Incremental Reasoning in $\mathcal{EL}^+$ 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 no longer hold. 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 still very efficient for small incremental changes in ontologies.

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 such information. The main idea is to split the derived conclusions into several partitions.

When some inference used to produce a conclusion in some partition is no longer valid as a result of a deletion, we simply re-compute all conclusions in this partition. Our hypothesis is that, if the number of partitions is sufficiently large and changes are relatively small, the proportion of the partitions affected by changes is small enough, so that re-computation of conclusions is not expensive. We describe a particular partitioning method for $\mathcal{EL}$ that does not require storing any additional information, 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 (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 collection of 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 axioms (to be on the safe side), but instead of checking which deleted conclusions are still derivable using remaining axioms (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 $\bot$. $\mathcal{EL}^+$ concepts are defined using the grammar $\mathbf{C} ::= A \mid \top \mid C_1 \sqcap C_2 \mid \exists R.C$, where A is an *atomic concept*, R an *atomic role*, and C, C_1, $C_2 \in \mathbf{C}$. $\mathcal{EL}^+$ *axiom* is either a *concept inclusion* $C_1 \sqsubseteq C_2$ for C_1, $C_2 \in \mathbf{C}$, a *role inclusion* $R \sqsubseteq S$, or a *role composition* $R_1 \circ R_2 \sqsubseteq S$, where R, R_1, R_2, S are atomic roles. $\mathcal{EL}^+$ *ontology* $\mathcal{O}$ is a finite set of $\mathcal{EL}^+$ axioms. Entailment of axioms by an ontology is defined in a usual way; a formal definition can be found in Appendix A. A concept C is *subsumed* by D w.r.t. $\mathcal{O}$ if $\mathcal{O} \models C \sqsubseteq D$. In this case, we call $C \sqsubseteq D$ an *entailed subsumption*. 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 of the rule* R (notation $\|\mathsf{R}\|$) is the number of inferences *inf* $\in \mathsf{R}$. Within this paper, we commonly 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 $\mathsf{R}(Exp)$ ($\mathsf{R}[Exp]$) 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, $\mathsf{R}[Exp_1](Exp_2)$ 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 $\mathsf{R}(Exp_2)[Exp_1]$. For simplicity, we write $\mathsf{R}()$, $\mathsf{R}[]$, $\mathsf{R}(exp)$, and $\mathsf{R}[exp]$ instead of $\mathsf{R}(\emptyset)$, $\mathsf{R}[\emptyset]$, $\mathsf{R}(\{exp\})$, and $\mathsf{R}[\{exp\}]$ 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

$\mathbf{R_0}\ \dfrac{}{C \sqsubseteq C}$: C occurs in $\mathcal{O}$

$\mathbf{R_\top}\ \dfrac{}{C \sqsubseteq \top}$: C and $\top$ occur in $\mathcal{O}$

$\mathbf{R_\sqsubseteq}\ \dfrac{C \sqsubseteq D}{C \sqsubseteq E}$: $D \sqsubseteq E \in \mathcal{O}$

$\mathbf{R_\sqcap^-}\ \dfrac{C \sqsubseteq D_1 \sqcap D_2}{C \sqsubseteq D_1 \quad C \sqsubseteq D_2}$

$\mathbf{R_\sqcap^+}\ \dfrac{C \sqsubseteq D_1 \quad C \sqsubseteq D_2}{C \sqsubseteq D_1 \sqcap D_2}$: $D_1 \sqcap D_2$ occurs in $\mathcal{O}$

$\mathbf{R_\exists}\ \dfrac{E \sqsubseteq \exists R.C \quad C \sqsubseteq D}{E \sqsubseteq \exists S.D}$: $\begin{array}{l}\exists S.D \text{ occurs in } \mathcal{O} \\ R \sqsubseteq^*_\mathcal{O} S\end{array}$

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

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

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;
```

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. Example 1 in Appendix C demonstrates how to apply the rules in Figure 1 to compute the classification.

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 expressions by applying inferences in R and collects those expressions 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

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;
```

inferences $\mathsf{R}() \subseteq \mathsf{R}$ (lines 2–3), and then in a cycle (lines 4–9), repeatedly takes the next expression $exp \in \mathsf{Todo}$, if any, inserts it into $\mathsf{Closure}$ 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 $\mathsf{Closure}$. The conclusions of such inferences are then inserted into Todo. Please see Example 2 in Appendix C illustrating an execution of Algorithm 1.

Let $\mathsf{Closure}'$ be the set computed by Algorithm 1. Note that the algorithm performs as many insertions into Todo as there are inferences in $\mathsf{R}(\mathsf{Closure}')$ because every inference $inf \in \mathsf{R}(\mathsf{Closure}')$ is eventually applied, and an inference cannot apply more than once because of line 6. Therefore, the running time of Algorithm 1 can be estimated as $O(\|\mathsf{R}(\mathsf{Closure}')\|)$, provided that 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.

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 $\mathsf{Closure}$ the closure under R. The objective is to compute the closure under the inferences in $(\mathsf{R} \setminus \mathsf{R}^-) \cup \mathsf{R}^+$, using $\mathsf{Closure}$, R, R^+, or R^-, if necessary.

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

Algorithm 3: Update modulo deletions (incomplete)

input : R, R^-: sets of inferences, $\mathsf{Closure}$: the closure under R
output : $\mathsf{Closure}$: 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;
```

Todo with conclusions of new inferences $\mathit{inf} \in (\mathsf{R}^+ \setminus \mathsf{R})$ applicable to 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 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. Therefore, the running time of Algorithm 2 can be estimated as $O(\|(\mathsf{R} \cup \mathsf{R}^+)(\mathsf{Closure}')\| - \|\mathsf{R}(\mathsf{Closure})\|)$, which saves us $O(\|\mathsf{R}(\mathsf{Closure})\|)$ operations compared to applying Algorithm 1 from scratch. Note that it is essential that we start with the closure under R. If we start with a set that is not closed under R, we may lose conclusions because the inferences in $\mathsf{R}(\mathsf{Closure})$ are never 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 expressions that should be removed from 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 together with conclusions of inferences from 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 in Todo during the initialization (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 $\mathsf{R} = \{/a,\ a/b,\ b/a\}$ (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 in 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]. But to efficiently check whether an expression is a conclusion of some inference from $\mathsf{Closure}$, one either needs to record how conclusions where produced, or build indexes that help to identify matching premises in $\mathsf{Closure}$ 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' the remaining inferences in R to the set $\mathsf{Closure}$ produced by Algorithm 3. This would require $O(\|\mathsf{R}(\mathsf{Closure})\|)$ operations—the same as for computing $\mathsf{Closure}$ from scratch using Algorithm 1. Most of the inferences are likely to be already applied to $\mathsf{Closure}$, 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, this approach saves us $\|\mathsf{R}_1(\mathsf{Closure}_1)\|$ rule applications compared to computing the closure from scratch.

Let $\mathsf{Closure}$ be the set in the input of Algorithm 3, and $\mathsf{Closure}'$ the set obtained in the output. Similarly to Algorithm 2, we can estimate the running time of Algorithm 3 as $O(\|\mathsf{R}(\mathsf{Closure})\| - \|(\mathsf{R} \setminus \mathsf{R}^-)(\mathsf{Closure}')\|)$. 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* $(\mathsf{R} \setminus \mathsf{R}^-)(\mathsf{Closure}')$.

The proof of Lemma 1 can be found in Appendix D.

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

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* $\in$ R(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;
```

marked as 'broken'. We then re-apply inferences that can produce conclusions in broken partitions to 'repair' the closure.

Formally, let **Pts** be a fixed countable set of *partition identifiers* (short *partitions*), and every expression *exp* $\in$ **Exp** be assigned with exactly one partition *exp*.partition $\in$ **Pts**. For an inference rule R and a set of partitions *Pts* $\subseteq$ **Pts**, let R$\langle$*Pts*$\rangle$ be the set of inferences *inf* $\in$ R such that *inf*.conclusion.partition $\in$ *Pts* and *exp*.partition $\notin$ *Pts* for every *exp* $\in$ *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 Closure in line 10, we add *exp*.partition to a special set of partitions Broken. This set is then used to repair Closure in Algorithm 4. The goal of the algorithm is to collect in the queue Todo the conclusions of inferences in R(Closure) that are missing in Closure. This is done by applying all possible inferences *inf* $\in$ R(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 R$\langle$Broken$\rangle$(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* $\in$ R$\langle$Broken$\rangle$(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 in the output Todo (lines 4, 13). Otherwise, we put it into a special queue ToRepair (lines 6, 15), and repeatedly apply for each $exp \in$ ToRepair the 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* $\mathsf{Todo} = \{inf.\text{conclusion} \mid inf \in \mathsf{R}(\mathsf{Closure})\} \setminus \mathsf{Closure}$.

The proof of Lemma 2 can be found in Appendix D.

After computing the repair Todo of the set 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 algorithm follows from Lemma 1, Lemma 2, and the correctness of our modification of Algorithm 2 when Todo is initialized with missing conclusions of $\mathsf{R}(\mathsf{Closure})$. The complete algorithm for updating the closure can be found in Appendix B, and its detailed correctness proof in Appendix D.

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 every expression, 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 deleted expressions that are still derivable (the conclusions of $\mathsf{R}\langle\mathsf{Broken}\rangle(\mathsf{Closure})$ 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 (as many as there are concepts in the input ontology) and also has the advantage that the inferences $\mathsf{R}\langle Pts\rangle$ for any set *Pts* of partitions can be easily identified. Indeed, note that every conclusion of a rule in

Table 1. 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.	|Broken|	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

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, $\mathsf{R}\langle Pts\rangle$ consists of exactly those inferences in $\mathbf{R_0}$ and $\mathbf{R}_\top$ of R for which $C \in Pts$.

5.2 Dealing with 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 a range of further optimizations that avoid unnecessary applications of some rules. For example, it is not necessary to apply $\mathbf{R}^-_\sqcap$ to conclusions of $\mathbf{R}^+_\sqcap$, and it is not necessary to apply $\mathbf{R}_\exists$ and $\mathbf{R}_\circ$ if its existential premises were 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 deletions could be missed. 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 Example 4 in Appendix C demonstrating 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 Closure are encountered, but it is sufficient to insert only conclusions of optimized inferences into Todo.

5.3 Experimental Evaluation

We have implemented the procedure described in Section 4.3 in the OWL EL reasoner ELK,[1] and performed some preliminary experiments to evaluate its performance. We used three large OWL EL ontologies listed in Table 1 which are frequently used in evaluations of $\mathcal{EL}$ reasoners [3–6]: the Gene Ontology GO [16] with $84,955$ axioms, an

[1] The incremental classification procedure is enabled by default in the latest nightly builds: `http://code.google.com/p/elk-reasoner/wiki/GettingElk`

$\mathcal{EL}^+$-restricted version of the GALEN ontology with $36,547$ axioms,[2] and the January 2013 release of SNOMED CT with $296,529$ axioms.[3]

The recent change history of GO is readily available from the public SVN 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. All experiments were performed on a PC with Intel Core i5-2520M 2.50GHz CPU, running Java 1.6 with 4GB of RAM available to JVM.

The results of the initial and incremental classifications are given in Table 1 (more details can be found in Appendix E). 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 $|\mathsf{Broken}|$ 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, the repair stage takes a relatively small fraction of the total time.

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 (for example, it is implemented in the same concurrent way as the base procedure in ELK [4]). Our experiments, though being preliminary due to the shortage of revision histories for real-life $\mathcal{EL}$ ontologies, show that its performance enables nearly real-time incremental updates. 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 [17, 18].

The main idea of the method is exploiting the granularity of the $\mathcal{EL}^+$ reasoning procedure. Specifically, subsumers of concepts can often be computed independently of each other. This 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.

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

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

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

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 Taylor, K., Meyer, T., Orgun, M., eds.: Proc. 6th Australasian Ontology Workshop (IAOA'10). Volume 122 of Conferences in Research and Practice in Information Technology., Australian Computer Society Inc. (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 from `http://code.google.com/p/elk-reasoner/wiki/Publications`.
16. 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

17. 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
18. 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

Table 2. The syntax and semantics of $\mathcal{EL}^+$

	Syntax	Semantics
Roles:		
atomic role	R	$R^{\mathcal{I}}$
Concepts:		
atomic concept	A	$A^{\mathcal{I}}$
top	$\top$	$\Delta^{\mathcal{I}}$
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	$R_1 \circ R_2 \sqsubseteq S$	$R_1^{\mathcal{I}} \circ R_2^{\mathcal{I}} \subseteq S^{\mathcal{I}}$

A Definitions

This section gives a detailed definition of syntax and semantics of $\mathcal{EL}^+$ introduced in Section 3.1. $\mathcal{EL}^+$ 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 2. 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 2. $\mathcal{I}$ *satisfies* an axiom α (written $\mathcal{I} \models \alpha$) if the corresponding condition in Table 2 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.

B Algorithms

This section presents a detailed description of the method for incremental deductive closure computation using partitions presented in Section 4.3. The method is described by Algorithm 5, which consists of three stages:

1. Deletion (lines 2–11): This stage is almost identical to Algorithm 3 except that we additionally collect in Broken the partitions of the removed expressions (line 11).
2. Repair (12–26): As we have discussed in Section 4.2, after deletions, the resulting set Closure might not be closed under $\mathsf{R} = \mathsf{R} \setminus \mathsf{R}^-$. The goal of this stage is to 'repair' this defect by computing in Todo those conclusions of $\mathsf{R}(\mathsf{Closure})$ that do not occur in Closure. We know that such conclusions must have been deleted in the previous stage, so their partitions are saved in Broken. We find such conclusions by

Algorithm 5: Incremental Update of Deductive Closure using Partitions

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

```
1  Todo, Broken, ToRepair, Repaired ← ∅;
   /* ====================== Deletion ====================== */
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         Broken.add(exp.partition);          /* add a broken partition */
   /* ====================== Repair ====================== */
12 for inf ∈ R⟨Broken⟩(Closure) do                            /* initialize */
13     if inf.conclusion ∉ Closure then
14         Todo.add(inf.conclusion);
15     else
16         ToRepair.add(inf.conclusion);
17 while ToRepair ≠ ∅ do                                       /* close */
18     exp ← ToRepair.takeNext();
19     if exp ∉ Repaired then
20         for inf ∈ R[exp](Closure) do
21             if inf.conclusion.partition ∈ Broken then
22                 if inf.conclusion ∉ Closure then
23                     Todo.add(inf.conclusion);
24                 else
25                     ToRepair.add(inf.conclusion);
26         Repaired.add(exp);
   /* ====================== Addition ====================== */
27 for inf ∈ (R⁺ \ R)(Closure) do                            /* initialize */
28     Todo.add(inf.conclusion);
29 R ← R ∪ R⁺;
30 while Todo ≠ ∅ do                                          /* close */
31     exp ← Todo.takeNext();
32     if exp ∉ Closure then
33         Closure.add(exp);
34         for inf ∈ R[exp](Closure) do
35             Todo.add(inf.conclusion);
36 return Closure;
```

applying inferences in $\mathsf{R}(\mathsf{Closure})$ producing expressions with partitions in Broken. There can be two types of such inferences: those whose premises do not belong to partitions in Broken, and those that have some premises belonging to partitions in Broken. The former inferences will be applied during initialization (lines 12–16). To implement the latter inferences, we use the queue ToRepair to iterate over those expressions in Closure whose partition is in Broken. Whenever we apply inferences to such expressions, we add them into the set Repaired to make sure we do not consider them again. Every time we derive a conclusion whose partition is in Broken, we either add it into Todo if it does not occur in Closure, or otherwise add it into ToRepair. We then process expressions *exp* in ToRepair that are not yet in Repaired in a loop (lines 17–26), in which we apply inferences in $\mathsf{R}[exp](\mathsf{Closure})$, and insert those conclusions whose partitions are in Broken into Todo or ToRepair similarly as above.

3. Addition (lines 27–35): This stage is identical to Algorithm 2, except that the initial set Closure is not closed under R. But since Todo contains all those conclusions of $\mathsf{R}(\mathsf{Closure})$ that do not occur in Closure as ensured by the repair stage, it is guaranteed that all inferences in $(\mathsf{R} \cup \mathsf{R}^+)$ will be applied to Closure.

Please see Example 3 in Appendix C illustrating an execution of Algorithm 5 with the partition assignment for $\mathcal{EL}^+$ described in Section 5.

C Examples

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:

$$A \sqsubseteq A \quad \text{by } \mathbf{R_0} \text{ since } A \text{ occurs in } \mathcal{O}, \tag{1}$$
$$B \sqsubseteq B \quad \text{by } \mathbf{R_0} \text{ since } B \text{ occurs in } \mathcal{O}, \tag{2}$$
$$C \sqsubseteq C \quad \text{by } \mathbf{R_0} \text{ since } C \text{ occurs in } \mathcal{O}, \tag{3}$$
$$\exists R.B \sqsubseteq \exists R.B \quad \text{by } \mathbf{R_0} \text{ since } \exists R.B \text{ occurs in } \mathcal{O}, \tag{4}$$
$$\exists S.A \sqsubseteq \exists S.A \quad \text{by } \mathbf{R_0} \text{ since } \exists S.A \text{ occurs in } \mathcal{O}, \tag{5}$$
$$\exists H.B \sqsubseteq \exists H.B \quad \text{by } \mathbf{R_0} \text{ since } \exists H.B \text{ occurs in } \mathcal{O}, \tag{6}$$
$$\exists S.C \sqsubseteq \exists S.C \quad \text{by } \mathbf{R_0} \text{ since } \exists S.C \text{ occurs in } \mathcal{O}, \tag{7}$$
$$A \sqsubseteq \exists R.B \quad \text{by } \mathbf{R_\sqsubseteq} \text{ to (1) using (ax1)}, \tag{8}$$
$$B \sqsubseteq \exists S.A \quad \text{by } \mathbf{R_\sqsubseteq} \text{ to (2) using (ax4)}, \tag{9}$$
$$\exists R.B \sqsubseteq \exists H.B \quad \text{by } \mathbf{R_\exists} \text{ to (4) and (2) using (ax3)}, \tag{10}$$
$$\exists H.B \sqsubseteq C \quad \text{by } \mathbf{R_\sqsubseteq} \text{ to (6) using (ax2)}, \tag{11}$$
$$\exists S.C \sqsubseteq C \quad \text{by } \mathbf{R_\sqsubseteq} \text{ to (7) using (ax5)}, \tag{12}$$

$$A \sqsubseteq \exists H.B \qquad \text{by } \mathbf{R}_{\exists} \text{ to (8) and (2) using (ax3),} \qquad (13)$$

$$\exists R.B \sqsubseteq C \qquad \text{by } \mathbf{R}_{\sqsubseteq} \text{ to (10) using (ax2),} \qquad (14)$$

$$A \sqsubseteq C \qquad \text{by } \mathbf{R}_{\sqsubseteq} \text{ to (13) using (ax2),} \qquad (15)$$

$$\exists S.A \sqsubseteq \exists S.C \qquad \text{by } \mathbf{R}_{\exists} \text{ to (5) and (15),} \qquad (16)$$

$$B \sqsubseteq \exists S.C \qquad \text{by } \mathbf{R}_{\exists} \text{ to (9) and (15),} \qquad (17)$$

$$\exists S.A \sqsubseteq C \qquad \text{by } \mathbf{R}_{\sqsubseteq} \text{ to (16) using (ax5),} \qquad (18)$$

$$B \sqsubseteq C \qquad \text{by } \mathbf{R}_{\sqsubseteq} \text{ to (17) using (ax5).} \qquad (19)$$

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 subsumptions between atomic concepts entailed by $\mathcal{O}$.

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

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

The following example illustrates the execution of Algorithm 5 for updating the deductive closure under inferences in Figure 1 using the partition-based approach described in Section 4.3.

Example 3 (Example 1 continued). Let us see how the closure (1)–(19) is updated using Algorithm 5 after deletions of the axiom (ax5) from the ontology $\mathcal{O}$. The deletion of this axiom results in R^{-} consisting of $\mathbf{R_0}$ and $\mathbf{R}_{\exists}$ for $\exists S.C$ since $\exists S.C$ does not occur in $\mathcal{O}$ anymore, and $\mathbf{R}_{\sqsubseteq}$ for $\exists S.C \sqsubseteq C$ since this axiom was deleted from $\mathcal{O}$. These inferences produce the conclusions (7), (12), and (16)–(19) in Todo during the initialization of deletions (lines 2–3). Only these conclusions will be removed during closure (lines 5–11), and their partitions will be collected in $\mathsf{Broken} = \{\exists S.C,\ \exists S.A,\ B\}$.

During the repair stage, we first need to apply the inferences in $\mathsf{R}\langle\mathsf{Broken}\rangle$ to Closure (lines 12–16), that is, the instances of $\mathbf{R_0}$ for all concepts in Broken except for $\exists S.C$, which does not occur in $\mathcal{O}$ anymore. This adds (2) and (5) into ToRepair since they still occur in Closure. When processing these conclusions during closure (lines 17–26), the conclusion (9) of an inference from (2) is added into ToRepair because its partition $B \in \mathsf{Broken}$ and it still occurs in Closure. Note that other conclusions (10) and (13) of inferences from (2) are ignored because their partitions $\exists R.B$ and A are not in Broken. Also, rule $\mathbf{R}_{\exists}$ is no longer applied to (9) and (15) since $\exists S.C$ does not occur in $\mathcal{O}$ anymore. So, no further conclusions are added into ToRepair or into Todo. At the end of this stage, therefore, Repaired contains (2), (5), and (9), and Todo remains empty.

During the addition stage, no inferences will be applied since Todo is empty and there were no axiom additions. The updated closure, thus, consists of subsumptions (1)–(6), (8)–(11), and (13)–(15).

The following example demonstrates why some rule optimizations cannot be applied in Algorithm 3 as discussed in Section 5.2.

Example 4. Consider the following $\mathcal{EL}^+$ ontology $\mathcal{O}$:
(ax1): $A \sqsubseteq \exists R.B$ (ax2): $B \sqsubseteq C$ (ax3): $C \sqsubseteq B$ (ax4): $\exists R.C \sqsubseteq \exists R.C$

Let us apply an optimized version of Algorithm 1 for rules in Figure 1, in which we do not apply rule $\mathbf{R}_\exists$ if its left premise is obtained by rule $\mathbf{R}_\exists$:

$$A \sqsubseteq A \quad \text{by } \mathbf{R_0} \text{ since } A \text{ occurs in } \mathcal{O}, \tag{20}$$
$$B \sqsubseteq B \quad \text{by } \mathbf{R_0} \text{ since } B \text{ occurs in } \mathcal{O}, \tag{21}$$
$$C \sqsubseteq C \quad \text{by } \mathbf{R_0} \text{ since } C \text{ occurs in } \mathcal{O}, \tag{22}$$
$$\exists R.B \sqsubseteq \exists R.B \quad \text{by } \mathbf{R_0} \text{ since } \exists R.B \text{ occurs in } \mathcal{O}, \tag{23}$$
$$\exists R.C \sqsubseteq \exists R.C \quad \text{by } \mathbf{R_0} \text{ since } \exists R.C \text{ occurs in } \mathcal{O}, \tag{24}$$
$$A \sqsubseteq \exists R.B \quad \text{by } \mathbf{R}_\sqsubseteq \text{ to (20) using (ax1)}, \tag{25}$$
$$B \sqsubseteq C \quad \text{by } \mathbf{R}_\sqsubseteq \text{ to (21) using (ax2)}, \tag{26}$$
$$C \sqsubseteq B \quad \text{by } \mathbf{R}_\sqsubseteq \text{ to (22) using (ax3)}, \tag{27}$$
$$\exists R.B \sqsubseteq \exists R.C \quad \text{by } \mathbf{R}_\exists \text{ to (23) and (26) using (ax4)}, \tag{28}$$
$$\exists R.C \sqsubseteq \exists R.B \quad \text{by } \mathbf{R}_\exists \text{ to (24) and (27) using (ax1)}, \tag{29}$$
$$A \sqsubseteq \exists R.C \quad \text{by } \mathbf{R}_\exists \text{ to (25) and (26) using (ax4)}. \tag{30}$$

Rule $\mathbf{R}_\sqsubseteq$ can be further applied to (26) using (ax3), to (27) using (ax2), to (24) using (ax4), and to (28) using (ax4), producing respectively (21), (22), (24), and (28) that have been already derived. Note that (28), (29), and (30) cannot be further used as the left premise of rule $R_\exists$ because they have been obtained using $R_\exists$ and we use the optimization that prevents such inferences. But even if we apply such inferences, they produce only subsumptions that are already derived; that is why this optimization does not have any effect on the resulting closure (except for performing fewer inferences).

Assume now that axioms (ax2) and (ax3) are deleted and let us apply Algorithm 3 for the computed set Closure (20)–(30) using the same optimized inference rules. During initialization (lines 2–3) the following inferences for (ax2) and (ax3) will be applied:

$$B \sqsubseteq C \quad \text{by } \mathbf{R}_\sqsubseteq \text{ to (21) using (ax2)}, \tag{31}$$
$$C \sqsubseteq C \quad \text{by } \mathbf{R}_\sqsubseteq \text{ to (27) using (ax2)}, \tag{32}$$
$$C \sqsubseteq B \quad \text{by } \mathbf{R}_\sqsubseteq \text{ to (22) using (ax3)}, \tag{33}$$
$$B \sqsubseteq B \quad \text{by } \mathbf{R}_\sqsubseteq \text{ to (26) using (ax3)}. \tag{34}$$

In the main loop (lines 5–10) for each resulting conclusion that occurs in Closure, the inferences are applied, and the conclusion is deleted from Closure. Assume that this happens in the following order (so far the order in which elements are taken from Todo was not important for correctness of the algorithm): first we process (33):

$$\exists R.C \sqsubseteq \exists R.B \quad \text{by } \mathbf{R}_\exists \text{ to (24) and (33) using (ax1)}, \tag{35}$$
$$\exists R.B \sqsubseteq \exists R.B \quad \text{by } \mathbf{R}_\exists \text{ to (28) and (33) using (ax1)}, \tag{36}$$

$$A \sqsubseteq \exists R.B \qquad \text{by } \mathbf{R}_{\exists} \text{ to (30) and (33) using (ax1),} \qquad (37)$$

after that we process (35), (36), (37), (31), (32), and finally (34).

Note that when processing (35), rule $\mathbf{R}_{\exists}$ is not applied to (35) and (26) using (ax4) because (35) was obtained using rule $\mathbf{R}_{\exists}$. Therefore, (24) does not get deleted. Similarly, rule $\mathbf{R}_{\exists}$ is not applied when processing (36) and (37), so (28) and (30) do not get deleted as well. Since (28) and (30) can only be obtained by rule $\mathbf{R}_{\exists}$ from (36) and (37), both of which get deleted after processing, the subsumptions (28) and (30) remain in Closure, even though they are not implied by the remaining axioms (ax1) and (ax4). The problem takes place because (28) and (30) were initially obtained by rule $\mathbf{R}_{\exists}$ from the same premises (23) and (25), but which were derived using different rules ($\mathbf{R_0}$ and $\mathbf{R}_{\sqsubseteq}$ respectively), to which the existential optimization did not apply.

D Proofs

This section presents the omitted proofs of Lemma 1 and Lemma 2, and the correctness proof of Algorithm 5.

Proof of Lemma 1. 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 *inf* $\in \mathsf{R}(\mathsf{Closure}_1)$. Then there are two possibilities:

1. *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. *inf* $\in (\mathsf{R} \setminus \mathsf{R}^-)(\mathsf{Closure}')$: Since *inf* $\in \mathsf{R}(\mathsf{Closure}_1)$ and $\mathsf{Closure}' \cap \mathsf{Closure}_1 \subseteq \mathsf{Closure}''$, we have *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* $\in (\mathsf{R} \setminus \mathsf{R}^-)(\mathsf{Closure}'')$. Therefore *inf*.conclusion $\in \mathsf{Closure}'' \subseteq \mathsf{Closure}_1$.

Now, since $\mathsf{Closure}_1 \subseteq \mathsf{Closure}$ is closed under R and 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. □

Proof of Lemma 2. Let $\mathsf{Closure}' = \{\textit{inf}.\mathsf{conclusion} \mid \textit{inf} \in \mathsf{R}(\mathsf{Closure})\}$. 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 *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 $\mathsf{Closure}_1 = \{\textit{exp} \in \mathsf{Closure} \mid \textit{exp}.\mathsf{partition} \notin \mathsf{Broken}\}$. Then it is easy to see from Algorithm 4 that for every *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 $\notin$ Broken then by assumption of Algorithm 4, since *inf* $\in \mathsf{R}(\mathsf{Closure})$ and *inf*.conclusion.partition $\notin$ Broken, we must have *inf*.conclusion $\in$ Closure, and thus *inf*.conclusion $\in \mathsf{Closure}_1$.

If *inf*.conclusion.partition $\in$ Broken, there are two cases possible. Either *inf* $\in \mathsf{R}(\mathsf{Closure}_1)$, thus, *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 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 $\mathsf{R}(\mathsf{Closure})$. 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. □

Theorem 1. *Let* R, R^+, R^- *be the input of Algorithm 5, and* $\mathsf{Closure}_3$ *the set obtained in the output. Then* $\mathsf{Closure}_3$ *is the closure under* $(\mathsf{R} \setminus \mathsf{R}^-) \cup \mathsf{R}^+$.

Proof. Let $\mathsf{Closure}_1$ and Broken be the sets obtained after the deletion stage of the algorithm. Then by Lemma 1, for $\mathsf{Closure}'$ the closure under $(\mathsf{R} \setminus \mathsf{R}^-)$, we have $\mathsf{Closure}_1 \subseteq \mathsf{Closure}'$. Furthermore, it is easy to see that Broken contains the partitions for all expressions in $\mathsf{Closure} \setminus \mathsf{Closure}_1$, and, in particular, of those in $\mathsf{Closure}' \setminus \mathsf{Closure}_1$ since, obviously, $\mathsf{Closure}' \subseteq \mathsf{Closure}$.

Let $\mathsf{Closure}_2$ and Todo_2 be contents of Closure and Todo after the repair stage of Algorithm 5. Clearly, $\mathsf{Closure}_2 = \mathsf{Closure}_1$ since Closure does not change in this stage. Then by Lemma 2, Todo_2 consists of all conclusions of $(\mathsf{R} \setminus \mathsf{R}^-)(\mathsf{Closure}_2)$ that do not occur in $\mathsf{Closure}_2$. In particular, $\mathsf{Todo}_2 \subseteq \mathsf{Closure}'$.

Let $\mathsf{Closure}''$ be the closure under $(\mathsf{R} \setminus \mathsf{R}^-) \cup \mathsf{R}^+$. Clearly, $\mathsf{Closure}_2 \cup \mathsf{Todo}_2 \subseteq \mathsf{Closure}' \subseteq \mathsf{Closure}''$. Let $\mathsf{Closure}_3$ be the content of Closure after the addition stage of Algorithm 5. We need to prove that $\mathsf{Closure}_3 = \mathsf{Closure}''$. Since in the addition stage, Closure is extended only with content of Todo, which, in turn, can contain only elements of Todo_2 or consequences of inferences in $\mathsf{R}(\mathsf{Closure})$, we have $\mathsf{Closure}_3 \subseteq \mathsf{Closure}''$. We now show that $\mathsf{Closure}_3$ is closed under $(\mathsf{R} \setminus \mathsf{R}^-) \cup \mathsf{R}^+$. For this, take any inference *inf* $\in ((\mathsf{R} \setminus \mathsf{R}^-) \cup \mathsf{R}^+)(\mathsf{Closure}_3)$. We need to prove that *inf*.conclusion $\in \mathsf{Closure}_3$. There are three cases possible:

1. *inf* $\in (\mathsf{R} \setminus \mathsf{R}^-)(\mathsf{Closure}_2)$. Then *inf*.conclusion $\in \mathsf{Todo}_2 \subseteq \mathsf{Closure}_3$.
2. *inf* $\in (\mathsf{R}^+ \setminus (\mathsf{R} \setminus \mathsf{R}^-))(\mathsf{Closure}_2)$. Then *inf*.conclusion $\in \mathsf{Closure}_3$ due to the initialization stage of additions (lines 27–28).
3. *inf* $\in ((\mathsf{R} \setminus \mathsf{R}^-) \cup \mathsf{R}^+)(\mathsf{Closure}_3) \setminus ((\mathsf{R} \setminus \mathsf{R}^-) \cup \mathsf{R}^+)(\mathsf{Closure}_2)$. That is, *inf* has at least one premise in $\mathsf{Closure}_3 \setminus \mathsf{Closure}_2$. Then *inf* should be applied in the main loop of the addition stage (lines 30–35). Therefore, *inf*.conclusion $\in \mathsf{Closure}_3$.

Since $\mathsf{Closure}_3 \subseteq \mathsf{Closure}''$ is closed under $(\mathsf{R} \setminus \mathsf{R}^-) \cup \mathsf{R}^+$ and $\mathsf{Closure}''$ is the closure under $(\mathsf{R} \setminus \mathsf{R}^-) \cup \mathsf{R}^+$, we have $\mathsf{Closure}_3 = \mathsf{Closure}''$, as required. □

E Detailed Experimental Results

Table 3 presents the detailed view of the results summarized in Table 1. For each performance metric, the number of inferences or time, it shows the average value as well as the [min–max] interval. In addition, the table presents numbers for the initialization

parts of the deletion and addition stages of Algorithm 5 separately since initialization often takes longer than closure. The main reason is that enumeration of all inferences in $\mathsf{R}^{-}(\mathsf{Closure})$ (resp. $(\mathsf{R}^{+} \setminus \mathsf{R})(\mathsf{Closure})$) is currently not very efficient. We are investigating some further optimizations of these parts. For the repair stage, initialization and closure are implemented in one loop, as there is only a limited interaction between these parts. Hence we present only the overall figures for this stage.

It can be seen that all three stages (deletion, repair, and addition) exhibit substantial variability in terms of the number of inferences. For the case of GO, this can partly be explained by the varying number of changes, as shown in the first column. For GALEN and SNOMED CT it is explained by the randomness of the change generation algorithm which sometimes picks axioms which affect frequently used concepts in the ontology (and, thus, require more effort during the incremental updates). One can speculate that such changes are less likely to occur during real-life editing process (though to justify this we would need more ontology version data). Also, observe that the dependency of the running time on the number of inferences is non-linear since many inferences are grouped and applied together for performance reasons.

Table 3. Detailed number of inferences and running times. The results for each incremental stage are averaged over all incremental changes (for GO only over changes with a non-empty set of deletions). The min and max values over all changes are presented as intervals.

Ontology	Stage	Averaged performance metrics # inferences		Time (ms)	
GO (r560)	del. init.	641	[1–18,935]	39	[6–383]
(+*add.* -*del.*):	del. closure	61,743	[2–1,739,034]	9	[0–227]
+84 [2–1,495]	repair	17,628	[1–537,095]	6	[0–149]
-26 [1–409]	add. init.	721	[2–17,855]	59	[7–320]
|Broken|:	add. closure	58,212	[38–1,571,624]	7	[0–232]
560 [1–15,928]	**total**	**138,945**	**[66–3,875,225]**	**134**	**[26–767]**
GALEN	del. init.	16	[1, 961]	16	[3–115]
(+*add.* -*del.*):	del. closure	3,428	[1–125,580]	2	[0–77]
± 1	repair	4,321	[0–190,271]	4	[0–370]
|Broken|:	add. init.	9	[1–479]	12	[3–423]
36 [1–956]	add. closure	3,046	[1–83,464]	1	[0–25]
	total	**10,820**	**[11–346,939]**	**39**	**[12–413]**
GALEN	del. init.	219	[13–8,688]	50	[15–392]
(+*add.* -*del.*):	del. closure	68,575	[528–2,196,705]	16	[0–542]
± 10	repair	37,583	[118–790,776]	17	[0–196]
|Broken|:	add. init.	137	[10–8,329]	39	[14–97]
473 [8–12,822]	add. closure	49,525	[967–1,241,359]	13	[0–516]
	total	**156,039**	**[3358–3,648,026]**	**147**	**[55–1,266]**
GALEN	del. init.	3,753	[379–43,094]	82	[25–181]
(+*add.* -*del.*):	del. closure	590,667	[28,684–2,171,376]	132	[7–707]
± 100	repair	314,666	[53,804–2,720,553]	96	[23–997]
|Broken|:	add. init.	1,777	[198–32,425]	57	[18–147]
4,508 [374–25,704]	add. closure	424,685	[43,170–1,344,674]	109	[10–594]
	total	**1,335,548**	**[168,293–4,453,413]**	**515**	**[177–1,895]**
SNOMED CT	del. init.	45	[1–3,378]	119	[37–2,186]
(+*add.* -*del.*):	del. closure	3,977	[2–342,960]	1	[0–112]
± 1	repair	423	[0–23,010]	1	[0–15]
|Broken|:	add. init.	20	[1–1,135]	67	[35–1,993]
64 [1–1,136]	add. closure	2,867	[3–226,842]	1	[0–72]
	total	**7,331**	**[8–596,525]**	**232**	**[81–2,724]**
SNOMED CT	del. init.	600	[11–42,705]	411	[197–2,194]
(+*add.* -*del.*):	del. closure	41,426	[443–2,703,165]	9	[0–587]
± 10	repair	8,343	[6–767,567]	4	[0–216]
|Broken|:	add. init.	207	[10–12,724]	341	[138–2,395]
251 [2–9,971]	add. closure	31,759	[514–1,808,499]	8	[0–409]
	total	**82,335**	**[1,503–4,760,242]**	**789**	**[408–3,009]**
SNOMED CT	del. init.	4,968	[428–105,150]	542	[375–3,169]
(+*add.* -*del.*):	del. closure	559,036	[20,654–9,425,976]	120	[5–4,651]
± 100	repair	138,633	[2,086–8,113,134]	56	[1–2,970]
|Broken|:	add. init.	2,280	[206–45,670]	445	[284–2,782]
3,577 [133–58,090]	add. closure	411,975	[20,201–7,000,073]	100	[5–1,960]
	total	**1,116,892**	**[51,266–19,314,629]**	**1,376**	**[901–7,762]**