Incremental Reasoning in OWL EL without Bookkeeping

Yevgeny Kazakov and Pavel Klinov

The University of Ulm, Germany
{yevgeny.kazakov, pavel.klinov}@uni-ulm.de

Abstract. We describe a method for updating the classification of ontologies expressed in the $\mathcal{EL}$ family of Description Logics after some axioms have been added or deleted. While incremental classification modulo additions is relatively straightforward, handling deletions is more problematic since it requires retracting logical consequences that are no longer valid. Known algorithms address this problem using various forms of bookkeeping to trace the consequences back to premises. But such additional data can consume memory and place an extra burden on the reasoner during application of inferences. In this paper, we present a technique, which avoids this extra cost while being very efficient for small incremental changes in ontologies. The technique is freely available as a part of the open-source $\mathcal{EL}$ reasoner ELK and its efficiency is demonstrated on naturally occurring and synthetic data.

1 Introduction and Motivation

The $\mathcal{EL}$ family of Description Logics (DLs) are tractable extensions of the DL $\mathcal{EL}$ featuring conjunction and existential restriction. Despite a limited number of constructors, $\mathcal{EL}$ became the language of choice in many applications, especially in Biology and Medicine, which require management of large terminologies. The DL $\mathcal{EL}^{++}$ [1]—an extension of $\mathcal{EL}$ with other features such as complex role inclusion axioms, nominals, and datatypes—became the basis of the OWL EL profile [2] of the Web ontology language OWL 2 specifically aimed at such applications.

Ontology classification is one of the main reasoning tasks. It requires computing all entailed (implicit) subsumption relations between atomic concepts. Specialized $\mathcal{EL}$ reasoners, such as CEL [3], ELK [4], jcel [5], and Snorocket [6] are able to compute the classification for ontologies as large as SNOMED CT [7] with about 300,000 axioms. Classification plays the key role during ontology development, e.g., for detecting modeling errors that result in mismatches between terms. But even with fast classification procedures, frequent re-classification of ontologies can introduce significant delays in the development workflow, especially as ontologies grow over time.

Several incremental reasoning procedures have been proposed to optimize frequent ontology re-classification after small changes. Most procedures maintain extra information to trace conclusions back to the axioms in order to deal with axiom deletions (see Section 2). Although managing this information typically incurs only a linear overhead, it can be a high cost for large ontologies such as SNOMED CT. In this paper, we propose an incremental reasoning method which does not require computing any of such

information. The main idea is to split the derived conclusions into several partitions. We identify partitions containing 'affected' consequences (those that could be invalidated by deletion) using a simple forward chaining procedure, and then re-compute all conclusions in these partitions. This way, we avoid storing any bookkeeping information for checking whether the affected consequences still follow from other conclusions. Our hypothesis is that, if the number of partitions is sufficiently large, changes are relatively small, and most inferences happen within individual partitions, the re-computation of affected partitions will not be too expensive. We describe a particular partitioning method for $\mathcal{EL}$ that has this property, and verify our hypothesis experimentally. Our experiments demonstrate that for large ontologies, such as SNOMED CT, incremental classification can be 10–40 times faster than the (already highly optimized) full classification, thus making re-classification almost instantaneous.

In this paper we focus on the DL $\mathcal{EL}^+$, which covers most of the existing OWL EL ontologies, and for simplicity, consider only additions and deletions of concept axioms, but not of role axioms. Although the method can be extended to changes in role axioms, it is unlikely to pay off in practice, because such changes are more likely to cause a significant impact on the result of the classification.

2 Related Work

Directly relevant to this work are various extensions to DL reasoning algorithms to support incremental changes.

Incremental classification in $\mathcal{EL}$ modulo additions implemented in the CEL system, comes closest [8]. The procedure works, essentially, by applying new inferences corresponding to the added axioms and closing the set of old and new conclusions under all inference rules. Deletion of axioms is not supported.

Known algorithms that support deletions require a form of bookkeeping to trace conclusions back to the premises. The Pellet reasoner [9] implements a technique called *tableau tracing* to keep track of the axioms used in tableau inferences [10]. Tracing maps tableau elements (nodes, labels, and relations) to the responsible axioms. Upon deletion of axioms, the corresponding elements get deleted. This method is memory-intensive for large tableaux and currently supports only ABox changes.

The *module-based* incremental reasoning method does not perform full tracing of inferences, but instead maintains a collection of modules for derived conclusions [11]. The modules consist of axioms in the ontology that entail the respective conclusion, but they are not necessarily minimal. If no axiom in the module was deleted then the entailment is still valid. Unlike tracing, the method does not require changes to the reasoning algorithm, but still incurs the cost of computing and storing the modules.

The approach presented in this paper is closely related to the classical DRed (*over-delete, re-derive*) strategy for incremental maintenance of recursive views in databases [12]. In the context of ontologies, this method was applied, e.g., for incremental updates of assertions materialized using datalog rules [13], and for *stream reasoning* in RDF [14]. Just like in DRed, we over-delete conclusions that were derived using deleted inferences (to be on the safe side), but instead of checking which deleted conclusions are

Table 1. The syntax and semantics of $\mathcal{EL}^+$

	Syntax	Semantics
Roles:		
atomic role	R	$R^{\mathcal{I}}$
Concepts:		
atomic concept	A	$A^{\mathcal{I}}$
top	$\top$	$\Delta^{\mathcal{I}}$
bottom	$\bot$	$\emptyset$
conjunction	$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$
existential restriction	$\exists R.C$	$\{x \mid \exists y \in C^{\mathcal{I}} : \langle x, y\rangle \in R^{\mathcal{I}}\}$
Axioms:		
concept inclusion	$C \sqsubseteq D$	$C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$
role inclusion	$R \sqsubseteq S$	$R^{\mathcal{I}} \subseteq S^{\mathcal{I}}$
role composition	$R_1 \circ R_2 \sqsubseteq S$	$R_1^{\mathcal{I}} \circ R_2^{\mathcal{I}} \subseteq S^{\mathcal{I}}$

still derivable using remaining inferences (which would require additional bookkeeping information), we re-compute some well-defined subset of 'broken' conclusions.

3 Preliminaries

3.1 The Description Logic $\mathcal{EL}^+$

In this paper, we will focus on the DL $\mathcal{EL}^+$ [3], which can be seen as $\mathcal{EL}^{++}$ [1] without nominals, datatypes, and the bottom concept $\bot$. It is defined w.r.t. a vocabulary consisting of countably infinite sets of *(atomic) roles* and *atomic concepts*. Complex *concepts* and *axioms* are defined recursively in Table 1. We use the letters R, S for roles, C, D, E for concepts, and A, B for atomic concepts. An *ontology* is a finite set of axioms. Given an ontology $\mathcal{O}$, we write $\sqsubseteq^*_{\mathcal{O}}$ for the smallest reflexive transitive binary relation over roles such that $R \sqsubseteq^*_{\mathcal{O}} S$ holds for all $R \sqsubseteq S \in \mathcal{O}$.

An *interpretation* $\mathcal{I}$ consists of a nonempty set $\Delta^{\mathcal{I}}$ called the *domain* of $\mathcal{I}$ and an interpretation function $\cdot^{\mathcal{I}}$ that assigns to each role R a binary relation $R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$, and to each atomic concept A a set $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$. This assignment is extended to complex concepts as shown in Table 1. $\mathcal{I}$ *satisfies* an axiom α (written $\mathcal{I} \models \alpha$) if the corresponding condition in Table 1 holds. $\mathcal{I}$ is a *model* of an ontology $\mathcal{O}$ (written $\mathcal{I} \models \mathcal{O}$) if $\mathcal{I}$ satisfies all axioms in $\mathcal{O}$. We say that $\mathcal{O}$ *entails* an axiom α (written $\mathcal{O} \models \alpha$), if every model of $\mathcal{O}$ satisfies α. The *ontology classification task* requires to compute all entailed subsumptions between atomic concepts occurring in $\mathcal{O}$.

3.2 Inferences and Inference Rules

Let **Exp** be a fixed countable set of *expressions*. An *inference* over **Exp** is an object *inf* which is assigned with a finite set of *premises inf*.Premises $\subseteq$ **Exp** and a *conclusion inf*.conclusion $\in$ **Exp**. When *inf*.Premises $= \emptyset$, we say that *inf* is an *initialization inference*. An *inference rule* R over **Exp** is a countable set of inferences over **Exp**; it is an *initialization rule* if all these inferences are initialization inferences. The *cardinality*

$$\mathbf{R_0}\ \frac{}{C \sqsubseteq C} : C \text{ occurs in } \mathcal{O} \qquad \mathbf{R}^+_\sqcap\ \frac{C \sqsubseteq D_1 \quad C \sqsubseteq D_2}{C \sqsubseteq D_1 \sqcap D_2} : D_1 \sqcap D_2 \text{ occurs in } \mathcal{O}$$

$$\mathbf{R}_\top\ \frac{}{C \sqsubseteq \top} : C \text{ and } \top \text{ occur in } \mathcal{O} \qquad \mathbf{R}_\exists\ \frac{E \sqsubseteq \exists R.C \quad C \sqsubseteq D}{E \sqsubseteq \exists S.D} : \begin{array}{l} \exists S.D \text{ occurs in } \mathcal{O} \\ R \sqsubseteq^*_\mathcal{O} S \end{array}$$

$$\mathbf{R}_\sqsubseteq\ \frac{C \sqsubseteq D}{C \sqsubseteq E} : D \sqsubseteq E \in \mathcal{O} \qquad \mathbf{R}_\circ\ \frac{E \sqsubseteq \exists R_1.C \quad C \sqsubseteq \exists R_2.D}{E \sqsubseteq \exists S.D} : \begin{array}{l} S_1 \circ S_2 \sqsubseteq S \in \mathcal{O} \\ R_1 \sqsubseteq^*_\mathcal{O} S_1 \\ R_2 \sqsubseteq^*_\mathcal{O} S_2 \end{array}$$

$$\mathbf{R}^-_\sqcap\ \frac{C \sqsubseteq D_1 \sqcap D_2}{C \sqsubseteq D_1 \quad C \sqsubseteq D_2}$$

Fig. 1. The inference rules for reasoning in $\mathcal{EL}^+$

of the rule R (notation $\|\mathsf{R}\|$) is the number of inferences *inf* $\in \mathsf{R}$. In this paper, we view an *inference system* as one inference rule R representing all of their inferences.

We say that a set of expressions *Exp* $\subseteq$ **Exp** is *closed under an inference inf* if *inf*.Premises $\subseteq$ *Exp* implies *inf*.conclusion $\in$ *Exp*. *Exp* is *closed under an inference rule* R if *Exp* is closed under every inference *inf* $\in \mathsf{R}$. *The closure under* R is the smallest set of expressions closed under R. Note that the closure is always empty if R does not contain initialization inferences.

We will often restrict inference rules to subsets of premises. Let *Exp* $\subseteq$ **Exp** be a set of expressions, and R an inference rule. By $\mathsf{R}(\mathit{Exp})$ ($\mathsf{R}[\mathit{Exp}]$) we denote the rule consisting of all inferences *inf* $\in \mathsf{R}$ such that *inf*.Premises $\subseteq$ *Exp* (respectively *Exp* $\subseteq$ *inf*.Premises). We can combine these operators: for example, $\mathsf{R}[\mathit{Exp}_1](\mathit{Exp}_2)$ consists of those inferences in R whose premises contain all expressions from Exp_1 and are a subset of Exp_2. Note that this is the same as $\mathsf{R}(\mathit{Exp}_2)[\mathit{Exp}_1]$. For simplicity, we write $\mathsf{R}()$, $\mathsf{R}[]$, $\mathsf{R}(\mathit{exp})$, and $\mathsf{R}[\mathit{exp}]$ instead of $\mathsf{R}(\emptyset)$, $\mathsf{R}[\emptyset]$, $\mathsf{R}(\{\mathit{exp}\})$, and $\mathsf{R}[\{\mathit{exp}\}]$ respectively. Note that $\mathsf{R}[] = \mathsf{R}$ and $\mathsf{R}()$ consists of all initialization inferences in R.

3.3 The Reasoning Procedure for $\mathcal{EL}^+$

The $\mathcal{EL}^+$ reasoning procedure works by applying inference rules to derive subsumptions between concepts. In this paper, we use the rules from $\mathcal{EL}^{++}$ [1] restricted to $\mathcal{EL}^+$, but present them in a way that does not require the normalization stage [4].

The rules for $\mathcal{EL}^+$ are given in Figure 1, where the premises (if any) are given above the horizontal line, and the conclusions below. Some rules have side conditions given after the colon that restrict the expressions to which the rules are applicable. For example, rule $\mathbf{R}^+_\sqcap$ contains one inference *inf* for each C, D_1, D_2, such that $D_1 \sqcap D_2$ occurs in $\mathcal{O}$ with *inf*.Premises $= \{C \sqsubseteq D_1,\ C \sqsubseteq D_2\}$, *inf*.conclusion $= C \sqsubseteq D_1 \sqcap D_2$. Note that the axioms in the ontology $\mathcal{O}$ are only used in side conditions of the rules and never used as premises of the rules.

The rules in Figure 1 are complete for deriving subsumptions between the concepts occurring in the ontology. That is, if $\mathcal{O} \models C \sqsubseteq D$ for C and D occurring in $\mathcal{O}$, then $C \sqsubseteq D$ can be derived using the rules in Figure 1 [1]. Therefore, in order to classify the ontology, it is sufficient to compute the closure under the rules and take the derived subsumptions between atomic concepts. The following example illustrates the application of rules in Figure 1 for deriving the entailed subsumption relations.

Example 1. Consider the following $\mathcal{EL}^+$ ontology $\mathcal{O}$:
(ax1): $A \sqsubseteq \exists R.B$ (ax2): $\exists H.B \sqsubseteq C$ (ax3): $R \sqsubseteq H$
(ax4): $B \sqsubseteq \exists S.A$ (ax5): $\exists S.C \sqsubseteq C$
The subsumptions below can be derived via rules in Figure 1:

$A \sqsubseteq A$	by $\mathbf{R_0}$ since A occurs in $\mathcal{O}$,	(1)
$B \sqsubseteq B$	by $\mathbf{R_0}$ since B occurs in $\mathcal{O}$,	(2)
$C \sqsubseteq C$	by $\mathbf{R_0}$ since C occurs in $\mathcal{O}$,	(3)
$\exists R.B \sqsubseteq \exists R.B$	by $\mathbf{R_0}$ since $\exists R.B$ occurs in $\mathcal{O}$,	(4)
$\exists S.A \sqsubseteq \exists S.A$	by $\mathbf{R_0}$ since $\exists S.A$ occurs in $\mathcal{O}$,	(5)
$\exists H.B \sqsubseteq \exists H.B$	by $\mathbf{R_0}$ since $\exists H.B$ occurs in $\mathcal{O}$,	(6)
$\exists S.C \sqsubseteq \exists S.C$	by $\mathbf{R_0}$ since $\exists S.C$ occurs in $\mathcal{O}$,	(7)
$A \sqsubseteq \exists R.B$	by $\mathbf{R}_\sqsubseteq$ to (1) using (ax1),	(8)
$B \sqsubseteq \exists S.A$	by $\mathbf{R}_\sqsubseteq$ to (2) using (ax4),	(9)
$\exists R.B \sqsubseteq \exists H.B$	by $\mathbf{R}_\exists$ to (4) and (2) using (ax3),	(10)
$\exists H.B \sqsubseteq C$	by $\mathbf{R}_\sqsubseteq$ to (6) using (ax2),	(11)
$\exists S.C \sqsubseteq C$	by $\mathbf{R}_\sqsubseteq$ to (7) using (ax5),	(12)
$A \sqsubseteq \exists H.B$	by $\mathbf{R}_\exists$ to (8) and (2) using (ax3),	(13)
$\exists R.B \sqsubseteq C$	by $\mathbf{R}_\sqsubseteq$ to (10) using (ax2),	(14)
$A \sqsubseteq C$	by $\mathbf{R}_\sqsubseteq$ to (13) using (ax2),	(15)
$\exists S.A \sqsubseteq \exists S.C$	by $\mathbf{R}_\exists$ to (5) and (15),	(16)
$B \sqsubseteq \exists S.C$	by $\mathbf{R}_\exists$ to (9) and (15),	(17)
$\exists S.A \sqsubseteq C$	by $\mathbf{R}_\sqsubseteq$ to (16) using (ax5),	(18)
$B \sqsubseteq C$	by $\mathbf{R}_\sqsubseteq$ to (17) using (ax5).	(19)

The subsumptions (1)–(19) are closed under these rules so, by completeness, $A \sqsubseteq A$, $B \sqsubseteq B$, $C \sqsubseteq C$, $A \sqsubseteq C$, $B \sqsubseteq C$ are all atomic subsumptions entailed by $\mathcal{O}$.

3.4 Computing the Closure under Inference Rules

Computing the closure under inference rules, such as in Figure 1, can be performed using a well-known *forward chaining procedure* presented in Algorithm 1. The algorithm derives consequences by applying inferences in R and collects those conclusions between which all inferences are applied in a set Closure and the remaining ones in a queue Todo. The algorithm first initializes Todo with conclusions of the initialization inferences $\mathsf{R}() \subseteq \mathsf{R}$ (lines 2–3), and then in a cycle (lines 4–9), repeatedly takes the next expression $exp \in$ Todo, if any, inserts it into Closure if it does not occur there, and applies all inferences $inf \in \mathsf{R}[exp](\mathsf{Closure})$ having this expression as one of the premises and other premises from Closure. The conclusions of such inferences are then inserted back into Todo.

Algorithm 1: Computing the inference closure

```
   input  : R: a set of inferences
   output : Closure: the closure under R
1  Closure, Todo ← ∅;
2  for inf ∈ R() do                                   /* initialize */
3      Todo.add(inf.conclusion);
4  while Todo ≠ ∅ do                                  /* close */
5      exp ← Todo.takeNext();
6      if exp ∉ Closure then
7          Closure.add(exp);
8          for inf ∈ R[exp](Closure) do
9              Todo.add(inf.conclusion);
10 return Closure;
```

The following example illustrates the execution of Algorithm 1 for computing the deductive closure under inferences in Figure 1.

Example 2 (Example 1 continued). The conclusions (1)–(19) in Example 1 are already listed in the order in which they would be inserted into Todo by Algorithm 1. When a conclusion is inserted into Closure, all inferences involving this and the previous conclusions are applied. For example, when (10) is inserted, the previous conclusions (1)–(9) are already in Closure, so (14) is derived and added into Todo after (11)–(13).

Note that Algorithm 1 performs as many insertions into Todo as there are inferences in $\mathsf{R}(\mathsf{Closure}')$ for the result $\mathsf{Closure}'$ because every inference $inf \in \mathsf{R}(\mathsf{Closure}')$ is eventually applied, and an inference cannot apply more than once. Therefore, the number of inferences performed by Algorithm 1 is exactly $\|\mathsf{R}(\mathsf{Closure}')\|$. The time complexity of the algorithm depends highly on the representation of the inference rules. If the initialization inferences $inf \in \mathsf{R}()$ in line 2 and matching inferences $inf \in \mathsf{R}[exp](\mathsf{Closure})$ in line 8 can be effectively enumerated, the algorithm runs in $O(\|\mathsf{R}(\mathsf{Closure}')\|)$.

4 Incremental Deductive Closure Computation

In this section, we discuss algorithms for updating the deductive closure under a set of inferences after the set of inferences has changed. Just like in Section 3.4, the material in this section is not specific to any particular inference system, i.e., does not rely on the $\mathcal{EL}^+$ classification procedure described in Section 3.3.

The problem of incremental computation of the deductive closure can be formulated as follows. Let R, R^+ and R^- be sets of inferences, and Closure the closure under R. The objective is to compute the closure under the inferences in $(\mathsf{R} \setminus \mathsf{R}^-) \cup \mathsf{R}^+$, using Closure, R, R^+, or R^-, if necessary.

Algorithm 2: Update modulo additions

input : R, R^+: sets of inferences, $\mathsf{Closure}$: the closure under R
output : $\mathsf{Closure}$: the closure under $\mathsf{R} \cup \mathsf{R}^+$

```
1  Todo ← ∅;
2  for inf ∈ (R⁺ \ R)(Closure) do                          /* initialize */
3      Todo.add(inf.conclusion);
4  R ← R ∪ R⁺;
5  while Todo ≠ ∅ do                                       /* close */
6      exp ← Todo.takeNext();
7      if exp ∉ Closure then
8          Closure.add(exp);
9          for inf ∈ R[exp](Closure) do
10             Todo.add(inf.conclusion);
11 return Closure;
```

4.1 Additions are Easy

If there are no deletions ($\mathsf{R}^- = \emptyset$), the closure under $\mathsf{R} \cup \mathsf{R}^+$ can be computed by Algorithm 2. Starting from $\mathsf{Closure}$, the closure under R, the algorithm first initializes Todo with conclusions of new inferences $\mathit{inf} \in (\mathsf{R}^+ \setminus \mathsf{R})$ applicable to $\mathsf{Closure}$, and then processes this queue with respect to the union of all inferences $\mathsf{R} \cup \mathsf{R}^+$ as it is done in Algorithm 1. Note that Algorithm 1 is just a special case of Algorithm 2 when $\mathsf{Closure} = \emptyset$ and the initial set of inferences R is empty.

Let $\mathsf{Closure}$ be the set in the input of Algorithm 2, and $\mathsf{Closure}'$ the set obtained in the output. Intuitively, the algorithm applies all inferences in $(\mathsf{R} \cup \mathsf{R}^+)(\mathsf{Closure}')$ that are not in $\mathsf{R}(\mathsf{Closure})$ because those should have been already applied. If, in contrast, we compute the closure from scratch using Algorithm 1, we would need to apply all inferences in $(\mathsf{R} \cup \mathsf{R}^+)(\mathsf{Closure}')$. Note that it is essential that Algorithm 2 starts with the closure under R. If we start with a set that is not closed under R, we may lose some conclusions because no inference in $\mathsf{R}(\mathsf{Closure})$ is applied by the algorithm.

4.2 Deletions are Difficult

Let us now see how to update the closure under deletions, i.e., when $\mathsf{R}^+ = \emptyset$. Consider Algorithm 3, which works analogously to Algorithm 2, but removes conclusions instead of adding them. In this algorithm, the queue Todo is used to buffer conclusions that should be removed from $\mathsf{Closure}$. We first initialize Todo with consequences of the removed inferences $\mathit{inf} \in \mathsf{R}^-(\mathsf{Closure})$ (lines 2–3), and then remove such elements from $\mathsf{Closure}$ together with the conclusions of inferences from $\mathsf{Closure}$ in which they participate (lines 5–10). Note that in this loop, it is sufficient to consider only consequences under the resulting $\mathsf{R} = \mathsf{R} \setminus \mathsf{R}^-$ because all consequences under R^- are already added into Todo during the initialization stage (lines 2–3).

Unfortunately, Algorithm 3 might not produce the closure under $\mathsf{R} \setminus \mathsf{R}^-$: it may delete expressions that are still derivable in $\mathsf{R} \setminus \mathsf{R}^-$. For example, for the input $\mathsf{R} =$

Algorithm 3: Update modulo deletions (incomplete)

input : R, R^-: sets of inferences, Closure: the closure under R
output : Closure: a subset of the closure under $(\mathsf{R} \setminus \mathsf{R}^-)(\mathsf{Closure})$

```
1  Todo ← ∅;
2  for inf ∈ R⁻(Closure) do                                  /* initialize */
3      Todo.add(inf.conclusion);
4  R ← R \ R⁻;
5  while Todo ≠ ∅ do                                          /* close */
6      exp ← Todo.takeNext();
7      if exp ∈ Closure then
8          for inf ∈ R[exp](Closure) do
9              Todo.add(inf.conclusion);
10         Closure.remove(exp);
11 return Closure;
```

$\{/a,\ a/b,\ b/a\}$ (x/y is an inference with the premise x and conclusion y), $\mathsf{R}^- = \{b/a\}$, and $\mathsf{Closure} = \{a,\ b\}$, Algorithm 3 removes both a since it is a conclusion of $\mathsf{R}^-(\mathsf{Closure})$, and b since it is a conclusion of $(\mathsf{R} \setminus \mathsf{R}^-)[a](\mathsf{Closure})$, yet both a and b are still derivable by the remaining inferences $\mathsf{R} \setminus \mathsf{R}^- = \{/a,\ a/b\}$.

A common solution to this problem is to check which of the removed expressions are conclusions of the remaining inferences in $\mathsf{R}(\mathsf{Closure})$, put them back into Todo, and re-apply the inferences for them like in the main loop of Algorithm 2 (lines 5–10). This is known as the DRed (over-delete, re-derive) strategy in logic programming [12]. To check whether an expression is a conclusion of some inference from Closure, however, one either needs to record how conclusions where produced, or build indexes that help to identify matching premises in Closure by conclusions. Storing this information for everything derived can consume a lot of memory and slow down the inference process.

Note that it makes little sense to 'simply re-apply' all inferences in R to the set Closure produced by Algorithm 3. This differs little from running Algorithm 1 from scratch, which applies exactly the same inferences anyway. Most of the inferences are likely to be already applied to Closure, so, even if it is not 'fully' closed under R, it may be 'almost' closed. The main idea behind our method presented in the next section, is to identify a large enough subset of expressions $\mathsf{Closure}_1 \subseteq \mathsf{Closure}$ and a large enough subset of inferences $\mathsf{R}_1 \subseteq \mathsf{R}$, such that $\mathsf{Closure}_1$ is already closed under R_1. We can then re-compute the closure under R incrementally from $\mathsf{Closure}_1$ using Algorithm 2 for $\mathsf{R}^+ = \mathsf{R} \setminus \mathsf{R}_1$. As has been shown, using this approach we can avoid applying the already applied inferences in $\mathsf{R}_1(\mathsf{Closure}_1)$.

Let Closure be the set in the input of Algorithm 3, and $\mathsf{Closure}'$ the set obtained in the output. Similarly to Algorithm 2, Algorithm 3 applies all inferences in $\mathsf{R}(\mathsf{Closure})$ except for those in $(\mathsf{R} \setminus \mathsf{R}^-)(\mathsf{Closure}')$. Indeed, during initialization (lines 2–3) the algorithm applies all inferences in $\mathsf{R}^-(\mathsf{Closure})$, and in the main loop (lines 5–10) it applies each inference in $(\mathsf{R} \setminus \mathsf{R}^-)(\mathsf{Closure})$ that is not in $(\mathsf{R} \setminus \mathsf{R}^-)(\mathsf{Closure}')$—exactly those inferences that have at least one premise in $\mathsf{Closure} \setminus \mathsf{Closure}'$. The conclusion of

every such inference is removed from $\mathsf{Closure}$, i.e., it is an element of $\mathsf{Closure} \setminus \mathsf{Closure}'$. Although, as has been pointed out, the output $\mathsf{Closure}'$ is not necessarily the closure under $\mathsf{R} \setminus \mathsf{R}^-$, it is, nevertheless, a subset of this closure:

Lemma 1. *Let* $\mathsf{Closure}$ *be the set in the input of Algorithm 3, and* $\mathsf{Closure}'$ *the set obtained in the output. Then* $\mathsf{Closure}'$ *is a subset of the closure under* $(\mathsf{R} \setminus \mathsf{R}^-)(\mathsf{Closure}')$.

Proof. Let $\mathsf{Closure}''$ be the closure under $(\mathsf{R} \setminus \mathsf{R}^-)(\mathsf{Closure}')$. We need to prove that $\mathsf{Closure}' \subseteq \mathsf{Closure}''$. Clearly, $\mathsf{Closure}' \subseteq \mathsf{Closure}$ and $\mathsf{Closure}'' \subseteq \mathsf{Closure}$. Define $\mathsf{Closure}_1 := (\mathsf{Closure} \setminus \mathsf{Closure}') \cup \mathsf{Closure}'' \subseteq \mathsf{Closure}$. We claim that $\mathsf{Closure}_1$ is closed under R. Indeed, take any $\mathit{inf} \in \mathsf{R}(\mathsf{Closure}_1)$. Then there are two cases possible:

1. $\mathit{inf} \in \mathsf{R}(\mathsf{Closure}) \setminus (\mathsf{R} \setminus \mathsf{R}^-)(\mathsf{Closure}')$: Then inf was applied in Algorithm 3. Therefore, inf.conclusion $\in \mathsf{Closure} \setminus \mathsf{Closure}' \subseteq \mathsf{Closure}_1$.
2. $\mathit{inf} \in (\mathsf{R} \setminus \mathsf{R}^-)(\mathsf{Closure}')$: Since $\mathit{inf} \in \mathsf{R}(\mathsf{Closure}_1)$ and $\mathsf{Closure}' \cap \mathsf{Closure}_1 \subseteq \mathsf{Closure}''$, we have $\mathit{inf} \in (\mathsf{R} \setminus \mathsf{R}^-)(\mathsf{Closure}'')$. Since $\mathsf{Closure}'' \subseteq \mathsf{Closure}_1$ and $\mathsf{Closure}''$ is closed under $(\mathsf{R} \setminus \mathsf{R}^-)(\mathsf{Closure}_1)$, then $\mathsf{Closure}''$ is closed under inf. Therefore inf.conclusion $\in \mathsf{Closure}'' \subseteq \mathsf{Closure}_1$.

Now, since $\mathsf{Closure}_1 \subseteq \mathsf{Closure}$ is closed under R and $\mathsf{Closure}$ is the smallest set closed under R, we have $\mathsf{Closure}_1 = \mathsf{Closure}$. Therefore, $\emptyset = \mathsf{Closure} \setminus \mathsf{Closure}_1 = \mathsf{Closure}' \setminus \mathsf{Closure}''$, and so, $\mathsf{Closure}' \subseteq \mathsf{Closure}''$, as required. □

Note that Lemma 1 claims something stronger than just that $\mathsf{Closure}'$ is a subset of the closure under $\mathsf{R} \setminus \mathsf{R}^-$. It is, in fact, a subset of the closure under $(\mathsf{R} \setminus \mathsf{R}^-)(\mathsf{Closure}') \subseteq \mathsf{R} \setminus \mathsf{R}^-$. Not every subset of the closure under $\mathsf{R} \setminus \mathsf{R}^-$ has this property. Intuitively, this property means that every expression in $\mathsf{Closure}'$ can be derived by inferences in $\mathsf{R} \setminus \mathsf{R}^-$ using only expressions in $\mathsf{Closure}'$ as intermediate conclusions. This property will be important for correctness of our method.

4.3 Incremental Updates using Partitions

Our new method for updating the closure under deletions can be described as follows. We partition the set of expressions in $\mathsf{Closure}$ on disjoint subsets and modify Algorithm 3 such that whenever an expression is removed from $\mathsf{Closure}$, its partition is marked as 'broken'. We then re-apply inferences that can produce conclusions in broken partitions to 'repair' the closure.

Formally, let $\mathbf{Pts}$ be a fixed countable set of *partition identifiers* (short *partitions*), and every expression $\mathit{exp} \in \mathbf{Exp}$ be assigned with exactly one partition exp.partition $\in \mathbf{Pts}$. For an inference rule R and a set of partitions $\mathit{Pts} \subseteq \mathbf{Pts}$, let $\mathsf{R}\langle \mathit{Pts} \rangle$ be the set of inferences $\mathit{inf} \in \mathsf{R}$ such that inf.conclusion.partition $\in \mathit{Pts}$ and exp.partition $\notin \mathit{Pts}$ for every $\mathit{exp} \in \mathit{inf}$.Premises. Intuitively, these are all inferences in R that can derive an expression whose partition is in Pts from expressions whose partitions are not in Pts.

We modify Algorithm 3 such that whenever an expression exp is removed from $\mathsf{Closure}$ in line 10, we add exp.partition into a special set of partitions Broken. This set is then used to repair $\mathsf{Closure}$ in Algorithm 4. The goal of the algorithm is to collect in the queue Todo the conclusions of inferences in $\mathsf{R}(\mathsf{Closure})$ that are missing in $\mathsf{Closure}$.

Algorithm 4: Repair of over-deletions

input : R: a set of inferences, Closure: a subset of the closure under R (Closure),
Broken: a set of partitions such that if $inf \in \mathsf{R}(\mathsf{Closure})$ and
inf.conclusion $\notin$ Closure then inf.conclusion.partition $\in$ Broken

output : Todo: the conclusions of inferences in R (Closure) that do not occur in Closure

```
1  Todo, ToRepair, Repaired ← ∅;
2  for inf ∈ R⟨Broken⟩(Closure) do                          /* initialize */
3      if inf.conclusion ∉ Closure then
4          Todo.add(inf.conclusion);
5      else
6          ToRepair.add(inf.conclusion);
7  while ToRepair ≠ ∅ do                                    /* close */
8      exp ← ToRepair.takeNext();
9      if exp ∉ Repaired then
10         for inf ∈ R[exp](Closure) do
11             if inf.conclusion.partition ∈ Broken then
12                 if inf.conclusion ∉ Closure then
13                     Todo.add(inf.conclusion);
14                 else
15                     ToRepair.add(inf.conclusion);
16         Repaired.add(exp);
17 return Todo;
```

This is done by applying all possible inferences $inf \in \mathsf{R}(\mathsf{Closure})$ that can produce such conclusions. There can be two types of such inferences: those whose premises do not belong to any partition in Broken, and those that have at least one such premise. The inferences of the first type are $\mathsf{R}\langle\mathsf{Broken}\rangle(\mathsf{Closure})$; they are applied in initialization (lines 2–6). The inferences of the second type are applied in the main loop of the algorithm (lines 7–16) to the respective expression in Closure whose partition is in Broken.

Whenever an inference *inf* is applied and *inf*.conclusion belongs to a partition in Broken (note that it is always the case for $inf \in \mathsf{R}\langle\mathsf{Broken}\rangle(\mathsf{Closure})$, see also line 11), we check if *inf*.conclusion occurs in Closure or not. If it does not occur, then we put the conclusion into the output Todo (lines 4, 13). Otherwise, we put it into a special queue ToRepair (lines 6, 15), and repeatedly apply for each $exp \in \mathsf{ToRepair}$ all inferences $inf \in \mathsf{R}[exp](\mathsf{Closure})$ of the second type in the main loop of the algorithm (lines 7–16). After applying all inferences, we move *exp* into a special set Repaired (line 16), which is there to make sure that we never consider *exp* again (see line 9).

Lemma 2. *Let* R, Closure, *and* Broken *be the inputs of Algorithm 4, and* Todo *the output. Then* $\mathsf{Todo} = \{inf.\mathsf{conclusion} \mid inf \in \mathsf{R}(\mathsf{Closure})\} \setminus \mathsf{Closure}$.

Proof. Let $\mathsf{Closure}' = \{inf.\mathsf{conclusion} \mid inf \in \mathsf{R}(\mathsf{Closure})\}$. We need to demonstrate that $\mathsf{Todo} = \mathsf{Closure}' \setminus \mathsf{Closure}$. Since $\mathsf{Todo} \subseteq \mathsf{Closure}'$ and $\mathsf{Closure} \cap \mathsf{Todo} = \emptyset$, it is sufficient to prove that $\mathsf{Closure}' \setminus \mathsf{Closure} \subseteq \mathsf{Todo}$.

First, note that $\mathsf{Closure}'$ is the closure under $\mathsf{R}(\mathsf{Closure})$. Indeed, if $\mathsf{Closure}''$ is the closure under $\mathsf{R}(\mathsf{Closure})$, then $\mathsf{Closure} \subseteq \mathsf{Closure}''$ by the assumption of Algorithm 4. Hence, for every $\mathit{inf} \in \mathsf{R}(\mathsf{Closure}) \subseteq \mathsf{R}(\mathsf{Closure}'')$, we have inf.conclusion $\in \mathsf{Closure}''$. Therefore, $\mathsf{Closure}' \subseteq \mathsf{Closure}''$, and since $\mathsf{Closure}'$ is closed under $\mathsf{R}(\mathsf{Closure})$, we have $\mathsf{Closure}' = \mathsf{Closure}''$.

Let $\mathsf{Closure}_1 = \{\mathit{exp} \in \mathsf{Closure} \mid \mathit{exp}.\mathsf{partition} \notin \mathsf{Broken}\}$. Then it is easy to see from Algorithm 4 that for every $\mathit{inf} \in \mathsf{R}(\mathsf{Closure}_1 \cup \mathsf{Repaired})$, we have inf.conclusion $\in \mathsf{Closure}_1 \cup \mathsf{Repaired} \cup \mathsf{Todo}$. Indeed, if inf.conclusion.partition $\notin \mathsf{Broken}$ then by assumption of Algorithm 4, since $\mathit{inf} \in \mathsf{R}(\mathsf{Closure})$ and inf.conclusion.partition $\notin \mathsf{Broken}$, we must have inf.conclusion $\in \mathsf{Closure}$, and thus inf.conclusion $\in \mathsf{Closure}_1$.

If inf.conclusion.partition $\in \mathsf{Broken}$, there are two cases possible. Either $\mathit{inf} \in \mathsf{R}(\mathsf{Closure}_1)$, thus, $\mathit{inf} \in \mathsf{R}\langle\mathsf{Broken}\rangle(\mathsf{Closure})$. In this case inf is applied in Algorithm 4 during initialization (lines 2–6). Or, otherwise, inf has at least one premise in $\mathsf{Repaired}$, and hence, it is applied in the main loop of Algorithm 4 (lines 7–16). In both cases the algorithm ensures that inf.conclusion $\in \mathsf{Repaired} \cup \mathsf{Todo}$.

Now, since $\mathsf{Closure}_1 \cup \mathsf{Repaired} \cup \mathsf{Todo}$ is closed under $\mathsf{R}(\mathsf{Closure}_1 \cup \mathsf{Repaired})$ and $\mathsf{Closure} \cap \mathsf{Todo} = \emptyset$, it is also closed under $\mathsf{R}(\mathsf{Closure})$ (if $\mathit{inf} \in \mathsf{R}(\mathsf{Closure})$ is applicable to $\mathsf{Closure}_1 \cup \mathsf{Repaired} \cup \mathsf{Todo}$ then $\mathit{inf} \in \mathsf{R}(\mathsf{Closure}_1 \cup \mathsf{Repaired})$). Since $\mathsf{Closure}'$ is the closure under $\mathsf{R}(\mathsf{Closure})$, we therefore, have $\mathsf{Closure}' = \mathsf{Closure}_1 \cup \mathsf{Repaired} \cup \mathsf{Todo} \subseteq \mathsf{Closure} \cup \mathsf{Todo}$. Hence, $\mathsf{Closure}' \setminus \mathsf{Closure} \subseteq \mathsf{Todo}$, as required. □

After computing the repair Todo of the set $\mathsf{Closure}$ using Algorithm 4, we can compute the rest of the closure as in Algorithm 2 using the partially initialized Todo. The correctness of the complete incremental procedure follows from Lemma 1, Lemma 2, and the correctness of our modification of Algorithm 2 when Todo is initialized with missing conclusions of $\mathsf{R}(\mathsf{Closure})$.

Algorithm 4 does not impose any restrictions on the assignment of partitions to expressions. Its performance in terms of the number of operations, however, can substantially depend on this assignment. If we assign, for example, the same partition to all expressions, then in the main loop (lines 7–16) we have to re-apply all inferences in $\mathsf{R}(\mathsf{Closure})$. Thus, it is beneficial to have many different partitions. At another extreme, if we assign a unique partition to every expression, then $\mathsf{R}\langle\mathsf{Broken}\rangle$ would consist of all inferences producing the deleted expressions, and we face the problem of identifying such inferences in lines 2–6. Next, we present a specific partition assignment for the $\mathcal{EL}^+$ rules in Figure 1, which circumvents both of these problems.

5 Incremental Reasoning in $\mathcal{EL}^+$

In this section, we apply our method for updating the classification of $\mathcal{EL}^+$ ontologies computed using the rules in Figure 1. We only consider changes in concept inclusion axioms while resorting to full classification for changes in role inclusions and compositions. We first describe our strategy of partitioning the derived subsumptions, then discuss some issues related to optimizations, and, finally, present an empirical evaluation measuring the performance of our incremental procedure on existing ontologies.

5.1 Partitioning of Derived $\mathcal{EL}^+$ Subsumptions

The inferences R in Figure 1 operate with concept subsumptions of the form $C \sqsubseteq D$. We partition them into sets of subsumptions having the same left-hand side. Formally, the set of partition identifiers $\mathbf{Pts}$ is the set of all $\mathcal{EL}^+$ concepts, and every subsumption $C \sqsubseteq D$ is assigned to the partition corresponding to its left-hand side C. This assignment provides sufficiently many different partitions, which could be as many as there are concepts in the input ontology. It also has the advantage that the inferences $\mathsf{R}\langle \mathit{Pts} \rangle$ for any set *Pts* of partitions can be easily identified. Indeed, note that every conclusion of a rule in Figure 1, except for the initialization rules $\mathbf{R_0}$ and $\mathbf{R}_\top$, has the same left-hand side as one of the premises of the rule. Therefore, $\mathsf{R}\langle \mathit{Pts} \rangle$ can only contain those initialization inferences in $\mathbf{R_0}$ and $\mathbf{R}_\top$ for which $C \in \mathit{Pts}$.

5.2 Optimizations

Let us discuss a few optimizations that are specific to the $\mathcal{EL}^+$ inference rules.

Rule optimizations: The approach described in Section 4 can be used with any $\mathcal{EL}^+$ classification procedure that implements the inference rules in Figure 1 as they are. Existing implementations, however, include several optimizations to avoid unnecessary applications of some rules. One of such optimizations in ELK prevents applying rule $\mathbf{R}^-_\sqcap$ to conclusions of $\mathbf{R}^+_\sqcap$, and rules $\mathbf{R}_\exists$ and $\mathbf{R}_\circ$ if its left premise was obtained by $\mathbf{R}_\exists$ [15]. Even though the closure computed by Algorithm 1 does not change under such optimizations (the algorithm just derives fewer duplicate conclusions), if the same optimizations are used for deletions in Algorithm 3, some subsumptions that are no longer derivable may remain in $\mathsf{Closure}$. Intuitively, this happens because the inferences for deleting conclusions in Algorithm 3 can be applied in a different order than they were applied in Algorithm 1 for deriving these conclusions. Please refer to the technical report [16] for an extended example of this situation.

To fix this problem, we do not use rule optimizations for deletions in Algorithm 3. To repair the closure using Algorithm 4, we also need to avoid optimizations to make sure that all expressions in broken partitions of $\mathsf{Closure}$ are encountered, but it is sufficient to insert only conclusions of optimized inferences into Todo.

Subsumptions that cannot be re-derived: When Algorithm 3 deletes an expression *exp* from $\mathsf{Closure}$, we mark *exp*.partition as broken because this expression could be re-derived. In some situations this is not possible. One property of the $\mathcal{EL}^+_\bot$ rules in Figure 1, is that they derive only subsumptions of the form $C \sqsubseteq D$ or $C \sqsubseteq \exists R.D$ where C and D occur in the ontology. So, if a deleted subsumption is not of this form for the ontology after deletion, we know that it cannot be re-derived. For example, consider the following ontology $\mathcal{O}$: (ax1) $A \sqsubseteq B$, (ax2) $B \sqsubseteq C$, from which (ax2) is deleted. When the previously derived conclusion $A \sqsubseteq C$ is deleted, there is no need to mark the partition of A as broken since C does not occur in the ontology after the deletion.

Structural rules: When we apply our incremental procedure for the $\mathcal{EL}^+_\bot$ in Figure 1, we take R^- to be the inferences that are no longer valid after deletion of axioms. An inference by a rule in Figure 1 is not valid when its side condition is not satisfied. For example, for the rule $\mathbf{R}_\sqsubseteq$, the subsumption $D \sqsubseteq E$ may be removed from the ontology, or for the rule $\mathbf{R}^+_\sqcap$, the conjunction $D_1 \sqcap D_2$ does not occur in the ontology any

more. But the impact of these two inferences is different: the conclusion of $\mathbf{R}_{\sqsubseteq}$ may be not correct if the side condition does not hold, but the conclusion of $\mathbf{R}^{+}_{\sqcap}$ is always correct, but may be just irrelevant. This distinction between the rules can be used in our next optimization. We call the rules $\mathbf{R_0}$, $\mathbf{R}_{\top}$, $\mathbf{R}^{+}_{\sqcap}$, $\mathbf{R}^{-}_{\sqcap}$, and $\mathbf{R}_{\exists}$ *structural*—these rules use only the structure of the concepts; they are sound even if their side conditions are not satisfied. Avoiding application of some structural rules during deletions may result in fewer broken partitions as shown in the next example.

Consider an ontology $\mathcal{O}$: (ax1) $A \sqsubseteq B$, (ax2) $A \sqsubseteq C$, (ax3) $(B \sqcap C) \sqcap D \sqsubseteq E$. The rules in Figure 1 derive the following conclusions (with the partition A):

$$A \sqsubseteq A \qquad \text{by } \mathbf{R_0} \text{ since } A \text{ occurs in } \mathcal{O}, \tag{20}$$

$$A \sqsubseteq B \qquad \text{by } \mathbf{R}_{\sqsubseteq} \text{ to (20) using (ax1)}, \tag{21}$$

$$A \sqsubseteq C \qquad \text{by } \mathbf{R}_{\sqsubseteq} \text{ to (20) using (ax2)}, \tag{22}$$

$$A \sqsubseteq B \sqcap C \qquad \text{by } \mathbf{R}^{+}_{\sqcap} \text{ to (21) and (22) using (ax3)}. \tag{23}$$

Now, assume that (ax3) is deleted from $\mathcal{O}$. Normally, we should revert the inference producing (23) by $\mathbf{R}^{+}_{\sqcap}$ using (ax3) in the deletion stage, which would then mark the partition of A as broken. We can, however, leave this rule applied (because it is still sound), which not only makes the partition of A unaffected, but also prevents further deletion of subsumptions $A \sqsubseteq B$ and $A \sqsubseteq C$ by rule $\mathbf{R}^{-}_{\sqcap}$ applied to (23).

5.3 Experimental Evaluation

We have implemented the procedure described in Section 4.3 in the OWL EL reasoner ELK v.0.4.0,[1] and performed some experiments to evaluate its performance.

We used three large OWL EL ontologies which are frequently used in evaluations of $\mathcal{EL}$ reasoners [3–6]: the Gene Ontology GO [17] with $84,955$ axioms, an $\mathcal{EL}^{+}$-restricted version of the GALEN ontology with $36,547$ axioms,[2] and the official January 2013 release of SNOMED CT with $296,529$ axioms.[3]

The recent change history of GO is readily available from the public repository.[4] We took the last (as of April 2013) 342 changes of GO (the first at r560 with $74,708$ axioms and the last at r7991 with $84,955$ axioms). Each change is represented as sets of added and deleted axioms (an axiom modification counts as one deletion plus one addition). Out of the 9 role axioms in GO, none was modified. Unfortunately, similar data was not available for GALEN or SNOMED CT. We used the approach of Cuenca Grau *et.al* [11] to generate 250 versions of each ontology with n random additions and deletions ($n = 1, 10, 100$). For each change history, we classified the first version of the ontology and then classified the remaining versions incrementally. We used a PC with Intel Core i5-2520M 2.50GHz CPU, running Java 1.6 with 4GB of RAM available to JVM.

[1] In fact, the incremental procedure in ELK supports many other features outside of $\mathcal{EL}^{+}$, such as assertions, disjointness axioms, and restricted use of nominals and datatype restrictions, see `http://elk.semanticweb.org` for the full release notes.

[2] `http://www.co-ode.org/galen/`

[3] `http://www.ihtsdo.org/snomed-ct/`

[4] `svn://ext.geneontology.org/trunk/ontology/`

Table 2. Number of inferences and running times (in ms.) for test ontologies. The results for each incremental stage are averaged (for GO the results are only averaged over changes with a non-empty set of deleted axioms). Time (resp. number of inferences) for initial classification is: GO (r560): 543 (2,224,812); GALEN: 648 (2,017,601); SNOMED CT: 10,133 (24,257,209)

Ontology	Changes	Deletion			Repair		Addition		Total	
	add.+del.	# infer.	$\lvert\mathsf{Broken}\rvert$	time	# infer.	time	# infer.	time	# infer.	time
GO (r560)	84+26	62,384	560	48	17,628	8	58,933	66	138,945	134
GALEN	1+1	3,444	36	18	4,321	4	3,055	13	10,820	39
($\mathcal{EL}^+$ version)	10+10	68,794	473	66	37,583	17	49,662	52	156,039	147
	100+100	594,420	4,508	214	314,666	96	426,462	168	1,335,548	515
SNOMED CT	1+1	4,022	64	120	423	1	2,886	68	7,331	232
(Jan 2013)	10+10	42,026	251	420	8,343	4	31,966	349	82,335	789
	100+100	564,004	3,577	662	138,633	56	414,255	545	1,116,892	1,376

The results of the initial and incremental classifications are given in Table 2. For GO we have only included results for changes that involve deletions (otherwise the averages for deletion and repair would be artificially lower). First note that in each case, the incremental procedure makes substantially fewer inferences and takes less time than the initial classification. Unsurprisingly, the difference is most pronounced for larger ontologies and smaller values of n. Also note that the number of inferences in each stage and the number of partitions $\lvert\mathsf{Broken}\rvert$ affected by deletions, depend almost linearly on n, but not the running times. This is because applying several inferences at once is more efficient than separately. Finally, observe that the repair stage takes a relatively small fraction of the total time.

In order to compare our method to the module-based approach of [11] (the only implemented incremental reasoning procedure for DLs which works for TBox additions and deletions that we are aware of) we classified the same history of GO changes using the implementation included in the standard distribution of Pellet 2.3.2.[5] Pellet provides a consequence-based procedure for $\mathcal{EL}$ classification which was used for re-classifying the affected parts of the ontology. Unfortunately the same experiment was not possible for the other two ontologies due to time-outs (10 hours). The results for GO are as follows: initial classification together with module extraction takes 126 seconds, the average incremental classification 101 seconds, the average numbers of re-computed modules are 634 (when processing deletions) and 672 (for additions).

Abstracting from the much worse time results,[6] which are likely due to a naive implementation of the module-based incremental procedure and/or the $\mathcal{EL}$ algorithm in Pellet, it is interesting to compare the average number of modules which are re-computed during the deletion stage with the average number of broken partitions reported by our algorithm. Intuitively, both of these metrics characterise the number of named concepts for which subsumers need to be re-computed upon an axiom change. The number of modules (634) is greater than the number of broken partitions (560). Interestingly, this relationship is of general nature. We prove in the technical report [16]

[5] `http://clarkparsia.com/pellet/`

[6] Note that the times in Table 2 are in milliseconds, not in seconds

that if a subsumption $C \sqsubseteq D$ is deleted by our (optimized) incremental algorithm as a result of deleting some axiom α, then α is contained in the locality-based module for C and thus the module must be re-computed. Simply put, the generic module-based approach may not incur less overhead than our method for $\mathcal{EL}^+$.

In general, this relationship does not hold in the other direction since modules can contain more axioms than used in derivations. For example, consider the ontology $\mathcal{O}$ containing $A \sqsubseteq \exists R.B$ and $B \sqsubseteq C$. The rules in Figure 1 derive only $A \sqsubseteq A$ and $A \sqsubseteq \exists R.B$ in the partition for A, thus removing $B \sqsubseteq C$ will not break the partition for A. On the other hand, the locality-based module for A contains all axioms in $\mathcal{O}$, and thus, it has to be re-computed after the deletion. The difference between the number of re-extracted modules and the number of broken partitions is likely to be greater for more complex ontologies, e.g., GALEN. The structure of the anatomical part of GALEN is known to induce very large locality-based modules [11].

Finally, we have evaluated the effectiveness of the two optimizations from Section 5.2 that can reduce the set of broken partitions when some concepts get deleted from the ontology. Avoiding applications of structural rules during deletion gives the most improvement. It reduces the set Broken by roughly 10%, e.g., 498 vs 560 on average for GO. This leads to reduction of the total number of rule applications also by 10%. The time difference is most visible for smaller change sizes, e.g. ± 1 and ± 10 for GALEN and SNOMED CT. Please see the technical report [16] for detailed results.

6 Summary and Future Research

In this paper we have presented a new method for incremental classification of $\mathcal{EL}^+$ ontologies. It is simple, supports both additions and deletions, and does not require deep modification of the base reasoning procedure. Our experiments, though being preliminary due to the shortage of revision histories for real-life $\mathcal{EL}$ ontologies, demonstrate that the reasoning results can be obtained almost instantly after small changes. Potential applications of the method range from background classification of ontologies in editors to stream reasoning and query answering. The method could also be used to handle ABox changes (via a TBox encoding) or easily extended to consequence-based reasoning procedures for more expressive Description Logics [18, 19].

The main idea of our method is that we can benefit from knowing the exact rules of $\mathcal{EL}^+$, which is not possible in the general DRed setting. In particular, we can exploit the 'granularity' of the $\mathcal{EL}^+$ procedure, namely that subsumers of different concepts can be often computed independently of each other. A similar property is a corner stone for the concurrent $\mathcal{EL}$ classification algorithm used in ELK where contexts are similar to our partitions [4]. In the future, we intend to further exploit this property for on-demand proof generation (for explanation and debugging) and distributed $\mathcal{EL}$ reasoning.

References

1. Baader, F., Brandt, S., Lutz, C.: Pushing the $\mathcal{EL}$ envelope. In Kaelbling, L., Saffiotti, A., eds.: Proc. 19th Int. Joint Conf. on Artificial Intelligence (IJCAI'05), Professional Book Center (2005) 364–369

2. Motik, B., Cuenca Grau, B., Horrocks, I., Wu, Z., Fokoue, A., Lutz, C., eds.: OWL 2 Web Ontology Language: Profiles. W3C Recommendation (27 October 2009) Available at http://www.w3.org/TR/owl2-profiles/.
3. Baader, F., Lutz, C., Suntisrivaraporn, B.: Efficient reasoning in $\mathcal{EL}^+$. In Parsia, B., Sattler, U., Toman, D., eds.: Proc. 19th Int. Workshop on Description Logics (DL'06). Volume 189 of CEUR Workshop Proceedings., CEUR-WS.org (2006)
4. Kazakov, Y., Krötzsch, M., Simančík, F.: Concurrent classification of $\mathcal{EL}$ ontologies. In Aroyo, L., Welty, C., Alani, H., Taylor, J., Bernstein, A., Kagal, L., Noy, N., Blomqvist, E., eds.: Proc. 10th Int. Semantic Web Conf. (ISWC'11). Volume 7032 of LNCS., Springer (2011) 305–320
5. Mendez, J., Ecke, A., Turhan, A.Y.: Implementing completion-based inferences for the $\mathcal{EL}$-family. In Rosati, R., Rudolph, S., Zakharyaschev, M., eds.: Proc. 24th Int. Workshop on Description Logics (DL'11). Volume 745 of CEUR Workshop Proceedings., CEUR-WS.org (2011) 334–344
6. Lawley, M.J., Bousquet, C.: Fast classification in Protégé: Snorocket as an OWL 2 EL reasoner. In: Proc. 6th Australasian Ontology Workshop (IAOA'10). (2010) 45–49
7. Schulz, S., Cornet, R., Spackman, K.A.: Consolidating SNOMED CT's ontological commitment. Applied Ontology **6**(1) (2011) 1–11
8. Suntisrivaraporn, B.: Module extraction and incremental classification: A pragmatic approach for ontologies. In Bechhofer, S., Hauswirth, M., Hoffmann, J., Koubarakis, M., eds.: ESWC. Volume 5021 of LNCS., Springer (June 1-5 2008) 230–244
9. Sirin, E., Parsia, B., Cuenca Grau, B., Kalyanpur, A., Katz, Y.: Pellet: A practical OWL-DL reasoner. J. of Web Semantics **5**(2) (2007) 51–53
10. Halaschek-Wiener, C., Parsia, B., Sirin, E.: Description logic reasoning with syntactic updates. In: OTM Conferences (1). (2006) 722–737
11. Cuenca Grau, B., Halaschek-Wiener, C., Kazakov, Y., Suntisrivaraporn, B.: Incremental classification of description logics ontologies. J. of Automated Reasoning **44**(4) (2010) 337–369
12. Gupta, A., Mumick, I.S., Subrahmanian, V.S.: Maintaining views incrementally. In Buneman, P., Jajodia, S., eds.: Proc. 1993 ACM SIGMOD Int. Conf. on Management of Data, Washington, D.C., ACM Press (May 26-28 1993) 157–166
13. Volz, R., Staab, S., Motik, B.: Incrementally maintaining materializations of ontologies stored in logic databases. J. of Data Semantics **2** (2005) 1–34
14. Barbieri, D.F., Braga, D., Ceri, S., Valle, E.D., Grossniklaus, M.: Incremental reasoning on streams and rich background knowledge. In: European Semantic Web Conference. (2010) 1–15
15. Kazakov, Y., Krötzsch, M., Simančík, F.: ELK: a reasoner for OWL EL ontologies. Technical report, University of Oxford (2012) available at http://elk.semanticweb.org.
16. Kazakov, Y., Klinov, P.: Incremental classification for OWL EL without bookkeeping. Technical report, University of Ulm (2013) available at http://elk.semanticweb.org.
17. Mungall, C.J., Bada, M., Berardini, T.Z., Deegan, J.I., Ireland, A., Harris, M.A., Hill, D.P., Lomax, J.: Cross-product extensions of the gene ontology. J. of Biomedical Informatics **44**(1) (2011) 80–86
18. Kazakov, Y.: Consequence-driven reasoning for Horn $\mathcal{SHIQ}$ ontologies. In Boutilier, C., ed.: Proc. 21st Int. Joint Conf. on Artificial Intelligence (IJCAI'09), IJCAI (2009) 2040–2045
19. Simančík, F., Kazakov, Y., Horrocks, I.: Consequence-based reasoning beyond Horn ontologies. In Walsh, T., ed.: Proc. 22nd Int. Joint Conf. on Artificial Intelligence (IJCAI'11), AAAI Press/IJCAI (2011) 1093–1098