# 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 record all inference steps during the application of rules, 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 potentially applied to many 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. 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—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 $\mathcal{ALCH}$ [10, 19]. 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 are obtained in 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

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

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

back in a goal-directed way using the pre-computed set of conclusions. The main idea, is to split the conclusions on small *partitions* using the properties of the inference system, such 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 $\mathcal{EL}^+$ [11]. 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.

## 2 Tutorial

In this section, we introduce the problem addressed in the paper and present the main ideas of our solution. For simplicity, we focus on reasoning in $\mathcal{EL}^+$ [2]. Later, in Section 3, we generalize the method to arbitrary (deterministic) inference systems.

### 2.1 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*. Complex *concepts* and *axioms* are defined recursively using Table 1. We use 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* $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ consists of a nonempty set $\Delta^{\mathcal{I}}$ called the *domain* of $\mathcal{I}$ and an interpretation function $\cdot^{\mathcal{I}}$ that assigns to each role $R$ a binary relation $R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$, and to each atomic concept $A$ a set $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$. This assignment is extended to complex concepts as shown in Table 1. $\mathcal{I}$ *satisfies* an axiom $\alpha$ (written $\mathcal{I} \models \alpha$) if the corresponding condition in Table 1 holds. $\mathcal{I}$ is a *model* of an ontology $\mathcal{O}$ (written $\mathcal{I} \models \mathcal{O}$) if $\mathcal{I}$ satisfies all axioms in $\mathcal{O}$. We say that $\mathcal{O}$ *entails* an axiom $\alpha$

$$\mathbf{R_0}\ \frac{}{C \sqsubseteq C} \qquad \mathbf{R}^+_\sqcap\ \frac{C \sqsubseteq D_1 \quad C \sqsubseteq D_2}{C \sqsubseteq D_1 \sqcap D_2} : D_1 \sqcap D_2 \in G(C)$$

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

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

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

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

(written $\mathcal{O} \models \alpha$), if every model of $\mathcal{O}$ satisfies $\alpha$. A concept $C$ is *subsumed* by $D$ w.r.t. $\mathcal{O}$ if $\mathcal{O} \models C \sqsubseteq D$. The *ontology classification task* requires to compute all entailed subsumptions between atomic concepts occurring in $\mathcal{O}$.

### 2.2 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 [14]. 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 inference rules have side conditions given after the colon that restrict the expressions to which the rules are applicable. The side conditions are formulated using the given ontology $\mathcal{O}$ and a mapping that assigns to every concept $X$ the set of *goal subsumers* $G(X)$ consisting of concepts $Y$ (not necessarily occurring in $\mathcal{O}$), subsumptions which should be checked by the procedure. That is, if we want the procedure to check whether the subsumption $X \sqsubseteq Y$ is entailed, we need to add $Y$ to $G(X)$. Intuitively, the mapping $G(X)$ restricts the concepts that should be constructed by the inference rules $\mathbf{R}_\top$, $\mathbf{R}^+_\sqcap$ and $\mathbf{R}_\exists$. For technical reasons, we need to require that each $G(X)$ also contains all concepts occurring in the left-hand sides of concept inclusion axioms of $\mathcal{O}$ (possibly as a sub-concept). Note that the axioms in the ontology $\mathcal{O}$ are only used in side conditions of the rules $\mathbf{R}_\sqsubseteq$, $\mathbf{R}_\exists$, and $\mathbf{R}_\circ$, and never as premises of the rules.

The rules in Figure 1 are complete for deriving all entailed goal subsumptions. That is, if $\mathcal{O} \models X \sqsubseteq Y$ and $Y \in G(X)$ (with $X$ and $D$ not necessarily occurring in $\mathcal{O}$) then $X \sqsubseteq Y$ can be derived using the rules in Figure 1 [14]. The rules can be also applied in a goal-directed way, if the set of concepts $X$ for which subsumptions $X \sqsubseteq Y$ should be derived (with $Y \in G(X)$) is also known in advance.

**Theorem 1 (Completeness of the rules in Figure 1 [14]).** *Let $\mathcal{O}$ be an $\mathcal{EL}^+$ ontology, $F$ a set of concepts, $G(\cdot)$ a mapping such that for each $X \in F$, and each $C \sqsubseteq D \in \mathcal{O}$, $G(X)$ contains all sub-concepts of $C$, and $S$ a set of subsumptions such that:*

*(i) If $X \in F$ and $X \sqsubseteq Y$ is obtained by a rule in Figure 1 applied to premises in $S$ using $\mathcal{O}$ and $G(\cdot)$ then $X \sqsubseteq Y \in S$,*
*(ii) If $X \in F$ and $X \sqsubseteq \exists R.Y \in S$ for some $R$ and $Y$, then $Y \in F$.*

*Then for every $X \in F$ and $Y \in G(X)$, we have $\mathcal{O} \models X \sqsubseteq Y$ iff $X \sqsubseteq Y \in S$.*

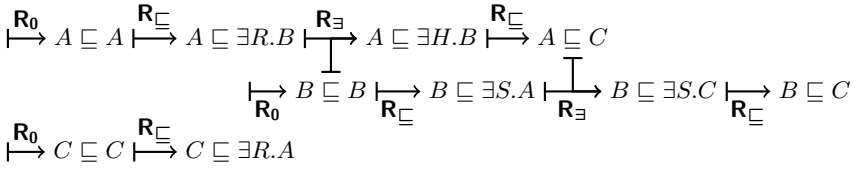


**Fig. 2.** The inference diagram for the proof in Example 1

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

Then the following subsumptions can be derived using the rules in Figure 1 using $G(X) = \{A,\, B,\, C,\, \exists H.B,\, \exists S.C\}$ for every concept $X$. We show the premises used in the inferences in parentheses and the matching side conditions after the colon:

| | | |
|---|---|---|
| $A \sqsubseteq A$ | by $\mathbf{R_0}()$, | (1) |
| $B \sqsubseteq B$ | by $\mathbf{R_0}()$, | (2) |
| $C \sqsubseteq C$ | by $\mathbf{R_0}()$, | (3) |
| $A \sqsubseteq \exists R.B$ | by $\mathbf{R_\sqsubseteq}(A \sqsubseteq A)$ : $A \sqsubseteq \exists R.B \in \mathcal{O}$, | (4) |
| $B \sqsubseteq \exists S.A$ | by $\mathbf{R_\sqsubseteq}(B \sqsubseteq B)$ : $B \sqsubseteq \exists S.A \in \mathcal{O}$, | (5) |
| $C \sqsubseteq \exists R.A$ | by $\mathbf{R_\sqsubseteq}(C \sqsubseteq C)$ : $C \sqsubseteq \exists R.A \in \mathcal{O}$, | (6) |
| $A \sqsubseteq \exists H.B$ | by $\mathbf{R_\exists}(A \sqsubseteq \exists R.B, B \sqsubseteq B)$ : $\exists H.B \in G(A)$, $R \sqsubseteq^*_\mathcal{O} H$, | (7) |
| $A \sqsubseteq C$ | by $\mathbf{R_\sqsubseteq}(A \sqsubseteq \exists H.B)$ : $\exists H.B \sqsubseteq C \in \mathcal{O}$, | (8) |
| $B \sqsubseteq \exists S.C$ | by $\mathbf{R_\exists}(B \sqsubseteq \exists S.A, A \sqsubseteq C)$ : $\exists S.C \in G(B)$, $R \sqsubseteq^*_\mathcal{O} R$, | (9) |
| $B \sqsubseteq C$ | by $\mathbf{R_\sqsubseteq}(B \sqsubseteq \exists S.C)$ : $\exists S.C \sqsubseteq C \in \mathcal{O}$. | (10) |

The inferences (1)–(10) are shown schematically in Figure 2. Let $F = \{A,\, B,\, C\}$. It is easy to see that no new subsumption of the form $X \sqsubseteq Y$ with $X \in F$ and $Y \in G(X)$ can be derived from (1)–(10) using the rules in Figure 1. Furthermore, $F$ satisfies the condition $(ii)$ of Theorem 1 for the set of subsumptions (1)–(10). Hence, by Theorem 1, all subsumptions of the form $X \sqsubseteq Y$ with $X \in F$ and $Y \in G(X)$ that are entailed by $\mathcal{O}$ must occur in (1)–(10). In particular, all entailed subsumptions between atomic concepts $A$, $B$, and $C$ that are required for classification of $\mathcal{O}$ are computed.

By Theorem 1, in order to classify an ontology $\mathcal{O}$, it is sufficient to $(i)$ initialize set $F$ with all atomic concepts of $\mathcal{O}$, $(ii)$ assign to every $X \in F$ a set $G(X)$ consisting of all atomic concepts and concepts occurring in the left-hand side of concept inclusion axioms of $\mathcal{O}$, $(iii)$ repeatedly apply the rules in Figure 1 that derive $X \sqsubseteq Y$ with $X \in F$, and $(iv)$ whenever a subsumption of the form $X \sqsubseteq \exists R.Y$ is derived, extend $F$ with $Y$. By induction, it can be shown that the resulting set $S$ of derived subsumptions satisfies the condition of Theorem 1 and contains only $X \sqsubseteq Y$ for which both $X$ and $Y$ occur in $\mathcal{O}$. Since the number of such subsumptions is at most quadratic in the size of $\mathcal{O}$, the classification procedure terminates in polynomial time.

### 2.3 Tracing of Inferences

As discussed in the previous section, reasoning in $\mathcal{EL}^+$ is typically performed by computing a set that is *closed* under (restricted) applications of rules in Figure 1, from which all entailed subsumptions of interest (e.g., subsumptions between all atomic concepts for classification) can be obtained using Theorem 1. It is not difficult to use the same procedure to also keep track of how each derived subsumption was obtained. Essentially, for every derived subsumptions, the procedure can store the *inference information* like in (1)–(10). The inference information specifies by which rules the subsumption was obtained, from which premises, and using which side conditions. We refer to a rule application procedure that retains this kind of information as *tracing*.

Tracing of inferences can be used for extracting *proofs* that represent a sequence of inferences producing the given subsumption in the end. Proofs can be reconstructed by simply traversing the inferences backwards: first take the inferences for the given subsumption, then for the subsumptions used in the premises of such inferences, and so on, disregarding cycles. Proofs can be used to automatically check correctness of entailments (proof checking), or to explain how the entailments were produced to the user. The latter is important for *ontology debugging* when the axioms responsible for erroneous conclusions need to be identified. For example, from Figure 2, one can see that the subsumption $A \sqsubseteq C$ was proved in 5 inference steps involving *intermediate subsumptions* $A \sqsubseteq A$, $A \sqsubseteq \exists R.B$, $A \sqsubseteq \exists H.B$, and $B \sqsubseteq B$. From the side conditions of the inferences (4), (7), and (8) used in this derivation, one can see that only axioms $A \sqsubseteq \exists R.B$, $R \sqsubseteq H$, and $\exists H.B \sqsubseteq C$ in $\mathcal{O}$ are responsible for the conclusion $A \sqsubseteq C$.

### 2.4 Goal-Directed Tracing of Inferences

Keeping information about all inferences in addition to storing the derived subsumptions can be impractical. First, the number of inferences can be significantly larger than the number of the derived conclusions (some conclusions may be derived multiple times by different rules). Second, storing each inference requires more memory than storing just the conclusion of the inference. Finally, inferences may not be required very often. For example, when debugging an ontology, one is usually interested in proofs for only few (unexpected) subsumptions. In applications using large ontologies, such as SNOMED CT, which involve hundreds of thousands of concepts, avoiding storing unnecessary information can make a significant difference in practice.

In this paper, we propose a method for finding the inferences used in the proofs of given subsumptions without performing *full tracing*, i.e., storing all inferences during rule application. The main idea is $(i)$ to over-estimate a subset of subsumptions in the closure from which the given subsumption can be derived (in one inference step), $(ii)$ re-apply the inferences for these subsumptions to find from which premises the subsumption is actually derived, and $(iii)$ repeat the process for these premises.

To illustrate the method, suppose that we have computed the closure (1)–(10) under the rules in Figure 1 (without recording the inference information), and our goal is to identify how subsumption $B \sqsubseteq C$ was derived. By inspecting the conclusions of rules in Figure 1, we can see that every subsumption of the form $X \sqsubseteq Y$ can be either derived by rules $\mathbf{R_0}$ or $\mathbf{R}_\top$, or by other rules using at least one premise of the form $X \sqsubseteq Z$,

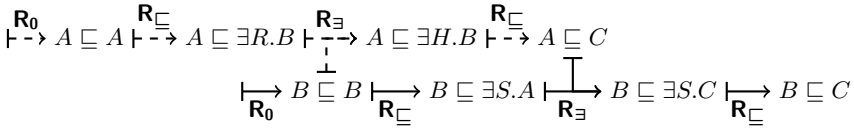


**Fig. 3.** The inferences applied for tracing subsumptions of the form $B \sqsubseteq Y$ (solid lines) and of the form $A \sqsubseteq Y$ (dashed lines) in (1)–(10).

i.e., with the same concept $X$ on the left-hand side. Thus, $B \sqsubseteq C$ must have been derived either by $\mathbf{R_0}$ or $\mathbf{R}_\top$, or from (2), (5), (9), or (10) using other rules (possibly using other premises from (1)–(10)). It is easy to re-apply all such inferences, this time, with tracing. This way, we reconstruct all inferences producing subsumptions of the form $B \sqsubseteq Y$ in (1)–(10), and, in particular of $B \sqsubseteq C$—see the solid lines in Figure 3.

After we have recorded all inferences producing conclusions of the form $B \sqsubseteq Y$ in (1)–(10), we traverse the computed inferences backwards to see which other premises were involved in the proof for $B \sqsubseteq C$. This way, we arrive at premise $A \sqsubseteq C$, for which no inference was recorded so far. This premise has $A$ on the left-hand side, and similarly as above, we can now compute all inferences deriving subsumptions of the form $A \sqsubseteq Y$ in (1)–(10): see the dashed lines in Figure 3. After we have done this, all inferences used in the proof for $B \sqsubseteq C$ are found.

Our tracing procedure does not guarantee to save only inferences that are used in the proof of the given subsumption. For example, to find the proof for the subsumption $B \sqsubseteq \exists S.A$, we would have similarly saved all inferences for $B$ (solid lines in Figure 3) because at that time, we do not know which of these inferences are used in the proof for $B \sqsubseteq \exists S.A$. Note that producing inferences for $A$ would not be necessary in this case since $A \sqsubseteq C$ is not reachable from $B \sqsubseteq \exists S.A$ when traversing inferences backwards. Similarly, tracing of inferences for $B$ is not necessary when finding proofs for $A \sqsubseteq \exists R.B$, and tracing inferences for neither $A$ nor $B$ is necessary when finding proofs for $C \sqsubseteq \exists R.A$. In our experiments, we will demonstrate that only few inferences need to be re-applied when producing proofs of the entailed subsumptions in existing ontologies.

## 3 Tracing of Inferences

In this section, we provide a formal description of the goal-directed procedure outlined in the previous section and prove its correctness. The procedure is formulated in a general way and can be used with arbitrary rule systems. In the end of the section we discuss which properties of the inference system are important for efficiency of the method.

### 3.1 Inferences, Rules, and Closure under the Rules

We start by formalizing the basic notions of inferences and inference rules. Let $\mathbf{E}$ be a fixed countable set of *expressions*. An *inference* over $\mathbf{E}$ is an object *inf* which has a finite set of *premises* $\mathit{inf}.\mathrm{Prem} \subseteq \mathbf{E}$ and a *conclusion* $\mathit{inf}.\mathrm{concl} \in \mathbf{E}$.[1] When $\mathit{inf}.\mathrm{Prem} = \emptyset$, we

[1] We assume that there can be different inferences with the same premises and conclusions.

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

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

say that *inf* is an *initialization inference*. An *inference rule* R over $\mathbf{E}$ is a countable set of inferences over $\mathbf{E}$; it is an *initialization rule* if all these inferences are initialization inferences. In this paper, we view an *inference system* consisting of several rules as one inference rule R containing all the inferences.

Intuitively, an inference corresponds to an application of a rule to particular premises. For example, the application of $\mathbf{R}_\exists$ producing (7) in Example 1 corresponds to an inference *inf* with *inf*.Prem $= \{A \sqsubseteq \exists R.B, B \sqsubseteq B\}$ and *inf*.concl $= (A \sqsubseteq \exists H.B)$. Rule $\mathbf{R}_\exists$ in Figure 1 consists of all such inferences in which the premises and conclusions satisfy the side conditions (for the given $G(\cdot)$ and $\mathcal{O}$). Similarly, $\mathbf{R_0}$ is an initialization rule consisting of *inf* with *inf*.Prem $= \emptyset$ and *inf*.concl $= (C \sqsubseteq C)$ for all $C$.

We say that a set of expressions $Exp \subseteq \mathbf{E}$ is *closed under an inference inf* if *inf*.Prem $\subseteq$ *Exp* implies *inf*.concl $\in$ *Exp*. *Exp* is *closed under an inference rule* R if *Exp* is closed under every inference *inf* $\in$ R. *The closure under* R is the smallest set of expressions Clos(R) that is closed under R. Note that Clos(R) is empty if and only if R does not contain any initialization inferences.

Using the introduced notation, we can now describe the well-known *forward chaining* procedure for computing the closure under the inference rules R. The procedure is formalized in Algorithm 1. Intuitively, the algorithm computes the (expanding) set of expression S, called the *saturation*, by repeatedly matching premises of the rules to S and adding their conclusions back to S until a fixpoint is reached (in contrast, *backward chaining* procedures match the conclusions of rules to the given goals to find the premises). A special care is taken to avoid repeated applications of rules to the same premises. For this purpose, the algorithm uses a queue Q to buffer the conclusions of the inferences. The queue Q is first initialized with conclusions of the initialization inferences (lines 2–3), and then in a cycle (lines 4–9), repeatedly takes the next expression *exp* $\in$ Q, and if it does not occur in the saturation S already, inserts it into S and applies all inferences having this expression as one of the premises and other premises from S (line 8). The conclusions of such inferences are then inserted back into Q. Note that

**Algorithm 2:** Full tracing of inferences

```
fullTracing(R):
input  : R: a set of inferences
output : M: a multimap from expressions to inferences such that M.Keys = Clos(R) and
         M(exp) = {inf ∈ R | inf.Prem ⊆ Clos(R) & inf.concl = exp} (exp ∈ M.Keys)
1  M, Q ← ∅;
2  for inf ∈ R with inf.Prem = ∅ do                          /* initialize */
3      Q.add(inf);
4  while Q ≠ ∅ do                                            /* close */
5      inf ← Q.takeNext();
6      exp ← inf.concl;
7      if exp ∉ M.Keys then
8          M.add(exp ↦ inf);
9          for inf ∈ R with exp ∈ inf.Prem ⊆ M.Keys do
10             Q.add(inf);
11     else
12         M.add(exp ↦ inf);
13 return M;
```

every inference $inf \in \mathsf{R}$ with $inf.\mathrm{Prem} \subseteq \mathsf{S}$ is applied by the algorithm exactly once, namely, when the last premise of this inference is added to $\mathsf{S}$.

### 3.2 Proofs, Inference Oracle, and Full Tracing

Given the closure $\mathsf{S} = \mathrm{Clos}(\mathsf{R})$ under the rules $\mathsf{R}$, i.e, the output of Algorithm 1, we are interested in finding proofs for a given expression $exp \in \mathsf{S}$ consisting of inferences by which $exp$ was derived in $\mathsf{S}$. Formally, a *proof* (in $\mathsf{R}$) is a sequence of inferences $p = inf_1, \ldots, inf_n$ ($inf_i \in \mathsf{R}, 1 \leq i \leq n$) such that $inf_j.\mathrm{Prem} \subseteq \{inf_i.\mathrm{concl} \mid 1 \leq i < j\}$ for each $j$ with $1 \leq j \leq n$. If $exp = inf_n.\mathrm{concl}$ then we say $p$ is a *proof for* $exp$. A proof $p = inf_1, \ldots, inf_n$ for $exp$ is *minimal* if no strict sub-sequence of $inf_1, \ldots, inf_n$ is a proof for $exp$. Note that in this case $inf_i.\mathrm{concl} \neq inf_j.\mathrm{concl}$ when $i \neq j$ $(1 \leq i, j \leq n)$. For example, the sequence of inferences (1)–(10) in Example 1 is a proof for $A \sqsubseteq C$, but not a minimal proof since there exists a sub-sequence (1), (2), (4), (5), (7)–(10), which is also a proof for $A \sqsubseteq C$ (in fact, a minimal proof for $A \sqsubseteq C$, see Figure 2).

To find minimal proofs, one can use an *inference oracle*, that given $exp \in \mathsf{S}$ returns inferences $inf \in \mathsf{R}$ such that $inf.\mathrm{Prem} \subseteq \mathsf{S}$ and $inf.\mathrm{concl} = exp$. Using such an oracle, the proofs can be easily found, e.g., by recursively calling the oracle for the premises of the returned inferences, avoiding repeated requests to ensure minimality of the proofs and termination. If one is interested in retrieving just one proof, it is sufficient to use an inference oracle that returns one inference per expression; if all proofs are required, the oracle should return all inferences producing the given expression from $\mathsf{S}$.

The inference oracle can be implemented by simply precomputing all inferences using a modification of Algorithm 1. This modification is shown in Algorithm 2. We

**Algorithm 3:** Partial tracing of inferences

**partialTracing**($\mathsf{R}, \mathsf{S}, \mathit{Exp}$):
**input** : $\mathsf{R}$: a set of inferences, $\mathsf{S} \subseteq \mathbf{E}$: a subset of expressions,
$\mathit{Exp} \subseteq \mathsf{S} \cap \mathrm{Clos}(\{\mathit{inf} \in \mathsf{R} \mid \mathit{inf}.\mathrm{Prem} \subseteq \mathsf{S}\})$: the expressions to trace
**output** : $\mathsf{M}$: a multimap from expressions to inferences such that $\mathsf{M}.\mathrm{Keys} = \mathit{Exp}$ and
$\mathsf{M}(\mathit{exp}) = \{\mathit{inf} \in \mathsf{R} \mid \mathit{inf}.\mathrm{Prem} \subseteq \mathsf{S}\ \&\ \mathit{inf}.\mathrm{concl} = \mathit{exp}\}$ ($\mathit{exp} \in \mathsf{M}.\mathrm{Keys}$)

```
1  M, Q ← ∅;
2  for inf ∈ R with inf.Prem ⊆ S \ Exp & inf.concl ∈ Exp do        /* initialize */
3      Q.add(inf);
4  while Q ≠ ∅ do                                                  /* close */
5      inf ← Q.takeNext();
6      exp ← inf.concl;
7      if exp ∉ M.Keys then
8          M.add(exp ↦ inf);
9          for inf ∈ R with exp ∈ inf.Prem ⊆ M.Keys ∪ (S \ Exp) & inf.concl ∈ Exp do
10             Q.add(inf);
11     else
12         M.add(exp ↦ inf);
13 return M;
```

refer to this algorithm as *full tracing*. Instead of collecting the expressions derived by the inferences in $\mathsf{S}$, we collect the inferences themselves and store them in a multimap $\mathsf{M}$: for each applied inference *inf*, we add a record $\mathit{exp} \mapsto \mathit{inf}$ to $\mathsf{M}$ where *exp* is the conclusion of *inf* ($\mathsf{M}$ is a multimap because several inferences can be stored for the same conclusion). Thus, the *keys* $\mathsf{M}.\mathrm{Keys}$ of $\mathsf{M}$ is the set of all conclusions of inferences in $\mathsf{S}$. The newly applied inferences are first buffered in the queue $\mathsf{Q}$. When processed, if the conclusion of the inference is new (line 7), we add this inference to $\mathsf{M}$ for the conclusion (line 8) and produce all inferences between this conclusion and the previously derived conclusions (lines 9-10). Otherwise, we just store the new inference and do nothing else (line 12). Note that it is easy to modify this algorithm so that only one inference is stored per conclusion. For this, one can just remove line 12 and make $\mathsf{M}$ an ordinary map. Similarly to Algorithm 1, Algorithm 2 generates every inference at most once.

Once the multimap is computed, the inference oracle can be easily implemented by performing a lookup in the multimap $\mathsf{M}$ for the given expression *exp*.

### 3.3 Partial Tracing of Inferences and Inference Oracle based on Partitions

The construction in Algorithm 2 can be further extended to perform *partial tracing*, that is, to compute not all inferences $\mathit{inf} \in \mathsf{R}$ with $\mathit{inf}.\mathrm{Prem} \subseteq \mathsf{S}$ for $\mathsf{S} = \mathrm{Clos}(\mathsf{R})$, but only those that produce conclusion in a given set of expressions *Exp*. This idea is realized in Algorithm 3. This algorithm constructs a multimap $\mathsf{M}$ that is a restriction of the multimap constructed by Algorithm 2 to the keys from *Exp*. The only difference in this algorithm are lines 2 and 9 where the inferences are applied. First, the algorithm considers only inferences with conclusions in *Exp* ($\mathit{inf}.\mathrm{concl} \in \mathit{Exp}$). Second, since the

algorithm never derives expressions from $\mathsf{S} \setminus \mathit{Exp}$, all premises in $\mathsf{S} \setminus \mathit{Exp}$ are considered in each application of the rule. In other words, if to ignore the premises from $\mathsf{S} \setminus \mathit{Exp}$ in rule applications, the algorithm would look exactly like Algorithm 2. Note that Algorithm 2 is just an instance of Algorithm 3 when $\mathit{Exp} = \mathsf{S} = \mathrm{Clos}(\mathsf{R})$, in which case $\mathsf{S} \setminus \mathit{Exp} = \emptyset$ and the restriction $\mathit{inf}.\mathrm{concl} \in \mathit{Exp}$ can be dropped. In the input of Algorithm 3, we require that $\mathit{Exp}$ is a subset of the closure under the inferences $\mathit{inf}$ with $\mathit{inf}.\mathrm{Prem} \subseteq \mathsf{S}$. This means that every $\mathit{exp} \in \mathit{Exp}$ is derivable by applying $\mathit{inf} \in \mathsf{R}$ to only premises in $\mathsf{S}$. This condition holds trivially if $\mathit{Exp} \subseteq \mathsf{S} = \mathrm{Clos}(\mathsf{R})$.

**Theorem 2 (Correctness of Algorithm 3).** *Let* $\mathsf{R}' = \{\mathit{inf} \in \mathsf{R} \mid \mathit{inf}.\mathrm{Prem} \subseteq \mathsf{S}\}$ *for some set of inferences* $\mathsf{R}$ *and a set of expressions* $\mathsf{S}$*. Let* $\mathit{Exp} \subseteq \mathsf{S} \cap \mathrm{Clos}(\mathsf{R}')$*, and* $\mathsf{M}$ *the output of Algorithm 3 on* $\mathsf{R}, \mathsf{S}$*, and Exp. Then* $\mathsf{M}.\mathrm{Keys} = \mathit{Exp}$ *and* $\mathsf{M}(\mathit{exp}) = \{\mathit{inf} \in \mathsf{R}' \mid \mathit{inf}.\mathrm{concl} = \mathit{exp}\}$ *for each* $\mathit{exp} \in \mathsf{M}.\mathrm{Keys}$*.*

*Proof.* First, observe that $\mathsf{M}.\mathrm{Keys} \subseteq \mathit{Exp}$ since only inferences $\mathit{inf} \in \mathsf{R}$ with $\mathit{inf}.\mathrm{concl} \in \mathit{Exp}$ are added to $\mathsf{Q}$. Next we show that $\mathsf{M}.\mathrm{Keys} = \mathit{Exp}$. Let $\mathrm{Clos}' = \mathsf{M}.\mathrm{Keys} \cup (\mathrm{Clos}(\mathsf{R}') \setminus \mathit{Exp})$. Note that $\mathrm{Clos}' \subseteq \mathrm{Clos}(\mathsf{R}')$ since $\mathsf{M}.\mathrm{Keys} \subseteq \mathit{Exp} \subseteq \mathrm{Clos}(\mathsf{R}')$. We claim that $\mathrm{Clos}'$ is closed under $\mathsf{R}'$, which implies that $\mathrm{Clos}' = \mathrm{Clos}(\mathsf{R}')$ since $\mathrm{Clos}(\mathsf{R}')$ is the minimal set closed under $\mathsf{R}'$. This proves $\mathsf{M}.\mathrm{Keys} = \mathit{Exp}$.

To prove that $\mathrm{Clos}'$ is closed under $\mathsf{R}'$, take an arbitrary inference $\mathit{inf} \in \mathsf{R}'$ such that $\mathit{inf}.\mathrm{Prem} \subseteq \mathrm{Clos}'$. We show that $\mathit{inf}.\mathrm{concl} \in \mathrm{Clos}'$. There are two cases possible:

1. $\mathit{inf}.\mathrm{concl} \notin \mathit{Exp}$: Since $\mathrm{Clos}(\mathsf{R}')$ is closed under $\mathsf{R}'$, in particular, $\mathit{inf}.\mathrm{concl} \in \mathrm{Clos}(\mathsf{R}') \setminus \mathit{Exp}$, so $\mathit{inf}.\mathrm{concl} \in \mathrm{Clos}'$.
2. $\mathit{inf}.\mathrm{concl} \in \mathit{Exp}$: Since $\mathit{inf}.\mathrm{Prem} \subseteq \mathrm{Clos}' \cap \mathsf{S}$, then $\mathit{inf}.\mathrm{Prem} \subseteq \mathsf{M}.\mathrm{Keys} \cup (\mathsf{S} \setminus \mathit{Exp})$. Hence $\mathit{inf}$ will be applied either in line 2 if $\mathit{inf}.\mathrm{Prem} \subseteq (\mathsf{S} \setminus \mathit{Exp})$, or in line 9 for the last premise $\mathit{exp} \in \mathit{inf}.\mathrm{Prem} \cap \mathsf{M}.\mathrm{Keys}$ added to $\mathsf{M}.\mathrm{Keys}$. In both cases, $\mathit{inf}$ is added to $\mathsf{Q}$, which means that $\mathit{inf}.\mathrm{concl} \in \mathsf{M}.\mathrm{Keys} \subseteq \mathrm{Clos}'$.

It remains to prove that $\mathsf{M}(\mathit{exp}) = \{\mathit{inf} \in \mathsf{R}' \mid \mathit{inf}.\mathrm{concl} = \mathit{exp}\}$ for $\mathit{exp} \in \mathsf{M}.\mathrm{Keys}$:

($\supseteq$): Take any $\mathit{exp} \in \mathit{Exp}$ and any $\mathit{inf} \in \mathsf{R}'$ with $\mathit{inf}.\mathrm{concl} = \mathit{exp}$. From Case 2 above, it follows that $\mathit{inf} \in \mathsf{M}(\mathit{exp})$.

($\subseteq$): Take any $\mathit{exp} \in \mathsf{M}.\mathrm{Keys}$ and any $\mathit{inf} \in \mathsf{M}(\mathit{exp})$. Then from lines 6, 8, and 12 of Algorithm 3 we have $\mathit{inf}.\mathrm{concl} = \mathit{exp}$ and from lines 2 and 9 we have $\mathit{inf}.\mathrm{Prem} \subseteq \mathsf{M}.\mathrm{Keys} \cup (\mathsf{S} \setminus \mathit{Exp})$. Since $\mathsf{M}.\mathrm{Keys} \subseteq \mathit{Exp} \subseteq \mathsf{S}$, we obtain that $\mathit{inf} \in \mathsf{R}'$. □

Performance of Algorithm 3 depends on whether it can efficiently enumerate the inferences in lines 2 and 9. Enumerating inferences in line 9 can be simply done by applying all inferences with the given premise $\mathit{exp}$ and other premises from $\mathsf{S}$ similarly to Algorithm 2, and disregarding all inferences that do not satisfy the additional conditions (it is reasonable to assume that the number of inference applicable to a given expression $\mathit{exp}$ is already small to start with). Efficient enumeration of inferences in line 2, however, is more problematic and may depend on the particular inference system $\mathsf{R}$ and set $\mathit{Exp}$. As shown in Section 2.4, if we take $\mathsf{R}$ to be the $\mathcal{EL}^+$ rules in Figure 1, and $\mathit{Exp}$ the set of subsumptions of the form $X \sqsubseteq Y$ for a fixed concept $X$, then the inferences $\mathit{inf} \in \mathsf{R}$ with $\mathit{inf}.\mathrm{Prem} \subseteq \mathsf{S} \setminus \mathit{Exp}$ and $\mathit{inf}.\mathrm{concl} \in \mathit{Exp}$ can be only inferences by $\mathbf{R_0}$ and $\mathbf{R}_\top$ producing $X \sqsubseteq X$ and $X \sqsubseteq \top$ respectively—the other rules cannot derive a subsumption of the form $X \sqsubseteq Z \in \mathit{Exp}$ without a premise of the form $X \sqsubseteq Y \in \mathit{Exp}$.

If all expressions $\mathbf{E}$ can be partitioned on subsets *Exp* for which the set of inferences $\{\mathit{inf} \in \mathsf{R} \mid \mathit{inf}.\text{Prem} \subseteq \mathsf{S} \setminus \mathit{Exp} \;\&\; \mathit{inf}.\text{concl} \in \mathit{Exp}\}$ can be efficiently enumerated, then one can implement an inference oracle by $(i)$ identifying the partition *Exp* to which the given expression *exp* belongs, $(ii)$ computing the inferences for all expressions in *Exp* using Algorithm 3, and $(iii)$ returning the inferences for $\mathit{exp} \in \mathit{Exp}$. This approach is more goal-directed compared to the oracle using full tracing, as it only computes and stores the inferences producing conclusions in one partition.

## 4 Goal-directed Tracing of Inferences in $\mathcal{EL}^+$

Next we show how to use the approach from Section 3 to implement the tracing procedure for $\mathcal{EL}^+$ described in Section 2.4. We assume that the closure $\mathsf{S}$ under the rules $\mathsf{R}$ in Figure 1 is computed (for the given parameters $\mathcal{O}$, $F$, and $G(\cdot)$) and we are given a subsumption $X \sqsubseteq Y \in \mathsf{S}$ for which all inferences used in the proofs should be found.

To partition the closure $\mathsf{S}$, we use the partitioning method used for incremental reasoning in $\mathcal{EL}^+$ [11]. Specifically, for a concept $X$ let $\mathsf{S}[X]$ be the subset of $\mathsf{S}$ consisting of all subsumption $X \sqsubseteq Y \in \mathsf{S}$ for all $Y$. Clearly, $\mathsf{S}$ is partitioned on disjoint subsets $\mathsf{S}[X]$ for different $X$. By inspecting the rules in Figure 1, it is easy to see that if the conclusion of the rule belongs to $\mathsf{S}[X]$ then either it has a premise that belongs to $\mathsf{S}[X]$ as well, or it does not have any premises. Thus, for each $\mathit{Exp} = \mathsf{S}[X]$ and each inference *inf* such that $\mathit{inf}.\text{Prem} \in \mathsf{S} \setminus \mathit{Exp}$ and $\mathit{inf}.\text{concl} \in \mathit{Exp}$, we have $\mathit{inf}.\text{Prem} = \emptyset$. Hence, if Algorithm 3 is applied with $\mathit{Exp} = \mathsf{S}[X]$, only initialization inferences need to be applied in line 2, which can be done very efficiently.

The inference oracle based on Algorithm 3 can be used to compute the proofs for the derived subsumptions $\mathit{exp} = (X \sqsubseteq Y) \in \text{Clos}$ in a goal directed way. For this, we first need to apply Algorithm 3 for the partition $\mathit{Exp} = \mathsf{S}[X]$, then, retrieve all inferences *inf* with conclusion *exp* from the set of computed inferences, and finally, repeat the process for each premise of the retrieved inferences. This process can be adapted accordingly if only one proof should be generated or proofs are to be explored interactively by the user. Since most inferences by the rules in Figure 1 are applied within individual partitions, this should not result in many calls of Algorithm 3, and since partitions are typically quite small, not many inferences should be saved. We will present empirical evidences confirming these claims in Section 6.

*Example 2.* Consider the set $\mathsf{S}$ consisting of subsumptions (1)–(10) derived in Example 1. These subsumptions are assigned to partitions $A$, $B$, and $C$. To compute the inferences with conclusion $B \sqsubseteq C$, it is sufficient to apply Algorithm 3 for $\mathit{Exp} = \mathsf{S}[B]$ using the precomputed closure $\mathsf{S}$. It is easy to see that only inferences for (2), (5), (9), and (10) will be produced by this algorithm (see the solid lines in Figure 3). During initialization (line 2), Algorithm 3 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 $\mathsf{S}[B]$, and keeps only inferences that produce subsumptions in the same partition. For example, the inference by $\mathbf{R}_\exists$ producing (7) should be ignored, even though it uses a premise $B \sqsubseteq B$ from partition $\mathsf{S}[B]$.

Note that knowing $\mathsf{S}$ is essential for computing these inferences. E.g., to produce $B \sqsubseteq \exists S.C$ we need to use the premise $A \sqsubseteq C$ from $\mathsf{S}$, which we do not derive during

tracing. Thus, the precomputed set $\mathsf{S}$ is used in our procedure as a 'set of support'[2] to reach the conclusions of interest as fast as possible.

Now, if we want to traverse the inferences backwards to obtain full proofs for $B \sqsubseteq C$, we need to iterate over the premises that were used to derive $B \sqsubseteq C$ and trace their partitions using Algorithm 3. In our case, $B \sqsubseteq C$ was derived from $B \sqsubseteq \exists S.C$, for which the partition $\mathsf{S}[B]$ was already traced, but continuing further to premise $A \sqsubseteq C$ will bring us to partition $\mathsf{S}[A]$. When tracing partition $\mathsf{S}[A]$ using Algorithm 3, we produce the remaining inferences for (1), (4), (7), and (8) used in the proof (see Figure 3). Note that it is not necessary to trace partition $\mathsf{S}[A]$ to find the proof for, e.g., $B \sqsubseteq \exists S.A$ because no subsumption in partition $\mathsf{S}[A]$ was used to derive $B \sqsubseteq \exists S.A$.

## 5 Related Work

In the context of ontologies, most research relevant to the issue of explanations has revolved around *justifications*[3]—the minimal subsets of the ontology which entail the result [3, 5, 8, 9, 16]. 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. The known approaches usually fall into one of the two categories: black-box and glass-box. The former use the underlying reasoning procedure as a black-box and employ generic model-based diagnosis algorithms [17] to find justifications [5, 9, 8]. Thus, they can work for any (monotonic) logic for which a black-box implementation of a decision procedure is available. On the other hand, such methods cannot use the properties of the reasoning procedure for optimizations and recover the steps used to prove the entailment. Thus, it is still up to the user to understand how to obtain the result from the justifications.

Glass-box methods do use the properties of the underlying reasoning procedure, in particular, for recording how inferences are made. They have been designed for tableau [7], automata [3], and consequence-based procedures [4, 18]. The latter methods are similar to what we call full tracing. For example, Sebastiani et. al. [18] encodes every inference performed during $\mathcal{EL}$ classification in propositional Horn logic. E.g., inference (8) in Example 1 would correspond to $s_{[A \sqsubseteq \exists H.B]} \wedge ax_{[\exists H.B \sqsubseteq C]} \rightarrow s_{[A \sqsubseteq C]}$. Once the entire formula $\phi_{\mathcal{O}}$ capturing all inferences is built, justifications, e.g., for $A \sqsubseteq C$, can be found by pinpointing conflicts in $\phi_{\mathcal{O}} \wedge \neg s_{[A \sqsubseteq C]}$ using the corresponding functionality of SAT solvers.

Any algorithm for computing justifications can be optimized using *ontology modularity* techniques to obtain a goal-directed behaviour (see, e.g., [5]). The so-called *locality-based modules* [6] are well-defined fragments of the ontology which guarantee to contain all justifications for a given entailment. Modules can be often computed efficiently and can limit the search for justifications to a smaller subset of the ontology.

We conclude this section by briefly discussing the relationship between our tracing technique and justifications. From any (minimal) proof obtained by unfolding the inferences computed by Algorithm 3 one can extract the axioms used in the side conditions

[2] Analogously to the 'set of support' strategy of resolution, we use premises from $\mathsf{S}$ only if at least one other premise comes from the traced partition.

[3] Not to be confused with justifications in the context of Truth Maintenance Systems.

of inferences ($\mathbf{R}_{\sqsubseteq}$, $\mathbf{R}_{\exists}$, and $\mathbf{R}_{\circ}$ in the $\mathcal{EL}$ case), which represent a subset of the ontology that entails the proved subsumption. This subset, however, might be not a minimal subset for this entailment, but all minimal subsets—that is, *justifications*—can be easily computed from all such sets. Note that this can be done "offline", i.e., the reasoner is not needed after tracing the inferences. Instead of computing the justification by enumerating proofs, one can also simply take the set of all axioms used in the proofs and extract all justifications from this set using any conventional method [4, 9, 5, 8, 18]. Our evaluation shows that these subsets are much smaller than the locality-based modules usually used for this purpose, and computation of justifications from them is much faster.

## 6 Experimental Evaluation

We implemented algorithms 2 and 3 as well as their recursive extension to obtain full proofs within the $\mathcal{EL}^+$ reasoner ELK[4] (which implements Algorithm 1 as its base procedure). We used three large OWL EL ontologies commonly used in evaluations of $\mathcal{EL}$ reasoners [2, 13, 15]: a version of the Gene Ontology (GO),[5] an $\mathcal{EL}^+$-restricted version of the GALEN ontology,[6] and the July 2013 release of SNOMED CT.[7] We used a PC with Intel Core i5-2520M 2.50GHz CPU, with Java 1.6 and 4GB RAM available to JVM. The following experiments have been performed:

**Full tracing overhead:** Our first experiment evaluates the overhead of full tracing (Algorithm 2) 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 2 show that there is roughly x2–x4 time overhead. The memory overhead is also significant which causes the algorithm to run out of memory (OOM) when classifying SNOMED CT with full tracing.[8] In addition, the results show that the maximal number of conclusions per partition is negligible in comparison with the total number of conclusions. Thus the fact that Algorithm 3 saves some inferences not used in proofs of the target expression should not result in a substantial overhead.

**Goal-directed tracing:** Next we evaluate performance of the recursive proof tracing procedure based on goal-directed tracing with partitions (i.e., on 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 the 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.

The results are presented in Table 3. First, they demonstrate that the proposed method is very fast as it takes just a few milliseconds to find all inferences used in

[4] Version 0.5.0, projected release date Oct 2014, see `http://elk.semanticweb.org`.

[5] 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`.

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

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

[8] 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.** Number of axioms, partitions, conclusions and inferences in test ontologies, running time (in ms.) and memory consumption (in MB) for classification with and without full tracing.

| Ontology | Num. of axioms | Num. of partitions | Num. of conclusions (max per partition) | Num. of inferences | No tracing time | No tracing mem. | Full tracing time | Full tracing mem. |
|---|---|---|---|---|---|---|---|---|
| GO | 87,492 | 46,846 | 2,450,505 (395) | 5,715,611 | 1,877 | 546 | 5,004 | 1,102 |
| GALEN | 36,547 | 25,965 | 1,922,549 (1,350) | 3,944,921 | 1,326 | 722 | 3,786 | 765 |
| SNOMED | 297,327 | 371,642 | 19,963,726 (484) | 54,553,961 | 16,968 | 924 | OOM | OOM |

**Table 3.** Results for recursive goal-directed tracing partitions on all atomic subsumptions in the ontology when all (resp. the first) inferences for each conclusion are recursively unfolded. All numbers, e.g., the number of traced partitions, are averaged over all traced subsumptions.

| Ontology | Total # of traced subsumptions | # of traced partitions | # of traced inferences | # of inferences used in proofs | # of used $\sqsubseteq$ axioms | Time (in ms.) |
|---|---|---|---|---|---|---|
| GO | 73,270 | 3.7 (1.4) | 456.2 (244.1) | 94.9 (6.4) | 18.6 (2.5) | 2.0 (1.2) |
| GALEN | 38,527 | 1.9 (1.5) | 414.4 (350.9) | 21.7 (10.1) | 5.2 (4.4) | 1.9 (0.8) |
| SNOMED | 443,402 | 8.6 (1.6) | 788.3 (332.9) | 385.9 (12.7) | 9.7 (3.7) | 10.1 (1.9) |

all proofs of a given subsumption. This is the effect of its goal-directed nature: it only traces inferences which belong to the same partitions as inferences actually used in the proofs, and the partitions are relatively small. From Table 2 one can calculate that the biggest partitions do not exceed 0.07% of the total number of derived subsumptions while on average there are only 122, 152, and 146 inferences per partition in GO, GALEN, and SNOMED CT respectively. 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 considering 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 are 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. We present a more detailed evaluation in the technical report [12], where we separately aggregate results over subsumptions that are provable from at least 10 axioms and show that performance stays on the same level.

**Tracing and justifications:** In our final experiment, we investigate if it makes sense to use our tracing method to improve the computation of justifications. As discussed in Section 5, the set of axioms used in the side conditions of the inferences in each proof must be a superset of some justification. So it is worth to compare the number of such axioms with the sizes of justifications. Finding all justifications is intractable so we consider only the first proof and the first justification. In addition, we compare the number of axioms used in all proofs of a subsumption with the size of the $(\top\bot)^*$-*module* extracted for this subsumption, and the times needed to extract the first justification from

**Table 4.** Computing the set of subsumption axioms used in all proofs $\mathcal{S}_{all}$ or the first proof $\mathcal{S}_1$ vs. extracting the module $\mathcal{M}$ for the signature of the entailment or computing the first justification $\mathcal{J}_1$. Size of $\mathcal{M}$ and $\mathcal{J}_1$ is measured as the number of subsumption axioms (concept equivalence axioms in the ontologies are rewritten as two subsumptions). Running times are in milliseconds.

| Ontology | Num. of traced subsumptions | $\mathcal{S}_{all}$ size (max size) | $\mathcal{S}_1$ size | $\mathcal{S}_{all}$ time | $\mathcal{S}_1$ time | $\mathcal{J}_1^{\mathcal{S}}$ time | $\mathcal{M}$ size | $\mathcal{M}$ time | $\mathcal{J}_1$ size | $\mathcal{J}_1$ time |
|---|---|---|---|---|---|---|---|---|---|---|
| GO | 73,270 | 18.6 (288) | 2.5 | 2.0 | 1.2 | 25.1 | 124.8 | 164 | 2.2 | 85 |
| GALEN | 38,527 | 5.2 (86) | 4.4 | 1.9 | 0.8 | 128.1 | 10,899 | 195 | 4.35 | N/A |
| SNOMED CT | 10,000 (sample) | 9.7 (133) | 3.7 | 10.1 | 1.9 | 19.0 | 38.8 | 887 | 1.9 | 48 |

these sets—as discussed, both sets must contain all justifications. We use the OWL API which provides generic methods for extracting modules and finding justifications.

The results are given in Table 4. While we were able to generate the first justifications for all subsumptions in GO, it was not feasible for GALEN and SNOMED CT (with a 10 hours time-out). The problem with GALEN is that modules are large and even the first justification often takes minutes to pinpoint in the corresponding module (but not in the results of tracing, as explained below). Thus we focus only on the size of modules and their extraction time. In contrast, modules of SNOMED CT are small but take nearly a second to extract (since the ontology is large). So instead of extracting them for all subsumptions, we took a random uniform sample of $10,000$ subsumptions.

For GO and SNOMED CT it can be seen that on average the number of axioms used in the first proof ($\mathcal{S}_1$) is not much larger than the size of the first justification ($\mathcal{J}_1$) while the number of axioms used in all proofs ($\mathcal{S}_{all}$) is 3–7 times smaller than the module ($\mathcal{M}$). The latter difference is several orders of magnitude for GALEN, which induces very large modules due to cyclic axioms. The time required to extract the first justification from $\mathcal{S}_{all}$ ($\mathcal{J}_1^{\mathcal{S}}$ time) is also 2.5–3.5 times smaller then the time to extract the first justification from $\mathcal{M}$ ($\mathcal{J}_1$ time) for the cases where it was possible.

## 7 Summary

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 offline to produce 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 [13, 11]. Specifically, concept subsumers 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.

It is worth pointing out that our goal-directed procedure does not guarantee to be an improvement over full tracing in worst case: it is easy to construct pathological examples which trigger full tracing. This happens, e.g., for $\mathcal{O} = \{A_i \sqsubseteq A_{i+1} \mid 1 \leq i < n\} \cup$

$\{A_n \sqsubseteq A_1\}$, for which every inference is used in the proof of every entailed subsumption (since all concepts are equivalent), or for $\mathcal{O} = \{A \sqsubseteq B_i \mid 1 \leq i \leq n\}$, for which there is only one partition. Arguably, such examples are not to be often expected in practice as the number of entailed subsumers of each concept is usually relatively small.

## 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. Halaschek-Wiener, C., Parsia, B., Sirin, E.: Description logic reasoning with syntactic updates. In: OTM Conferences (1). pp. 722–737 (2006)
8. 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)
9. 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)
10. 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)
11. 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)
12. Kazakov, Y., Klinov, P.: Goal-directed tracing of inferences in $\mathcal{EL}$ ontologies. Tech. rep., University of Ulm (2014), available from `http://elk.semanticweb.org`
13. 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)
14. Kazakov, Y., Krötzsch, M., Simančík, F.: The incredible ELK: From polynomial procedures to efficient reasoning with $\mathcal{EL}$ ontologies. J. of Automated Reasoning 53(1), 1–61 (2014)
15. 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)
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. Sebastiani, R., Vescovi, M.: Axiom pinpointing in lightweight description logics via Horn-SAT encoding and conflict analysis. In: Proc. 22nd Conf. on Automated Deduction (CADE'09). pp. 84–99 (2009)
19. 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)