,z)) \vee \neg((R(b,c,z) \wedge \neg Q(c,z)))$.

**Definition 5.** A formula $\alpha$ is said to be positive (respectively negative) w.r.t the predicate symbol $X$ if and only if $X$ only occurs positively (respectively negatively) in $\alpha$. The formula $\alpha$ is said to be neutral w.r.t. $X$ if and only if $X$ does not occur in $\alpha$.

[1] Sometimes referred to as rewrite-substitution.

Before we introduce the inference system, we introduce Ackermann's lemma, the lemma that the forgetting approach is based upon.

**Lemma 1** (Ackermann's Lemma)**.** *Let $X$ be a predicate symbol and let $\alpha(\bar{x}, \bar{z})$ and $\beta(X)$ be classical first-order formulas, where the number of distinct variables in $\bar{x}$ is equal to the arity of $X$, and $\alpha$ contains no occurrences of $X$.*

*If $\beta(X)$ is positive w.r.t. $X$ then*

$$\exists X\{\forall \bar{x}[X(\bar{x}) \rightarrow \alpha(\bar{x}, \bar{z})] \wedge \beta(X)\} \equiv \beta^{X(\bar{x})}_{\alpha(\bar{x},\bar{z})}.$$

*If $\beta(X)$ is negative w.r.t. $X$ then*

$$\exists X\{\forall \bar{x}[\alpha(\bar{x}, \bar{z}) \rightarrow X(\bar{x})] \wedge \beta(X)\} \equiv \beta^{X(\bar{x})}_{\alpha(\bar{x},\bar{z})}.$$

Proofs of the lemma can be found in several sources, for example, in [6]. Intuitively, to forget a predicate symbol $X$, the idea is to first compute a definition for $X$, then replace every occurrence of $X$ by the definition, applying unification appropriately. In that way, we eliminate $X$ but still maintain the semantics. To illustrate the idea, we give an example.

*Example* 3. Consider:

$$\exists X\{\forall x[X(x) \rightarrow \forall y Q(x, y)] \wedge [X(a) \vee X(b)]\}$$

Applying the first form of Ackermann's lemma to eliminate the predicate symbol $X$ gives the following equivalent formula modulo $X$

$$\forall y Q(a, y) \vee \forall y Q(b, y).$$

The inference system that we discuss is based on the DLS algorithm by [5] together with an Ackermann-based inference system for modal logic described in [14]. The idea here is to generalise the novel ideas presented in [14] to the full first-order case. Given a set of first-order formulas, a forgetting signature and a total ordering over that signature, the aim is to eliminate these symbols in order to return a set of formulas or clauses that is equivalent to the original set up to the remaining symbols. To simplify the description, we assume that the formulas have been preprocessed using the steps outlined in the DLS algorithm in [6]. The only modification we advise to apply is miniscoping; that is to minimise the scope of the quantifiers as much as possible. In the end, the set of formulas are transformed into first-order clausal form.

The main difference between the inference system presented here and the DLS algorithm is that the inference system incorporates an ordering over the forgetting signature which is vital for Ackermann-based forgetting approaches. The reason behind this is that in many cases, applying Ackermann's lemma over a set of clauses may succeed for one ordering but not for another.

*Example* 4. Given a forgetting signature $\Sigma = \{X_1, X_2\}$, an ordering $>$ and $X_1 > X_2$. $F$, $G$, $H$ are possibly complex first-order formulas that share variables with the arguments of forgetting symbols. Consider:

1. $\underline{\neg X_1(x_1)} \vee F$
2. $\underline{X_1(x_4)} \vee G$
3. $\underline{X_1(x_2)} \vee \underline{\neg X_1(x_3)} \vee X_2(x_3) \vee H$

The underlining here is to put emphasis on the maximal symbol in each clause. Attempting to eliminate $X_1$ fails since $X_1$ occurs positively and negatively in 3, and the forgetting algorithm will terminate with failure. However, if the ordering is changed to $X_2 > X_1$, then eliminating $X_2$ gives

1. $\underline{\neg X_1(x_1)} \vee F$
2. $\underline{X_1(x_4)} \vee G$

$X_1$ is then no longer problematic. The forgetting solution is

$$F\{x_1/x_4\} \vee G,$$

where $\{x_1/x_4\}$ denotes the substitution mapping $x_1$ to $x_4$.

The inference rules in the calculus are given in Figure 1. The first three rules are based on Ackermann's lemma [1]. The purification rules are only applied when all occurrences of a forgetting symbol have the same polarity. They are applied independent of the ordering on the forgetting signature. This is beneficial since in Example 4, the application of this rule allows for the solution to be computed using the first ordering, because $X_2$ only occurs positively in the set and is then pure.

The Ackermann rule performs the elimination by replacing the positive occurrences of the maximal symbol in the forgetting signature with a definition that is based on the grouping of the negative occurrences of that symbol. What is important for the application of the Ackermann rule is that the arguments of all negative occurrences of the forgetting symbol are variables.

The term abstraction rule is used if a negative occurrence of the maximal forgetting symbol has non-variable arguments. The existential term abstraction rule is used to group the negative occurrences of the maximal forgetting symbol together by introducing an existential quantifier over them, in order to allow the application of the Ackermann rule. Appropriate Skolemization and clausification rules are used to eliminate the existential quantifier and the conjunction introduced by this rule.

The rules of the first-order Ackermann calculus are sound but incomplete. If the positive and negative occurrences of a forgetting predicate symbol cannot be separated and no rule in the calculus can be applied, the elimination problem cannot be solved. The following gives a counter example for a strictly maximal forgetting symbol $X$, which is not solvable.

$$N = \{\neg X(x) \vee X(y) \vee F(x, y), \neg X(x) \vee X(y) \vee G(x, y)\}$$

The output of the inference system is a *partial forgetting solution* if it contains at least one symbol from the forgetting signature.

# 3 Implementation

The implementation of DLS-Forgetter undergoes two main phases: the preprocessing phase and the forgetting phase. This section starts by describing the input and output of the system, which is followed with a high level description of what occurs in each stage.

## 3.1 Input and Output

A forgetting problem is specified by two files: the first file contains the first-order formulas and the second file contains the forgetting signature. The first-order formulas must be expressed

**Ackermann:** $$\frac{N,\ \alpha_1(\bar{x}) \vee \neg X(\bar{x}),\ \ldots\ \alpha_n(\bar{x}) \vee \neg X(\bar{x})}{N^{X(\bar{x})}_{(\alpha_1 \wedge \cdots \wedge \alpha_n)(\bar{x})}}$$

provided:

(i) $X$ is in the forgetting signature $\Sigma$,

(ii) $X$ is strictly maximal with respect to each $\alpha_i(\bar{x}) \vee \neg X(\bar{x})$,

(iii) $X$ only occurs positively in the formulas of $N$, and

(iv) For any distinct $\alpha_i$ and $\alpha_j$, no free first-order variables are shared except for possibly some or all of $\bar{x}$.

**Positive purification:** $$\frac{N}{N^{X(\bar{x})}_{\top}}$$

provided:

(i) $X$ is in the forgetting signature $\Sigma$, and

(ii) $X$ only occurs positively in the formulas of $N$.

**Negative purification:** $$\frac{N}{N^{\neg X(\bar{x})}_{\top}}$$

provided:

(i) $X$ is in the forgetting signature $\Sigma$, and

(ii) $X$ only occurs negatively in the formulas of $N$.

**Term abstraction:** $$\frac{N,\ \alpha \vee \neg X(\bar{t})}{N,\ \alpha \vee \neg X(\bar{x}) \vee \bar{x} \not\approx \bar{t}}$$

provided:

(i) $X$ is in the forgetting signature $\Sigma$,

(ii) $X$ is maximal with respect to the ordering,

(iii) $\bar{t}$ are function symbols, and

(iv) $\bar{x}$ are fresh variables.

**Existential term abstraction:** $$\frac{N,\ \alpha \vee \neg X(\bar{t}_1) \vee \cdots \vee \neg X(\bar{t}_n)}{N,\ \alpha \vee \exists \bar{x}(\neg X(\bar{x}) \wedge (\bar{x} \approx \bar{t}_1 \vee \cdots \vee \bar{x} \approx \bar{t}_n))}$$

provided:

(i) $X$ is in the forgetting signature $\Sigma$,

(ii) $X$ is maximal with respect to the ordering,

(iii) $X$ occurs negatively multiple times (i.e. $n > 1$), and

(iv) $\bar{x}$ are fresh variables.

Figure 1: The elimination rules of the first-order DLS calculus. $\bar{x} \not\approx \bar{t}$ is shorthand notation for $x_1 \not\approx t_1 \vee x_n \not\approx t_n$ if $\bar{x} = (x_1, \ldots, x_n)$ and $\bar{t} = (t_1, \ldots, t_n)$, while $\bar{x} \approx \bar{t}$ is shorthand notation for $x_1 \approx t_1 \wedge \cdots \wedge x_n \approx t_n$. Similarly for $\bar{t}$.

using the DFG syntax [9], which is also used by the SPASS prover [15]. The output of the system is a set of clauses that may contain Skolem terms. The correct way to interpret the output is to assume that all variables expressed using $x_i$, $y_i$, $z_i$ for $i \in \mathbb{N}$ are universally quantified. Skolem terms in the result are expressed in the form $f_i(\bar{x})$ for $i \in \mathbb{N}$, where $\bar{x}$ are the variables in its arity. The output of the system is a forgetting solution if none of the forgetting symbols appear in the result. If the system fails to eliminate a forgetting symbol, the symbol appears in the result; and in this case, the output is a partial forgetting solution.

## 3.2 The Preprocessing Phase

DLS-Forgetter uses a parser that was generated using ANTLR (ANother Tool for Language Recognition) [13] to translate a forgetting problem into data structure of the system. The data structure extensively uses the HashSet class from the Collection framework for the Java programming language [3] to automate simplifications that are due to redundancies in the input.[2] Suitable hashing and caching techniques are used to make this process more efficient. This has improved the performance of the current system in comparison to a previous version by up to four times for large inputs. The idea is to compute a hashcode via a well-designed hashing function for each new object. The hashcode is subsequently cached within the object for the duration of its lifetime. In this way, the system gains increased speed for relatively little memory. DLS-Forgetter executes the following preprocessing steps to transform the input into a set of clauses. First, implications, equivalences and redundant first-order quantifiers are eliminated. The formulas are then transformed into negation normal form. The scope of all first-order quantifiers are minimised until they are as minimal as possible. Skolemization is applied to eliminate the existential quantifiers. The universal quantifiers are subsequently dropped. Finally, all top-level conjunctions over the disjunctions occurring in the conjuncts are removed.

## 3.3 The Forgetting Phase

After the preprocessing phase, DLS-Forgetter starts the forgetting procedure. The rules of the DLS calculus that were presented in the previous section are applied in this phase.

The procedure starts by purifying the clauses if possible. This means if any of the symbols in the forgetting signature occur only positively or only negatively in the input, DLS-Forgetter applies the purification rules to eliminate them.

After the clauses are purified, the procedure picks a total ordering over the symbols remaining in the forgetting signature. In the default setting, the forgetting signature is ordered as specified by the user. After this, the procedure is ready to start the forgetting loop.

The forgetting loop attempts to eliminate each of the forgetting symbols remaining after the purification step in the order as given by the ordering. For each maximal symbol in the forgetting signature, the set of clauses are split based on the polarity of the forgetting symbol. If no clause contains an occurrence of the maximal symbol negatively and positively at the same time, then the elimination is possible, and the Ackermann rule is applied. Otherwise, the procedure attempts to purify the clauses and reattempts to eliminate the maximal forgetting symbol. If the elimination is still not possible, the procedure alters the ordering to make the problematic (currently maximal) forgetting symbol in hand minimal. After eliminating each

[2] We highlight this implementation detail to encourage other authors to use this class in building similar applications. Java programmers will know that the implementation of this class is highly debated within the community and many are reluctant to use it. From our experience, if coupled with good hashing functions, it could drastically improve performance.

| | | | | |
|---|---|---|---|---|
| $\neg\top \equiv \bot$ | $\alpha \wedge \bot \equiv \bot$ | $\alpha \vee \top \equiv \top$ | $\neg\alpha \wedge \alpha \wedge \beta \equiv \bot$ | $\neg\neg\alpha \equiv \alpha$ |
| $\neg\bot \equiv \top$ | $\alpha \wedge \top \equiv \alpha$ | $\alpha \vee \bot \equiv \alpha$ | $\neg\alpha \vee \alpha \vee \beta \equiv \top$ | |

Figure 2: The simplification rules used in DLS-Forgetter for a possibly complex first-order formula $\alpha$. $\beta$ also denotes a first-order formula.

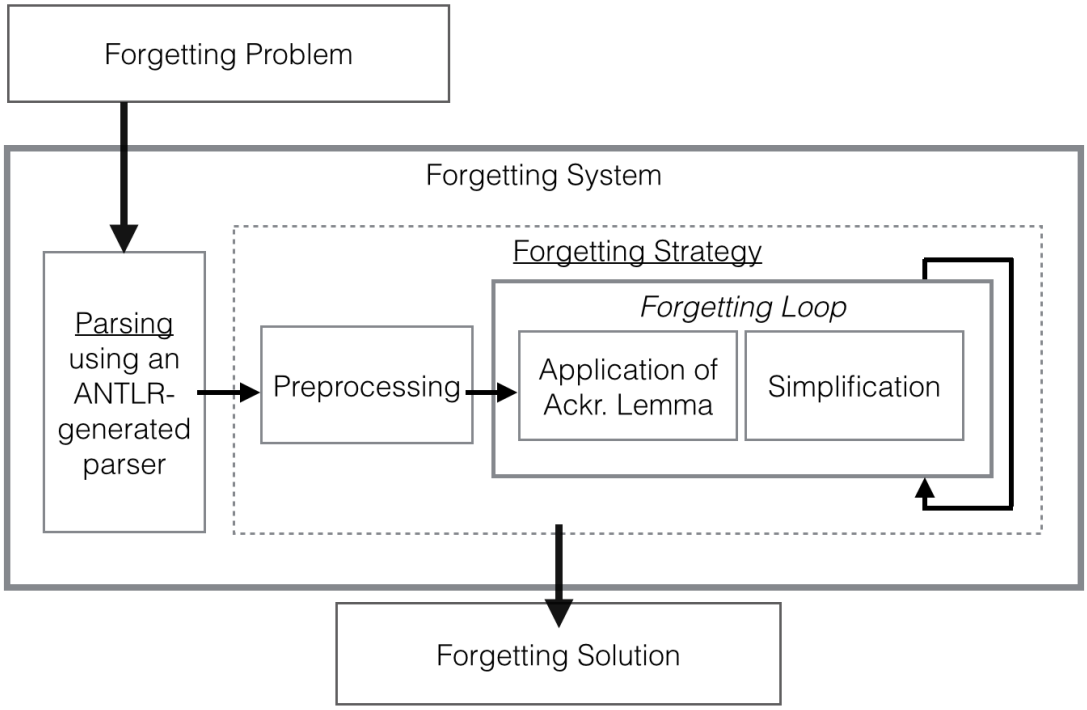


Figure 3: A top-level view of the main software components of DLS-Forgetter.

symbol, the system also checks if possible simplifications can be performed. The simplifications that the system performs can be found in Figure 2.

The procedure terminates if all the symbols in the signature are eliminated or if the system could not eliminate any of the symbols remaining in the forgetting signature.

Finally, as mentioned earlier, the output is a forgetting solution if it does not contain any of the predicate symbols specified in the forgetting signature. We say that forgetting has succeeded. Otherwise, the output is a partial forgetting solution. If the system does not succeed and outputs a partial forgetting solution, that does not imply that no solution exists. This may be because the heuristics used by the system did not compute a forgetting solution. The Ackermann approach may still be able to compute a forgetting solution if the forgetting procedure attempted to eliminate the forgetting symbols by exhaustively trying all possible orderings. Even if all eliminated order are tried there is no guarantee that a solution is found because a solution may not exist.

A summary of the internal structure of DLS-Forgetter is illustrated in Figure 3.

# 4 Related tools

The first implementation of DLS algorithm was developed and described in [8] but unfortunately, it is no longer maintained. The most closely related tool to DLS-Forgetter is a recent implementation of the DLS algorithm that is briefly described in [16] as part of the PIE environment. Another Ackermann-based forgetting tool is Fame 1.0 [18] for description logic. Recent work on Fame 1.0 has shown excellent performance and great potential for other Ackermann-based approaches [17].

The Scan algorithm [7], due to Gabbay and Ohlbach, is a resolution-based forgetting algorithm and was the first fully automated forgetting algorithm (for first-order logic). The Scan forgetting tool may be accessed via the web [12]. Even though it does not accept large inputs, it was used during the development of our tool to confirm the correctness of parts of our system. Related to the Scan algorithm is the resolution-based approach used in the Lethe forgetting tool [10] for description logics.

# 5 Evaluation

We conducted an evaluation over a corpus of large real-world description logic ontologies taken from the Oxford repository,[3] a commonly used resource that is available to evaluate reasoners. The ontologies in the Oxford repository are a collection of ontologies that were selected from other publicly available repositories such as the NCBO BioPortal repository.[4] We compare our results with results computed using the Lethe[5] forgetting tool to show the practicality of the approach and the implementation. For the purpose of the conversion from the OWL syntax, we have developed a translator to facilitate the translation of the ontologies into first-order logic (DFG syntax).

The experiments were run on a standard machine with an Intel Core i5-4200 processor, four cores running at up to 2.50 GHz. To conduct the experiments, we randomly selected 124 ontologies from the Oxford repository. The total number of experiments that were run over the ontologies is 9181 for each system. A timeout of 5 minutes was imposed on each run. To ensure that both tools receive the same ontologies, we first ran the ontologies through Lethe without forgetting any symbols.

Overall, the average amount of time it took DLS-Forgetter to compute a result was 14 seconds. The result was a partial forgetting solution in 23.6% of the tests that terminated within the allowed time period. The system terminated due to exceeding the time limit in 204 tests from 7 ontologies; 93 of which were from the ontology with the id 677. Moreover, it was observed that Lethe successfully solved 200 of these tests within the 5 minute time-out.

On the other hand, Lethe computed a forgetting solution in an average time of 32 seconds. The time limit was exceeded in 537 tests which spanned 25 ontologies. DLS-Forgetter successfully terminated in 28 of them.

Statistical information on a sample of the ontologies that we have used can be found in Table 1. The first row is the unique id given to the ontology by the developers of the Oxford repository. The row headed 'Expr.' indicates the expressivity of the test ontology. The row headed '$\#(\mathcal{O})$' indicates the number of logical axioms in the ontology. The row headed '$\#sig_c(\mathcal{O})$' indicates

[3] http://www.cs.ox.ac.uk/isg/ontologies/

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

[5] Ideally, the comparison would have been against the Scan forgetting tool since it is for first-order logic. However, this was not feasible since it does not compute forgetting for large input problems.

the number of concept symbols in the ontology. The column headed '$\#sig_r(\mathcal{O})$' indicates the number of role symbols in the ontology.

| UID | 2 | 17 | 22 | 392 | 590 | 605 | 609 | 677 |
|---|---|---|---|---|---|---|---|---|
| Expr. | $\mathcal{ALCH}$ | $\mathcal{ALCH}$ | $\mathcal{ALCH}$ | $\mathcal{ALE}$ | $\mathcal{ALCH}$ | $\mathcal{ALE}$ | $\mathcal{ALCH}$ | $\mathcal{ALE}$ |
| $\#(\mathcal{O})$ | 3368 | 1034 | 367 | 1214 | 599 | 734 | 164 | 1882 |
| $\#sig_c(\mathcal{O})$ | 761 | 97 | 39 | 1153 | 401 | 552 | 38 | 1582 |
| $\#sig_r(\mathcal{O})$ | 46 | 259 | 90 | 12 | 34 | 2 | 87 | 11 |

Table 1: Statistical information on the 8 samples of the test ontologies.

The results of the evaluation over the ontologies described in Table 1 are summarised in Table 2 and Table 3. The first column identifies the ontologies being used to conduct the experiments. The column headed '$\#\mathcal{F}$(avg)' is the average forgetting signature size for the test cases of each ontology. The column headed 'Time' gives the average time it took each system to compute an output. The results show that DLS-Forgetter is noticeably faster than Lethe on all ontologies.

| UID | $\#\mathcal{F}$(avg) | Time | Timeout | Success | Remaining $\#\mathcal{F}$(avg) |
|---|---|---|---|---|---|
| 2 | 237 | 3.2s | 0% | 41% | 2.2 |
| 17 | 48 | 1s | 0% | 0% | 9.4 |
| 22 | 20 | 0.14s | 0% | 88% | 2.2 |
| 392 | 555 | 56s | 0% | 0% | 230 |
| 590 | 200 | 1.1s | 0% | 93% | 11.6 |
| 605 | 300 | 0.74s | 0% | 100% | 0 |
| 609 | 20 | 0.06s | 0% | 11% | 2.6 |
| 677 | 1004 | 52.5s | 74% | 0% | 80.8 |

Table 2: Results of forgetting concept symbols using DLS-Forgetter.

| UID | $\#\mathcal{F}$(avg) | Time | Timeout |
|---|---|---|---|
| 2 | 237 | 30.9s | 11.2% |
| 17 | 48 | 15.7s | 61.2% |
| 22 | 20 | 5.2s | 0% |
| 392 | 555 | 69.5s | 0% |
| 590 | 200 | 9.2s | 0% |
| 605 | 300 | 9.5s | 0% |
| 609 | 20 | 0.8s | 0% |
| 677 | 1004 | 93.8s | 5.6% |

Table 3: Results of forgetting concept symbols using Lethe.

The column headed 'Timeout' shows the percentage of cases exceeding the 5 minute time limit for each tool. This evaluation shows that expect for the case of ontology 677, DLS-Forgetter did not exceed the time limit in any of the tests. In fact, if ontology 677 and 392 were excluded, the maximum time it took DLS-Forgetter to compute an output for the other ontologies was 7 seconds. The output was a partial solution with only two symbols remaining in the forgetting signature. For the same test, Lethe terminated due to exceeding the time limit.

The column headed 'Success' gives the rate in which DLS-Forgetter succeeds in computing a forgetting solution, and the column headed 'Remaining $\#\mathcal{F}$ (avg)' gives the average number of symbols that remain in the forgetting signature for partial solutions. In the cases where the success rate was high, it was observed that the reason was due to a small number of problematic symbols that appeared in the tests multiple times. For example, the maximum number of remaining symbols in the first ontology 2 and ontology 609 were 9 and 4 symbols respectively.

It was interesting to observe that in some cases DLS-Forgetter succeeded in computing a forgetting solution within seconds while Lethe was terminated due to exceeding the allowed time. On the other hand, ontology 677 shows that the contrary also occurred, as the time limit was exceeded by DLS-Forgetter in 74% of the test cases as opposed to 5.6% in the case of Lethe.

This example encouraged us to lift the time limit constraint imposed on DLS-Forgetter. We found that, for the experiments that were conducted, the maximum amount of time it took for DLS-Forgetter to terminate was 21 minutes. The output was a partial forgetting solution that contained 835 symbols from a forgetting signature of a 1500 symbol size.

These observations might indicate that for application purposes, it might be useful to investigate a hybrid approach that benefits from the speed of Ackermann-based approaches and the high success rate of resolution-based methods as was suggested in [2].

# 6 Conclusion

To summarise, this paper has introduced DLS-Forgetter; a practical nonstandard reasoning system that provides functionality for forgetting. The system is inspired by the DLS algorithm [5] with some optimisations that were adopted from [14]. Specifically, the description of this system assumes that symbols in the forgetting signature are given in a specific order. The core idea we integrated into our system was purification: if a forgetting symbol occurs only positively (or negatively), eliminating that symbol will only reduce the complexity of the problem. Therefore, symbols that occur with a single polarity should be eliminated as early as possible. However, it is important to note that in practice, this is highly dependent on the application. In certain circumstances, it may be unlikely that a symbol can be purified, and hence constantly preforming polarity checks could be expensive.

We showed through an extensive evaluation that an Ackermann-based system computes forgetting solutions in a shorter amount of time when compared against a resolution-based system. We have also seen many examples in the evaluation that show that resolution-based systems are capable of solving more forgetting problems.

In the process of comparing the results of DLS-Forgetter and the Scan forgetting tool to confirm the correctness of our system, it was observed that even for small syntactically uncomplex problems, DLS-Forgetter computed forgetting solutions for problems that Scan did not.

*Example* 5. Consider the problem of forgetting the predicate symbol $X$ from the following clauses:

1. $X(x) \vee X(y) \vee R(x, y)$
2. $\neg X(x) \vee \neg X(y) \vee T(x, y)$

Since the positive and negative occurrences of the forgetting symbol $X$ are separated, DLS-Forgetter is able to compute a result. However, the Scan algorithm does not terminate if

applied to this problem. The solution requires the use of the existential term abstraction rule which is not implemented in SCAN.

Even though a hard problem, as a future step, it would be interesting to define the class of problems each forgetting approach can solve.

The latest version of DLS-FORGETTER can be downloaded from http://personalpages.manchester.ac.uk/postgrad/ruba.alassaf/software.

# References


[1] W. Ackermann. Untersuchungen über das Eliminationsproblem der mathematischen Logik. *Mathematische Annalen*, 110:390–413, 1935.

[2] R. Alassaf and R. A. Schmidt. A preliminary comparison of the forgetting solutions computed using SCAN, LETHE and FAME. In *Proceedings of the Workshop on Second-Order Quantifier Elimination and Related Topics (SOQE 2017).*, volume 2013 of *CEUR Workshop Proceedings*, pages 21–26. CEUR-WS.org, 2017.

[3] K. Arnold, J. Gosling, and D. Holmes. *The Java Programming Language*. Addison-Wesley Longman, 2000.

[4] W. Del-Pinto and R. A. Schmidt. ABox abduction via forgetting in ALC. In P. Van Hentenryck and Z.-H. Zhou, editors, *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence (AAAI-2019)*. AAAI Press, 2019. In press.

[5] P. Doherty, W. Łukaszewicz, and A. Szałas. Computing circumscription revisited: A reduction algorithm. *Journal of Automated Reasoning*, 18(3):297–336, 1997.

[6] D. Gabbay, R. A. Schmidt, and A. Szałas. *Second-Order Quantifier Elimination: Foundations, Computational Aspects and Applications*, volume 12 of *Studies in Logic: Mathematical Logic and Foundations*. College Publications, 2008.

[7] D. M. Gabbay and H. J. Ohlbach. Quantifier elimination in second-order predicate logic. In *Proceedings of the Third International Conference on Principles of Knowledge Representation and Reasoning*, KR'92, pages 425–435. Morgan Kaufmann, 1992.

[8] J. Gustafsson. An Implementation and Optimization of an Algorithm for Reducing Formulea in Second-Order Logic. Master's thesis, Linkoping University, Sweden, 1996.

[9] R. Hähnle, M. Kerber, C. Weidenbach, and R. A. Schmidt. Common syntax of the DFG-Schwerpunktprogramm deduktion version 1.5. , Technical Report TR 10/96, Fakultät für Informatik, Universität Karlsruhe, Karlsruhe, Germany, 1996.

[10] P. Koopmann and R. A. Schmidt. LETHE: A saturation-based tool for non-classical reasoning. In M. Dumontier, B. Glimm, R. Goncalves, M. Horridge, E. Jiménez-Ruiz, N. Matentzoglu, B. Parsia, G. Stamou, and G. Stoilos, editors, *Proceedings of the 4th International Workshop on OWL Reasoner Evaluation (ORE-2015)*, volume 1387 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2015.

[11] M. Ludwig and B. Konev. Practical uniform interpolation and forgetting for ALC TBoxes with applications to logical difference. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Fourteenth International Conference, KR 2014*, 2014.

[12] H. Ohlbach, T. Engel, R. Schmidt, and D. Gabbay. Scan: Home page. http://www.mettel-prover.org/scan/index.html. [Online; accessed 2-March-2019].

[13] T. Parr. *The Definitive ANTLR 4 Reference*. Pragmatic Bookshelf, 2013.

[14] R. A. Schmidt. The Ackermann approach for modal logic, correspondence theory and second-order reduction. *Journal of Applied Logic*, 10(1):52–74, 2012.

[15] C. Weidenbach, R. A. Schmidt, T. Hillenbrand, R. Rusev, and D. Topic. System description: SPASS version 3.0. In F. Pfenning, editor, *Automated Deduction—CADE-21*, volume 4603 of *Lecture Notes in Artificial Intelligence*, pages 514–520. Springer, 2007.

[16] C. Wernhard. The PIE system for proving, interpolating and eliminating. In P. Fontaine, S. Schulz, and J. Urban, editors, *5th Workshop on Practical Aspects of Automated Reasoning, PAAR 2016*, volume 1635 of *CEUR Workshop Proceedings*, pages 125–138. CEUR-WS.org, 2016.

[17] Y. Zhao and R. Schmidt. On concept forgetting in description logics with qualified number restrictions. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*, pages 1984–1990. IJCAI/AAAI Proc., 2018.

[18] Y. Zhao and R. A. Schmidt. Fame: An automated tool for semantic forgetting in expressive description logics. In D. Galmiche, S. Schulz, and R. Sebastiani, editors, *Automated Reasoning IJCAR 2018*, pages 19–27, Cham, 2018.