# Goal-Directed Tracing of Inferences in EL Ontologies

Yevgeny Kazakov and Pavel Klinov

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

**Abstract.** $\mathcal{EL}$ is a family of tractable Description Logics (DLs) that is the basis of the OWL 2 EL profile. Unlike for many expressive DLs, reasoning in $\mathcal{EL}$ can be performed by computing a deductively-closed set of logical consequences of some specific form. In some ontology-based applications, e.g., for ontology debugging, knowing the logical consequences of the ontology axioms is often not sufficient. The user also needs to know from which axioms and how the consequences were derived. Although it is possible to keep track of all inferences applied during reasoning, this is usually not done in practice to avoid the overheads. In this paper, we present a goal-directed method that can generate inferences for selected consequences in the deductive closure without re-applying all rules from scratch. We provide an empirical evaluation demonstrating that the method is fast and economical for large $\mathcal{EL}$ ontologies. Although the main benefits are demonstrated for $\mathcal{EL}$ reasoning, the method can be easily extended to other procedures based on deductive closure computation using fixed sets of rules.

## 1 Introduction and Motivation

The majority of existing DL reasoners are based on optimized *(hyper)tableau*-based procedures which essentially work by trying to construct counter-models for the entailment [19, 15, 20]. If the reasoner could not find a counter-model by trying all alternatives, it declares that the entailment holds. It is not easy to use such procedures to generate an *explanation* for the entailment, or even to determine which axioms are responsible for the entailment, because the axioms that were used to construct the models are not necessarily the ones that are causing the clash. Recently another kind of reasoning procedures, which work by deriving logical consequences of ontology axioms directly, became popular. Such *consequence-based* procedures were first introduced for the $\mathcal{EL}$ family of tractable ontology languages [1], and later the same principle has been extended to more expressive (non-tractable) languages such as Horn-$\mathcal{SHIQ}$ and $\mathcal{ALCH}$ [9, 18]. The consequence-based procedures work by computing the closure under the rules by forward chaining. The inference rules make sure that the result is *complete*—all entailed conclusions of interest can be simply read off of the closure.

It is easy to extend any consequence-based procedure so that for each derived conclusion, it also records the inferences that have produced it. This way, one can easily generate proofs for the entailed conclusions. Unfortunately, saving all applied inferences during reasoning is not practical, as each conclusion could be derived in many ways and storing all inferences requires a lot of memory. In practice, one usually does not need to retrieve all inferences, but just inferences for some particular (e.g., unexpected) consequences. In this paper, we demonstrate how these inferences can be traced in a goal-directed way using the pre-computed set of conclusions. The main idea, is to split the conclusions on small *partitions*, so that most inferences are applied within each individual partition. It is then possible to re-compute the inferences for conclusions within each partition by forward chaining using conclusions from other partitions as set of support. A similar idea has been recently used for incremental reasoning in $\mathcal{EL}^+$ [10]. We demonstrate empirically that only a small fraction of inferences is produced when generating proofs for $\mathcal{EL}^+$ consequences and that the inferences can be computed in just a few milliseconds even for ontologies with hundreds thousands of axioms. Some omitted details, e.g., proofs, can be found in the technical report [11].

## 2 Related Work

In the context of ontologies, most research relevant to the issue of explanations has revolved around *justifications*—the minimal subsets of the ontology which entail the result [5, 8, 3, 7, 16]. For simple ontologies, justifications are often small and can be computed using generic axiom pinpointing procedures, e.g. based on model-based diagnosis [17], that use the decision procedure for the target (monotonic) logic as a black-box. While these procedures can significantly narrow down the set of relevant axioms, it is still up to the user to understand how to obtain the result from the justifications.

The fact that justifications are minimal subsets is useful to the user but also makes them intractable to compute [16]. Much research has gone into developing optimizations to cope with this complexity in practical cases. For example, employing ontology modularity techniques for black-box methods can be seen as a step towards goal-directed behavior [5]. The so-called *locality-based modules* [6] are well-defined fragments of the ontology which, first, can be computed fast using syntactic analysis and, second, contain all justifications for a given entailment. Modules bound the search for justification to an (often small) set of axioms. Using our goal-directed tracing procedure, one can limit the sets of relevant axioms even further: it is sufficient to take all axioms used in side conditions of the computed inferences to obtain all axioms that could be used in proofs. On the other hand, the glass-box methods (e.g., [3, 4]) tackle the problem by using the properties of the underlying reasoning procedure (instead of using it as a black-box). They are, however, not goal-directed and work similarly to what we below call 'full tracing'. Our method, in contrast, exploits specific properties of the inference system also to perform computations in a goal-directed way.

## 3 Preliminaries

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

The syntax of $\mathcal{EL}^+$ is defined using a vocabulary consisting of countably infinite sets of *(atomic) roles* and *atomic concepts*. $\mathcal{EL}^+$ 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 $C$, $C_1$, $C_2 \in \mathbf{C}$. An $\mathcal{EL}^+$ *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 role names. An $\mathcal{EL}^+$ *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 [11]. 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* (w.r.t. $\mathcal{O}$). The *ontology classification task* requires to compute all entailed subsumptions between atomic concepts occurring in $\mathcal{O}$.

### 3.2 Inferences, Inference Rules, and Proofs

Let $\mathbf{Exp}$ be a fixed countable set of *expressions*. An *inference* over $\mathbf{Exp}$ is an object *inf* which is assigned with a finite set of *premises* *inf*.Premises $\subseteq \mathbf{Exp}$ and a *conclusion* *inf*.conclusion $\in \mathbf{Exp}$.[1] When *inf*.Premises $= \emptyset$, we say that *inf* is an *initialization inference*. An *inference rule* $\mathsf{R}$ over $\mathbf{Exp}$ is a countable set of inferences over $\mathbf{Exp}$; it is an *initialization rule* if all these inferences are initialization inferences. In this paper, we view an *inference system* as one inference rule $\mathsf{R}$ representing all of their inferences.

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

We will often restrict inference rules to subsets of premises. Let *Exp* $\subseteq \mathbf{Exp}$ be a set of expressions, and $\mathsf{R}$ an inference rule. By $\mathsf{R}(\mathit{Exp})$ ($\mathsf{R}[\mathit{Exp}]$, $\mathsf{R}\langle\mathit{Exp}\rangle$) we denote the rule consisting of all inferences *inf* $\in \mathsf{R}$ such that *inf*.Premises $\subseteq$ *Exp* (respectively *Exp* $\subseteq$ *inf*.Premises, and *inf*.conclusion $\in$ *Exp*). We can arbitrarily combine these filters: for example, $\mathsf{R}[\mathit{Exp}_1](\mathit{Exp}_2)\langle\mathit{Exp}_3\rangle$ consists of those inferences in $\mathsf{R}$ whose premises contain all expressions from $\mathit{Exp}_1$, are a subset of $\mathit{Exp}_2$, and produce a conclusion in $\mathit{Exp}_3$. The order of filters is not relevant: $\mathsf{R}[\mathit{Exp}_1](\mathit{Exp}_2)\langle\mathit{Exp}_3\rangle$ is the same as, e.g, $\mathsf{R}(\mathit{Exp}_2)\langle\mathit{Exp}_3\rangle[\mathit{Exp}_1]$. For simplicity, we write $\mathsf{R}()$, $\mathsf{R}[]$, $\mathsf{R}(\mathit{exp})$, $\mathsf{R}[\mathit{exp}]$, and $\mathsf{R}\langle\mathit{exp}\rangle$ instead of $\mathsf{R}(\emptyset)$, $\mathsf{R}[\emptyset]$, $\mathsf{R}(\{\mathit{exp}\})$, $\mathsf{R}[\{\mathit{exp}\}]$, and $\mathsf{R}\langle\{\mathit{exp}\}\rangle$ respectively. Note that $\mathsf{R}[] = \mathsf{R}\langle\mathbf{Exp}\rangle = \mathsf{R}$ and $\mathsf{R}()$ consists of all initialization inferences in $\mathsf{R}$.

A *proof* (in $\mathsf{R}$) is a sequence of inferences $p = \mathit{inf}_1, \ldots, \mathit{inf}_n$ ($\mathit{inf}_i \in \mathsf{R}$, $1 \leq i \leq n$) such that $\mathit{inf}_j$.Premises $\subseteq \{\mathit{inf}_i.\text{conclusion} \mid 1 \leq i < j\}$ for each $j$ with $1 \leq j \leq n$. If $\mathit{exp} = \mathit{inf}_n$.conclusion then we say $p$ is a *proof for exp*. A proof $p = \mathit{inf}_1, \ldots, \mathit{inf}_n$ for *exp* is *minimal* if no strict sub-sequence of $\mathit{inf}_1, \ldots, \mathit{inf}_n$ is a proof for *exp*. Note that in this case $\mathit{inf}_i$.conclusion $\neq \mathit{inf}_j$.conclusion when $i \neq j$ ($1 \leq i, j \leq n$).

### 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 a variant of the rules that does not require normalization of the input ontology [13]. The rules for $\mathcal{EL}^+$ are given in Fig. 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 $= \{C \sqsubseteq D_1,\ C \sqsubseteq D_2\}$,

[1] There can be different inferences with the same sets of premises and the same conclusion.

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

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

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

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

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

*inf*.conclusion $= C \sqsubseteq D_1 \sqcap D_2$. The side conditions of rules $\mathsf{R}_\exists$ and $\mathsf{R}_\circ$ use the closure $\sqsubseteq^*_\mathcal{O}$ of role inclusion axioms—the smallest reflexive transitive relation such that $R \sqsubseteq S \in \mathcal{O}$ implies $R \sqsubseteq^*_\mathcal{O} S$. 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 Fig. 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 Fig. 1 [13]. 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. This procedure is polynomial since it derives only subsumptions of the form $C \sqsubseteq D$ and $C \sqsubseteq \exists R.D$ where $C$ and $D$ occur in $\mathcal{O}$. Example 1 illustrates the application of rules in Fig. 1 for deriving the entailed subsumption relations.

*Example 1.* Consider the $\mathcal{EL}^+$ ontology $\mathcal{O}$ consisting of the following axioms:
$A \sqsubseteq \exists R.B, \quad B \sqsubseteq \exists S.A, \quad \exists H.B \sqsubseteq C, \quad \exists S.C \sqsubseteq C, \quad R \sqsubseteq H.$

Then subsumption $B \sqsubseteq C$ has the following proof using the rules in Fig. 1. We show the premises used in the inferences in parentheses and side conditions after the colon:

$$A \sqsubseteq A \qquad \text{by } \mathsf{R_0}() : A \text{ occurs in } \mathcal{O}, \tag{1}$$
$$B \sqsubseteq B \qquad \text{by } \mathsf{R_0}() : B \text{ occurs in } \mathcal{O}, \tag{2}$$
$$A \sqsubseteq \exists R.B \qquad \text{by } \mathsf{R}_\sqsubseteq(A \sqsubseteq A) : A \sqsubseteq \exists R.B \in \mathcal{O}, \tag{3}$$
$$B \sqsubseteq \exists S.A \qquad \text{by } \mathsf{R}_\sqsubseteq(B \sqsubseteq B) : B \sqsubseteq \exists S.A \in \mathcal{O}, \tag{4}$$
$$A \sqsubseteq \exists H.B \qquad \text{by } \mathsf{R}_\exists(A \sqsubseteq \exists R.B,\ B \sqsubseteq B) : \exists R.B \text{ occurs in } \mathcal{O},\ R \sqsubseteq^*_\mathcal{O} H, \tag{5}$$
$$A \sqsubseteq C \qquad \text{by } \mathsf{R}_\sqsubseteq(A \sqsubseteq \exists H.B) : \exists H.B \sqsubseteq C \in \mathcal{O}, \tag{6}$$
$$B \sqsubseteq \exists S.C \qquad \text{by } \mathsf{R}_\exists(B \sqsubseteq \exists S.A,\ A \sqsubseteq C) : \exists S.C \text{ occurs in } \mathcal{O},\ R \sqsubseteq^*_\mathcal{O} R, \tag{7}$$
$$B \sqsubseteq C \qquad \text{by } \mathsf{R}_\sqsubseteq(B \sqsubseteq \exists S.C) : \exists S.C \sqsubseteq C \in \mathcal{O}. \tag{8}$$

The inferences producing the subsumptions (1)–(8) are shown graphically in Fig. 2.

## 3.4 Computing the Closure under Inference Rules

Computing the closure under inference rules, such as in Fig. 1, can be performed using a well-known *forward chaining procedure* detailed in Algorithm 1. The algorithm derives consequences by applying inferences in R and collects those conclusions between

$$\begin{tikzcd}
{} \arrow[r, mapsto, "\mathsf{R_0}"] & A \sqsubseteq A \arrow[r, mapsto, "\mathsf{R_\sqsubseteq}"] & A \sqsubseteq \exists R.B \arrow[r, mapsto, "\mathsf{R_\exists}"] & A \sqsubseteq \exists H.B \arrow[r, mapsto, "\mathsf{R_\sqsubseteq}"] & A \sqsubseteq C \arrow[dr, dash] & & \\
& & {} \arrow[r, mapsto, "\mathsf{R_0}"'] & B \sqsubseteq B \arrow[u, dash] \arrow[r, mapsto, "\mathsf{R_\sqsubseteq}"'] & B \sqsubseteq \exists S.A \arrow[r, mapsto, "\mathsf{R_\exists}"'] & B \sqsubseteq \exists S.C \arrow[r, mapsto, "\mathsf{R_\sqsubseteq}"'] & B \sqsubseteq C
\end{tikzcd}$$

**Fig. 2.** The inference diagram for the proof in Example 1

**Algorithm 1:** Computing the inference closure by saturation

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

which all inferences are applied in a set S (the *saturation*) and the other conclusions in a queue Q. The algorithm first initializes Q with conclusions of the initialization inferences R() $\subseteq$ R (lines 2–3), and then in a cycle (lines 4–9), repeatedly takes the next expression *exp* $\in$ Q, if any, inserts it into S if it does not occur there, and applies all inferences *inf* $\in$ R[*exp*](S) having this expression as one of the premises and other premises from S. The conclusions of such inferences are then inserted back into Q. Note that every inference in R(S) is applied by the algorithm exactly once: this happens in line 8 only when *exp* is the last premise of the inference added to S.

## 4 Tracing of Inferences

In this section we describe algorithms for computing the inferences in R deriving given expressions in the closure S under R. Depending on a particular application, one might be interested in computing all inferences for every given expression or just one inference per expression. One might also be interested in unfolding such inferences recursively, that is, computing for each inference also (some or all) inferences deriving its premises. This can be useful for generation of full proofs or for interactive debugging.

Our algorithms are based on 'tracing of inferences', that is, we record the relevant inferences during the forward chaining application of the rules. This has an advantage that one can reuse the existing forward chaining infrastructure without requiring additional indexes, e.g., to identify matching premises of the rules by conclusions.

**Algorithm 2:** Goal-directed tracing of inferences

```
goalDirectedTracing($\mathsf{R}, \mathsf{S}, \mathit{Exp}$):
input  : Rules: a set of inferences, $\mathsf{S}$, $\mathit{Exp} \subseteq \mathsf{S} \cap \mathrm{Closure}(\mathsf{R}(\mathsf{S}))$: sets of expressions
output : $\mathsf{M}$: a multimap from expressions to inferences such that $\mathsf{M}.\mathrm{Keys} = \mathit{Exp}$ and for
         each $\mathit{exp} \in \mathit{Exp}$ we have that $\mathsf{M}(\mathit{exp}) = \mathsf{R}(\mathsf{S})\langle \mathit{exp} \rangle$

1  $\mathsf{M}, \mathsf{Q} \leftarrow \emptyset$;
2  for $\mathit{inf} \in \mathsf{R}(\mathsf{S} \setminus \mathit{Exp})\langle \mathit{Exp} \rangle$ do                /* initialize */
3      $\mathsf{Q}$.add($\mathit{inf}$);
4  while $\mathsf{Q} \neq \emptyset$ do                                      /* close */
5      $\mathit{inf} \leftarrow \mathsf{Q}$.takeNext();
6      $\mathit{exp} \leftarrow \mathit{inf}$.conclusion;
7      if $\mathit{exp} \notin \mathsf{M}.\mathrm{Keys}$ then
8          $\mathsf{M}$.add($\mathit{exp} \mapsto \mathit{inf}$);
9          for $\mathit{inf} \in \mathsf{R}[\mathit{exp}](\mathsf{M}.\mathrm{Keys} \cup (\mathsf{S} \setminus \mathit{Exp}))\langle \mathit{Exp} \rangle$ do
10             $\mathsf{Q}$.add($\mathit{inf}$);
11     else
12         $\mathsf{M}$.add($\mathit{exp} \mapsto \mathit{inf}$);
13 return $\mathsf{M}$;
```

### 4.1 Full Tracing

It is relatively straightforward to adapt Algorithm 1 so that all inferences applied in the computation are saved. (cf. [11] for a formal presentation). Instead of collecting the expressions derived by the inferences in $\mathsf{S}$, we collect the inferences themselves and store them in a multimap $\mathsf{M}$. A multimap is like a map, except that it can store several values (in our case inferences) for the same key (in our case the conclusion of the inferences). The newly found inferences are also processed through the queue $\mathsf{Q}$. If the conclusion of the inference was not derived before by any other inference, it is added to $\mathsf{M}$ and all inferences between its conclusion and the previously derived conclusions are produced. Otherwise, we just store the new inference and do nothing else. The algorithm can be easily adapted to store only one traced inference per conclusion: for that it suffices to make $\mathsf{M}$ an ordinary map. As Algorithm 1, closure computation with full tracing generates every inference at most once and thus retains polynomial complexity.

### 4.2 Goal-directed Tracing of Inferences

It might be not practical to store all inferences used in the computation of the closure because this can require much more memory compared to just storing the conclusions of inferences. In practice, one usually does not require all inferences, but just inferences for some conclusions of interest (e.g., resulting from modeling errors). While it is possible to modify full tracing to record only inferences of interest, it is also possible to avoid applying many unnecessary inferences if the closure $\mathsf{S}$ under $\mathsf{R}$ is already computed.

A goal-directed method for computing $\mathsf{R}(\mathsf{S})\langle \mathit{Exp} \rangle$, that is, the inferences in $\mathsf{R}(\mathsf{S})$ that derive expressions from the given set *Exp*, is detailed in Algorithm 2. The algorithm first generates only those inferences in $\mathsf{R}(\mathsf{S})$ that do not use any premises from *Exp* (lines 2–3), and then repeatedly generates inferences by applying the rules to their conclusions that are in *Exp* similarly to full tracing. Specifically, the rules are only applied to conclusions of the inferences saved so far (stored in $\mathsf{M}$) and the expressions in $\mathsf{S}$ that are not in *Exp* (see line 9). This is needed to ensure that every inference is generated at most once: an inference *inf* is generated only when its last premise from *Exp* is computed. Note that in Algorithm 2 we do not require $\mathsf{S}$ to be the closure under $\mathsf{R}$. We just require that *Exp* is a subset of $\mathsf{S}$ and $\mathrm{Closure}(\mathsf{R}(\mathsf{S}))$, which means that every expression $exp \in Exp$ must be derivable using only inferences from $\mathsf{R}$ restricted to premises in $\mathsf{S}$. The following theorem asserts correctness of the algorithm.

**Theorem 1 (Correctness of Algorithm 2).** *Let* $\mathsf{R}$ *be a set of inferences and* $\mathsf{S}$, $Exp \subseteq \mathsf{S} \cap \mathrm{Closure}(\mathsf{R}(\mathsf{S}))$ *be sets of expressions, and* $\mathsf{M}$ *be the output of Algorithm 2 on* $\mathsf{R}, \mathsf{S}$, *and Exp. Then* $\mathsf{M}.\mathrm{Keys} = Exp$ *and for each* $exp \in Exp$ *we have* $\mathsf{M}(exp) = \mathsf{R}(\mathsf{S})\langle exp\rangle$.

### 4.3 Tracing of Inferences using Partitions

The reader may wonder, why we could not simply iterate over $inf \in \mathsf{R}(\mathsf{S})\langle Exp\rangle$ directly, especially if we could iterate over $inf \in \mathsf{R}(\mathsf{S} \setminus Exp)\langle Exp\rangle$ in line 2 of Algorithm 2? In general, iteration over $\mathsf{R}(\mathsf{S}')\langle Exp\rangle$ for $\mathsf{S}' \subseteq \mathsf{S}$ can be performed by iterating over all inferences $inf \in \mathsf{R}(\mathsf{S}')$ using a forward chaining procedure similar to full tracing and checking if $inf.\mathrm{conclusion} \in Exp$. In this case, of course, Algorithm 2 does not give any advantage over full tracing since both algorithms enumerate all inferences in $\mathsf{R}(\mathsf{S})$. Algorithm 2, however, gives advantage if $\mathsf{R}(\mathsf{S} \setminus Exp)\langle Exp\rangle$ is small and can be computed more effectively. For example, in Sect. 5.1, we demonstrate that one can select some non-trivial subsets *Exp* such that $\mathsf{R}(\mathsf{S} \setminus Exp)\langle Exp\rangle = \mathsf{R}()\langle Exp\rangle$, that is, only initialization inferences can produce expressions in *Exp* from expressions not in *Exp*. If the set of all expressions $\mathbf{Exp}$ can be partitioned on such subsets, it is possible to develop a more efficient inference tracing algorithm.

Specifically, let $\mathbf{Pts}$ be a fixed countable set of *partition identifiers* (short *partitions*), and assume that each expression $exp \in \mathbf{Exp}$ is assigned with exactly one partition $exp.\mathrm{partition} \in \mathbf{Pts}$. Further, let $Exp[pt] = \{exp \in Exp \mid exp.\mathrm{partition} = pt\}$. Then one can use Algorithm 3 to compute $\mathsf{R}(\mathsf{S})\langle Exp\rangle$ for arbitrary $Exp \subseteq \mathbf{Exp}$. Essentially, Algorithm 3 performs tracing of inferences using Algorithm 2 for the partitions assigned to expressions in *Exp*. This way, one can still enumerate many inferences that do not derive expressions from *Exp*, but if the partitions are small enough, this overhead is not significant, and the method is still more practical than using full tracing.

Algorithm 3 can be easily modified to trace just one inference per expression and/or to unfold the inferences recursively. For the former, it is sufficient to use the appropriate modification of Algorithm 2 as discussed in Sect. 4.2. For the latter, it is sufficient to repeatedly add the premises for each computed inference with the conclusion in *Exp* as additional inputs of the algorithm until a fixpoint is reached.

## 5 Tracing of Inferences in $\mathcal{EL}^+$

In this section we apply Algorithm 3 for goal-directed tracing of inferences in $\mathcal{EL}^+$ ontologies. To this end, we assume that the closure $\mathsf{S}$ under the rules in Fig. 1 is computed,

**Algorithm 3:** Goal-directed tracing of inferences using partitions

```
**goalDirectedPartitionTracing**(R, S, *Exp*):
input  : R: a set of inferences, $\mathsf{S} \subseteq \mathrm{Closure}(\mathsf{R}(\mathsf{S}))$, $Exp \subseteq \mathsf{S}$: sets of expressions
output : M: $\mathbf{Pts} \to \mathbf{Exp} \to 2^{\mathsf{R}(\mathsf{S})}$ a two level multimap such that for each $exp \in Exp$, we
         have $\mathsf{M}(exp.\mathrm{partition})(exp) = \mathsf{R}(\mathsf{S})\langle exp\rangle$
1 $\mathsf{M} \leftarrow \emptyset$;
2 for $exp \in Exp$ do
3     $pt \leftarrow exp$.partition;
4     if $pt \notin \mathsf{M}$.Keys then
5         $\mathsf{M}(pt) \leftarrow$ goalDirectedTracing(R, S, S[$pt$]) ;        /* by Algorithm 2 */
6 return M;
```

$$\begin{tikzcd}
{} \arrow[r, dashed, "\mathbf{R_0}"] & A \sqsubseteq A \arrow[r, dashed, "\mathbf{R_\sqsubseteq}"] & A \sqsubseteq \exists R.B \arrow[r, dashed, "\mathbf{R_\exists}"] & A \sqsubseteq \exists H.B \arrow[r, dashed, "\mathbf{R_\sqsubseteq}"] & A \sqsubseteq C \arrow[d] & & \\
& & {} \arrow[r, "\mathbf{R_0}"'] & B \sqsubseteq B \arrow[u, dashed] \arrow[r, "\mathbf{R_\sqsubseteq}"'] & B \sqsubseteq \exists S.A \arrow[r, "\mathbf{R_\exists}"'] & B \sqsubseteq \exists S.C \arrow[r, "\mathbf{R_\sqsubseteq}"'] & B \sqsubseteq C
\end{tikzcd}$$

**Fig. 3.** The inferences applied for tracing partition $B$ (solid lines) and for tracing partition $A$ (dashed lines) using Algorithm 2 for the closure S containing subsumptions (1)–(8).

and we are given a set of subsumptions $Exp \subseteq \mathsf{S}$, for which the inferences that have derived them in S should be found. We first describe our strategy of partitioning the derived subsumptions, then discuss some issues related to optimizations, and, finally, present an empirical evaluation of tracing performance on existing ontologies.

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

To partition the set **Exp** of concept subsumptions $C \sqsubseteq D$ derived by the rules R in Fig. 1, we use the partitioning function used for incremental reasoning in $\mathcal{EL}^+$ [10]. Specifically, the set of partition identifiers **Pts** is the set of all $\mathcal{EL}^+$ concepts, and every subsumption $exp = C \sqsubseteq D$ is assigned with the partition $exp.\mathrm{partition} = C \in \mathbf{Pts}$. It is easy to see that each conclusion of a rule in Fig. 1 has the same left-hand side as one of the premises of the rule, unless it is rule $\mathbf{R_0}$ or $\mathbf{R_\top}$. Thus, for the set $Exp = \mathsf{S}[C]$ of subsumptions $C \sqsubseteq D \in \mathsf{S}$ that belong to partition $C$, we have $\mathsf{R}(Exp)\langle Exp\rangle = \mathsf{R}()\langle Exp\rangle$. Therefore, tracing of inferences for expressions in $\mathsf{S}[C]$ can be performed efficiently using Algorithm 2. Note that, according to Algorithm 3, to find all inferences for $exp = C \sqsubseteq D$, it is sufficient to trace just the inferences for partition $C$.

*Example 2.* Consider subsumptions (1)–(8) derived in Example 1. These subsumptions are assigned to two different partitions $A$ and $B$. To compute the inferences for $B \sqsubseteq C$, it is sufficient to apply Algorithm 2 for partition $B$ ($Exp = \mathbf{Exp}[B]$) using the precomputed closure S containing (1)–(8). It is easy to see that only inferences for (2), (4), (7), and (8) will be produced by this algorithm (compare Fig. 3 with Fig. 2). During initialization (line 2), Algorithm 2 applies only rule $\mathbf{R_0}$ deriving $B \sqsubseteq B$. During closure (line 4), the algorithm applies only inferences to the derived subsumptions in partition $B$, and keeps only those inferences that produce subsumptions in the same partition. For example, the inference by $\mathsf{R}_\exists$ producing (5) should be ignored, even though it uses a premise $B \sqsubseteq B$ from partition $B$. Note that these inferences would not be computed without $\mathsf{S}$: to produce $B \sqsubseteq \exists S.C$ we use the premise $A \sqsubseteq C$ from $\mathsf{S}$. But we do not need to compute the inferences for $A \sqsubseteq C$ first. Thus, the precomputed set $\mathsf{S}$ is used in our procedure as a set of support to reach the conclusions of interest as fast as possible.

Now, if we want to unfold the inferences backwards to obtain the full proof for $B \sqsubseteq C$, we need to iterate over the premises that were used to derive $B \sqsubseteq C$ and trace their partitions. In our case, $B \sqsubseteq C$ was derived from $B \sqsubseteq \exists S.C$, for which the partition $B$ was already traced, but continuing further to the premise $A \sqsubseteq C$ will bring us to the partition $A$. When tracing partition $A$, we produce the remaining inferences for (1), (3), (5), and (6) used in the proof (see Fig. 3). Note that it is not necessary to trace partition $A$ to find the proof, e.g., for $B \sqsubseteq \exists S.A$ because no expression in partition $A$ was used to derive $B \sqsubseteq \exists S.A$.

## 5.2 Optimizations

In this section we describe how our tracing algorithms can co-exist with known optimizations for $\mathcal{EL}$ saturation and present one specific optimization for Algorithm 2 to avoid tracing of unnecessary partitions.

**Forward redundancy:** Existing implementations of the $\mathcal{EL}^+$ procedure based on the rules in Fig. 1, employ a number of additional optimizations to reduce the number of operations. In particular, reasoner ELK avoids applications of some redundant inferences—inferences that are not necessary for obtaining the required consequences. For example, it is not necessary to apply rule $\mathsf{R}_\sqcap^-$ to conclusions of rule $\mathsf{R}_\sqcap^+$, and it is not necessary to apply rules $\mathsf{R}_\exists$ and $\mathsf{R}_\circ$ if their first premise was derived by rule $\mathsf{R}_\exists$ [13]. Since the rules are applied by Algorithm 1 only when the conclusion is produced for the first time, the result of the saturation can be different if the rules are applied in a different order. Consequently, if we apply only non-redundant inferences during tracing, due to a potentially different order of rule applications (which may well happen when tracing only partitions), we may miss some of the inferences that were originally applied, and may even fail to derive the expressions to be traced. So, to ensure that all original inferences are traced, we also have to apply the redundant inferences (but if the conclusion of the inference is not in the closure, we do not need to keep it).

**Cyclic inferences:** As mentioned in Sect. 4.3, Algorithm 3 for tracing inferences using partitions, can be extended to unfold the proofs recursively by repeating the tracing procedure for the premises of relevant inferences. In practice, one is usually interested in minimal proofs, i.e., proofs without repetitions. However, not every inference that is obtained by our procedure can be used in a minimal proof. If an inference is *cyclic*, that is, some premise of the inference could only be derived from its conclusion, it can be always removed from the proof. Disregarding cyclic inferences can result in fewer traced partitions when unfolding inferences recursively. Note that if we only save the first inference per expression in Algorithm 2, we never create cyclic inferences.

We can avoid many, but not all, cyclic inferences by using a modification of Algorithm 2 in which we save an inference *inf* in $\mathsf{M}$ only if every premise of *inf* is derived by

**Table 1.** Number of axioms, conclusions, and inferences in test ontologies, as well running time (in ms.) and memory consumption (in MB) with and without full tracing during classification.

| Ontology / Number of axioms | (partitions) | Redun. elim. | Number of conclusions | Number of inferences | No tracing time | No tracing mem. | Full tracing time | Full tracing mem. |
|---|---|---|---|---|---|---|---|---|
| GO | 87,492 | on | 2,183,400 | 3,614,034 | 1,309 | 643 | 2,698 | 1,066 |
| (extended) | (46,846) | off | 2,450,505 | 5,715,611 | 1,877 | 546 | 5,004 | 1,102 |
| GALEN | 36,547 | on | 1,479,450 | 2,015,877 | 935 | 690 | 1,830 | 711 |
| ($\mathcal{EL}^+$ version) | (25,965) | off | 1,922,549 | 3,944,921 | 1,326 | 722 | 3,786 | 765 |
| SNOMED CT | 297,327 | on | 16,673,337 | 24,338,895 | 12,738 | 900 | 51,603 | 3,010 |
| (July 2013) | (371,642) | off | 19,963,726 | 54,553,961 | 16,968 | 924 | OOM | OOM |

some inference that does not use the conclusion of *inf*, i.e., we look only at short cycles. More details and empirical evaluation can be found in the technical report [11].

## 5.3 Experimental Evaluation

We have implemented Algorithm 3 and optimizations described in Sect.s 5.1 and 5.2 in the $\mathcal{EL}^+$ reasoner ELK[2] and evaluated their performance. We used three large OWL EL ontologies which are often used in evaluations of $\mathcal{EL}$ reasoners [2, 12, 14]: a version of the Gene Ontology (GO),[3] an $\mathcal{EL}^+$-restricted version of the GALEN ontology,[4] and the July 2013 release of SNOMED CT.[5] Classification for each experiment was performed with the redundancy optimizations enabled and disabled (the latter eliminates non-determinism and computes all conclusions used in proofs). We used a PC with Intel Core i5-2520M 2.50GHz CPU, with Java 1.6 and 4GB RAM available to JVM.

**Full tracing overhead:** Our first experiment evaluates the overhead of full tracing (cf. Sect. 4.1 and [11] for more details) comparing to pure saturation (Algorithm 1). Each ontology was classified 10 times with and without full tracing and the results were averaged (excluding 5 warm-up runs). The results in Table 1 show that there is roughly x2–x4 time overhead as well x3 memory overhead on SNOMED CT when the redundancy elimination is enabled.[6] When redundant inferences are applied, the overhead is bigger such that full tracing of SNOMED CT runs out of memory (OOM).

**Goal-directed tracing:** Next we evaluate performance of goal-directed tracing (Algorithm 3). The experimental setup is as follows: each ontology was classified and then each direct subsumption between concept names was traced with recursive unfolding. Each subsumption was traced independently of others. We separately report results for tracing all of inferences for each conclusion or just of the first inference. The former is useful for generating all proofs whereas the latter can be used to produce one proof.

[2] `http://elk.semanticweb.org`

[3] It is a richer version with concept equivalences, role hierarchies and chains. We thank Chris Mungall from the Lawrence Berkley Lab for providing it. It is accessible at: `http://elk.semanticweb.org/ontologies/go_ext.owl`.

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

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

[6] With the smaller ontologies the difference is less observable because of the relatively high (4GB) memory limit set to JVM; the difference is better observable with lower memory limits.

**Table 2.** Evaluation results for goal-directed tracing of all atomic subsumptions in the ontology when all (resp. the first) inferences for each conclusion are unfolded.

| Ontology (Num. subsumptions) | Redun. elim. | # of traced partitions | # of traced inferences | # of inferences used in proofs | # of used $\sqsubseteq$ axioms | Time (in ms.) |
|---|---|---|---|---|---|---|
| GO | on | 2.5 (1.4) | 241.3 (149.0) | 52.1 (6.4) | 17.7 (2.5) | 0.9 (0.6) |
| (73,270) | off | 3.7 (1.4) | 456.2 (244.1) | 94.9 (6.4) | 18.6 (2.5) | 2.0 (1.2) |
| GALEN | on | 1.6 (1.5) | 210.9 (188.9) | 12.2 (9.9) | 4.9 (4.4) | 0.8 (0.7) |
| (38,527) | off | 1.9 (1.5) | 414.4 (350.9) | 21.7 (10.1) | 5.2 (4.4) | 1.9 (0.8) |
| SNOMED CT | on | 2.9 (1.6) | 187.8 (136.6) | 49.6 (12.7) | 8.9 (3.7) | 0.7 (0.5) |
| (443,402) | off | 8.6 (1.6) | 788.3 (332.9) | 385.9 (12.7) | 9.7 (3.7) | 10.1 (1.9) |

The results are presented in Table 2. First, they demonstrate that the proposed method is very fast as it usually takes around a millisecond to trace a subsumption. (the only exception is SNOMED CT without redundancy elimination, which is likely due to cyclic inferences not detected by the optimization from Sect. 5.2). This is the effect of its goal-directed nature: it only traces inferences which belong to the same partition as an inference used by some proof of the target subsumption. The performance results thus follow from the low number of traced partitions and traced inferences. Second, the algorithm traces more inferences when all proofs are needed but the difference is not substantial and the running times are usually close. This is because alternative proofs overlap and the algorithm rarely needs to trace new partitions when unfolding more than one inference per conclusion. Third, the difference between the number of traced and used inferences shows granularity of partitioning in $\mathcal{EL}^+$ (inferences not used in proofs can be traced only if they happen to be in one partition with used inferences). Finally, since the results are averaged over all subsumptions, the reader may wonder if the good performance is because most of them can be proved trivially. Results in the technical report [11], which are separately aggregated over subsumptions that are provable from at least 10 axioms, show that performance stays on the same level.

## 6 Summary and Future Research

In this paper we have presented a simple, efficient, and generalizable method for goal-directed tracing of inferences in $\mathcal{EL}^+$. Depending on the application, the inferences can be used to recover proofs or compute justifications. The method is goal-directed in the sense that it re-applies only a limited number of inferences using the previously computed conclusions as a set of support. It does not require storing additional indexing information or any sort of bookkeeping during the normal classification. The method is based on the same *granularity property* of reasoning in $\mathcal{EL}^+$ as was previously used for concurrent and incremental reasoning [12, 10]. Specifically, concept subsumer in $\mathcal{EL}^+$ most of the time can be computed independently of each other. This enables efficient partitioning of all derived expressions so that tracing a particular expression requires re-applying inferences only in few partitions, as shown empirically. The same property can be further exploited for other tasks, for example, distributed reasoning in $\mathcal{EL}^+$.

## References

1. Baader, F., Brandt, S., Lutz, C.: Pushing the $\mathcal{EL}$ envelope. In: Proc. 19th Int. Joint Conf. on Artificial Intelligence (IJCAI'05). pp. 364–369 (2005)
2. Baader, F., Lutz, C., Suntisrivaraporn, B.: Efficient reasoning in $\mathcal{EL}^+$. In: Proc. 2006 Int. Workshop on Description Logics (DL'06). vol. 189. CEUR-WS.org (2006)
3. Baader, F., Peñaloza, R.: Automata-based axiom pinpointing. J. of Automated Reasoning 45(2), 91–129 (2010)
4. Baader, F., Peñaloza, R., Suntisrivaraporn, B.: Pinpointing in the description logic $\mathcal{EL}$. In: Proc. 2007 Int. Workshop on Description Logics (DL'07). vol. 250. CEUR-WS.org (2007)
5. Baader, F., Suntisrivaraporn, B.: Debugging SNOMED CT using axiom pinpointing in the description logic $\mathcal{EL}^+$. In: KR-MED. vol. 410. CEUR-WS.org (2008)
6. Cuenca Grau, B., Horrocks, I., Kazakov, Y., Sattler, U.: Modular reuse of ontologies: Theory and practice. J. of Artificial Intelligence Research 31, 273–318 (2008)
7. Horridge, M., Parsia, B., Sattler, U.: Laconic and precise justifications in OWL. In: Proc. 7th Int. Semantic Web Conf. (ISWC'08). LNCS, vol. 5318, pp. 323–338. Springer (2008)
8. Kalyanpur, A., Parsia, B., Horridge, M., Sirin, E.: Finding all justifications of OWL DL entailments. In: Proc. 6th Int. Semantic Web Conf. (ISWC'08). LNCS, vol. 4825, pp. 267–280. Springer (2007)
9. Kazakov, Y.: Consequence-driven reasoning for Horn $\mathcal{SHIQ}$ ontologies. In: Proc. 21st Int. Joint Conf. on Artificial Intelligence (IJCAI'09). pp. 2040–2045. IJCAI (2009)
10. Kazakov, Y., Klinov, P.: Incremental reasoning in OWL EL without bookkeeping. In: Proc. 12th Int. Semantic Web Conf. (ISWC'13). LNCS, vol. 8218, pp. 232–247. Springer (2013)
11. Kazakov, Y., Klinov, P.: Goal-directed tracing of inferences in $\mathcal{EL}$ ontologies. Tech. rep., University of Ulm (2014), available from `http://http://elk.semanticweb.org/publications/elk-tracing-trdl-2014.pdf`
12. Kazakov, Y., Krötzsch, M., Simančík, F.: Concurrent classification of $\mathcal{EL}$ ontologies. In: Proc. 10th Int. Semantic Web Conf. (ISWC'11). LNCS, vol. 7032, pp. 305–320. Springer (2011)
13. Kazakov, Y., Krötzsch, M., Simančík, F.: The incredible ELK: From polynomial procedures to efficient reasoning with $\mathcal{EL}$ ontologies. J. of Automated Reasoning (2013), to appear
14. Lawley, M.J., Bousquet, C.: Fast classification in Protégé: Snorocket as an OWL 2 EL reasoner. In: Proc. 6th Australasian Ontology Workshop (IAOA'10). vol. 122, pp. 45–49 (2010)
15. Motik, B., Shearer, R., Horrocks, I.: Hypertableau reasoning for description logics. J. of Artificial Intelligence Research 36, 165–228 (2009)
16. Peñaloza, R., Sertkaya, B.: On the complexity of axiom pinpointing in the $\mathcal{EL}$ family of description logics. In: Proc. 12th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR'10). pp. 280–289. AAAI Press (2010)
17. Reiter, R.: A theory of diagnosis from first principles. Artificial Intelligence 32(1), 57–95 (1987)
18. Simančík, F., Kazakov, Y., Horrocks, I.: Consequence-based reasoning beyond Horn ontologies. In: Proc. 22nd Int. Joint Conf. on Artificial Intelligence (IJCAI'11). pp. 1093–1098. AAAI Press/IJCAI (2011)
19. Sirin, E., Parsia, B., Grau, B.C., Kalyanpur, A., Katz, Y.: Pellet: A practical OWL-DL reasoner. J. of Web Semantics 5(2), 51–53 (2007)
20. Tsarkov, D., Horrocks, I.: FaCT++ description logic reasoner: System description. In: Proc. 3rd Int. Joint Conf. on Automated Reasoning (IJCAR'06). LNCS, vol. 4130, pp. 292–297. Springer (2006)