Correcting Access Restrictions to a Consequence

Martin Knechtel[1] and Rafael Peñaloza[2]

[1] SAP Research Center Dresden, Germany
martin.knechtel@sap.com
[2] Theoretical Computer Science TU Dresden, Germany
penaloza@tcs.inf.tu-dresden.de

Abstract. Recent research has shown that annotations are useful for representing access restrictions to the axioms of an ontology and their implicit consequences. Previous work focused on computing a consequence's access restriction efficiently from the restrictions of its implying axioms. However, a security administrator might not be satisfied since the intended restriction differs from the one obtained through these methods. In this case, one is interested in finding a minimal set of axioms which need changed restrictions. In this paper we look at this problem and present algorithms based on ontology repair for solving it. Our first experimental results on large scale ontologies show that our methods perform well in practice.

1 Introduction

Description Logics (DL) [1] have been successfully used to model a wide variety of real-world application domains. The relevant portions of these domains are described through a DL ontology and highly optimized reasoners can then be used to deduce facts implicitely described in the ontology. In information systems with a huge ontology it is desirable to restrict the access of users to only a portion of the whole ontology, selected in accordance to an appropriate criterion. Motivations might be reducing information overload, filtering with respect to a trust level, or controlled access following a strict policy. For the access control scenario, each axiom is assigned a privacy level and each user is assigned a security clearance. A user can then see only those axioms whose privacy level is exceeded by the clearance of the user. One naive approach would be to maintain a separate sub-ontology obtained from one big ontology for each possible security clearance which means that any update in the ontology needs to be propagated to each of the sub-ontologies, and any change in the privacy levels or security clearances may result in a full recomputation of the sub-ontologies. Moreover, this would require separate reasoning for each sub-ontology. In order to avoid this, one rather keeps only the big ontology and stores the access information for axioms and users so that they can be retrieved easily. The approach proposed in [2] is to use a labeling lattice $(L, \leq)$. Every axiom and user gets a label in L assigned, and the sub-ontology accessible to a user with label ℓ is the set of all axioms whose label is greater than or equal to ℓ. In [2] it was also shown that

any implicit consequence c from the ontology can be assigned a label, called a *boundary*, such that deciding whether a user has access to c requires again only a computationally cheap label comparison.

DL systems consist of an ontology which represents explicit knowledge and a reasoner which makes implicit consequences of this knowledge explicit. The explicit and implied knowledge is exploited by the application by interacting with the DL system. A correct access labeling of an ontology is a difficult task. Indeed, several seemingly harmless axioms might possibly be combined to deduce information that is considered private. On the other hand, an over-restrictive labeling of axioms may cause public information to be inaccessible to some users. If the knowledge engineer finds that the boundary for a given consequence differs from the desired one, then she would like to automatically receive suggestions on how to modify the labeling function and correct this error. In this paper we present some methods in this direction. We assume that the knowledge engineer knows the exact boundary ℓ_g that the consequence c should receive, and propose a set $\mathcal{S}$ of axioms of minimal cardinality such that if all the axioms in $\mathcal{S}$ are relabeled to ℓ_g, then the boundary of c will be ℓ_g. We call $\mathcal{S}$ a *change set*.

We show that the main ideas from axiom-pinpointing [11, 10, 8, 4, 3] can be exploited in the computation of a change set and present a hitting set tree-based black-box approach that yields the desired set. Our experimental results at the end of the paper show that our algorithms behave well in practice.

2 Preliminaries

To keep our presentation and results as general as possible, we impose only minimal restrictions to our ontology language. We just assume that an *ontology* is a finite set, whose elements are called *axioms*, such that every subset of an ontology is itself an ontology. If $\mathcal{O}' \subseteq \mathcal{O}$ and $\mathcal{O}$ is an ontology, then $\mathcal{O}'$ is called a *sub-ontology* of $\mathcal{O}$. A *monotone consequence relation* $\models$ is a binary relation between ontologies $\mathcal{O}$ and *consequences* c such that if $\mathcal{O} \models c$, then for every ontology $\mathcal{O}' \supseteq \mathcal{O}$ it holds that $\mathcal{O}' \models c$. If $\mathcal{O} \models c$, we say that c *follows from* $\mathcal{O}$ or that $\mathcal{O}$ *entails* c. An ontology language specifies which sets of axioms are admitted as ontologies. Consider, for instance, a Description Logic $\mathcal{L}$. Then, an ontology is a finite set of general concept inclusion axioms (GCIs) of the form $C \sqsubseteq D$, with C, D $\mathcal{L}$-concept descriptions and assertion axioms of the form $C(a)$, with C an $\mathcal{L}$-concept description and a an individual name. Examples of consequences are subsumption relations $A \sqsubseteq B$ for concept names A, B.

If $\mathcal{O} \models c$, we may be interested in finding the axioms responsible for this fact. A sub-ontology $\mathcal{S} \subseteq \mathcal{O}$ is called a *MinA* for $\mathcal{O}$,c if $\mathcal{S} \models c$ and for every $\mathcal{S}' \subset \mathcal{S}, \mathcal{S}' \not\models c$. The dual notion of a MinA is that of a diagnosis. A *diagnosis* for $\mathcal{O}$,c is a sub-ontology $\mathcal{S} \subseteq \mathcal{O}$ such that $\mathcal{O} \setminus \mathcal{S} \not\models c$ and $\mathcal{O} \setminus \mathcal{S}' \models c$ for all $\mathcal{S}' \subset \mathcal{S}$.

For a lattice $(L, \leq)$ and a set $K \subseteq L$, we denote as $\bigoplus_{\ell \in K} \ell$ and $\bigotimes_{\ell \in K} \ell$ the *join* (least upper bound) and *meet* (greatest lower bound) of K, respectively. We consider that ontologies are *labeled* with elements of the lattice. More formally,

for an ontology $\mathcal{O}$ there is a labeling function lab that assigns a *label* $\mathsf{lab}(a) \in L$ to every element a of $\mathcal{O}$. We will often use the notation $L_{\mathsf{lab}} := \{\mathsf{lab}(a) \mid a \in \mathcal{O}\}$.

For a user labeled with $\ell \in L$, we denote as $\mathcal{O}_{\geq \ell}$ the sub-ontology $\mathcal{O}_{\geq \ell} := \{a \in \mathcal{O} \mid \mathsf{lab}(a) \geq \ell\}$ visible for him. The sub-ontologies $\mathcal{O}_{\leq \ell}, \mathcal{O}_{=\ell}, \mathcal{O}_{\neq \ell}, \mathcal{O}_{\not\geq \ell}$, and $\mathcal{O}_{\not\leq \ell}$ are defined analogously. This notion is extended to sets of labels in the natural way, e.g. $\mathcal{O}_{=K} := \{a \in \mathcal{O} \mid \mathsf{lab}(a) = \ell \text{ for some } \ell \in K\}$. Conversely, for a sub-ontology $\mathcal{S} \subseteq \mathcal{O}$, we define $\lambda_{\mathcal{S}} := \bigotimes_{a \in \mathcal{S}} \mathsf{lab}(a)$ and $\mu_{\mathcal{S}} := \bigoplus_{a \in \mathcal{S}} \mathsf{lab}(a)$. An element $\ell \in L$ is called *join prime relative to* L_{lab} if for every $K_1, \ldots, K_n \subseteq L_{\mathsf{lab}}$, $\ell \leq \bigoplus_{i=1}^{n} \lambda_{K_i}$ implies that there is $i, 1 \leq i \leq n$ such that $\ell \leq \lambda_{K_i}$. For instance, in Figure 1, ℓ_1 and ℓ_4 are the only elements that are not join prime relative to $L_{\mathsf{lab}} = \{\ell_1, \ldots, \ell_5\}$, since $\ell_1 \leq \ell_2 \oplus \ell_4$ but neither $\ell_1 \leq \ell_2$ nor $\ell_1 \leq \ell_4$ and similarly $\ell_4 \leq \ell_5 \oplus \ell_3$ but neither $\ell_4 \leq \ell_5$ nor $\ell_4 \leq \ell_3$. Join prime elements relative to L_{lab} are called *user labels*. The set of all user labels is denoted as U. When dealing with labeled ontologies, the reasoning problem of interest consists on the computation of a boundary for a consequence c. Intuitively, the boundary divides the user labels ℓ of U according to whether $\mathcal{O}_{\geq \ell}$ entails c or not.

Definition 1 (Boundary). *Let $\mathcal{O}$ be an ontology and c a consequence. An element $\nu \in L$ is called a* boundary *for $\mathcal{O}$,c if for every join prime element relative to L_{lab} ℓ it holds that $\ell \leq \nu$ iff $\mathcal{O}_{\geq \ell} \models c$.*

Given a user label ℓ_u, we will say that the user *sees* a consequence c if $\ell_u \leq \nu$ for some boundary ν. The following lemma relating MinAs and boundaries was shown in [2].

Lemma 1. *If $\mathcal{S}_1, \ldots, \mathcal{S}_n$ are all MinAs for $\mathcal{O}$,c, then $\bigoplus_{i=1}^{n} \lambda_{\mathcal{S}_i}$ is a boundary for $\mathcal{O}$,c.*

A dual result, which relates the boundary with the set of diagnoses, also exists. The proof follows easily from the definitions given in this section.

Lemma 2. *If $\mathcal{S}_1, \ldots, \mathcal{S}_n$ are all diagnoses for $\mathcal{O}$,c, then $\bigotimes_{i=1}^{n} \mu_{\mathcal{S}_i}$ is a boundary for $\mathcal{O}$,c.*

Example 1. Let $(L_d, \leq_d)$ be the lattice shown in Figure 1, and $\mathcal{O}$ a labeled ontology from a marketplace in the Semantic Web with the following axioms

$a_1 : \mathit{EUecoService} \sqcap \mathit{HighperformanceService}(\mathit{ecoCalculatorV1})$
$a_2 : \mathit{HighperformanceService}$
$\quad \sqsubseteq \mathit{ServiceWithLowCustomerNr} \sqcap \mathit{LowProfitService}$
$a_3 : \mathit{EUecoService} \sqsubseteq \mathit{ServiceWithLowCustomerNr} \sqcap \mathit{LowProfitService}$
$a_4 : \mathit{ServiceWithLowCustomerNr} \sqsubseteq \mathit{ServiceWithComingPriceIncrease}$
$a_5 : \mathit{LowProfitService} \sqsubseteq \mathit{ServiceWithComingPriceIncrease}$

where the function lab assigns to each axiom the labels as shown in Figure 1. This ontology entails $c : \mathit{ServiceWithComingPriceIncrease}(\mathit{ecoCalculatorV1})$. The MinAs for $\mathcal{O}$,c are $\{a_1, a_2, a_4\}, \{a_1, a_2, a_5\}, \{a_1, a_3, a_4\}, \{a_1, a_3, a_5\}$, and its diagnoses are $\{a_1\}, \{a_2, a_3\}, \{a_4, a_5\}$. Using Lemma 2, we can compute the boundary as $\mu_{\{a_1\}} \otimes \mu_{\{a_2, a_3\}} \otimes \mu_{\{a_4, a_5\}} = \ell_1 \otimes \ell_2 \otimes \ell_4 = \ell_3$. Valid user labels are $\ell_0, \ell_2, \ell_3, \ell_5$ which represent user roles as illustrated. Only for ℓ_0 and ℓ_3, c is visible.

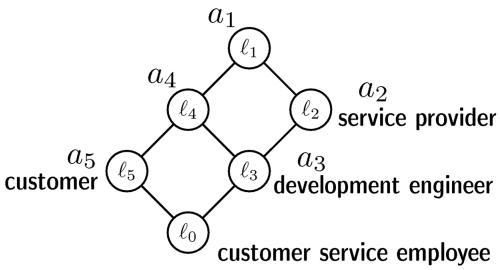

Fig. 1. Lattice $(L_d, \leq_d)$ with 4 user labels and an assignment of 5 axioms to labels

3 Modifying the Boundary

Once the boundary for a consequence c has been computed, it is possible that the knowledge engineer or the security administrator considers this solution erroneous. For instance, the boundary may express that a given user u is able to deduce c, although this was not intended. Alternatively, the boundary may imply that c is a very confidential consequence, only visible to a few, high-clearance users, while in reality c should be more publicly available.

Example 2. The boundary ℓ_3 computed in Example 1 expresses that the consequence c can only be seen by the development engineers and customer service employees (see Figure 1). It could be, however, that c is not expected to be accessible to development engineers, but rather to customers. In that case, we wish to modify the boundary of c to ℓ_5.

The problem we face is how to change the labeling function so that the computed boundary corresponds to the desired label in the lattice. This problem can be formalized and approached in several different ways. In our approach, we fix a goal label ℓ_g and try to modify the labeling of as few axioms as possible so that the boundary equals ℓ_g.

Definition 2. *Let $\mathcal{O}$ be an ontology,* lab *a labeling function, $\mathcal{S} \subseteq \mathcal{O}$ and $\ell_g \in L$ the* goal label. *The* modified assignment $\mathsf{lab}_{\mathcal{S},\ell_g}$ *is given by*

$$\mathsf{lab}_{\mathcal{S},\ell_g}(a) = \begin{cases} \ell_g, & \text{if } a \in \mathcal{S}, \\ \mathsf{lab}(a), & \text{otherwise.} \end{cases}$$

A sub-ontology $\mathcal{S} \subseteq \mathcal{O}$ is called a change set *(CS) for ℓ_g if the boundary for $\mathcal{O}$,c under the labeling function $\mathsf{lab}_{\mathcal{S},\ell_g}$ equals ℓ_g.*

Obviously, the original ontology $\mathcal{O}$ is always a CS set for any goal label if $\mathcal{O} \models c$. However, we are interested in performing minimal changes to the labeling function. Hence, we search for a CS of minimum cardinality.

Let ℓ_g denote the goal label and ℓ_c the computed boundary for c. If $\ell_g \neq \ell_c$, we have three cases: either (1) $\ell_g < \ell_c$, (2) $\ell_c < \ell_g$, or (3) ℓ_g and ℓ_c are incomparable.

In our example, the three cases are given by ℓ_g being ℓ_0, ℓ_4, and ℓ_5, respectively. Consider the first case, where $\ell_g < \ell_c$. Then, from Lemma 2 it follows that any diagnosis $\mathcal{S}$ is a CS for ℓ_g: since $\ell_g < \ell_c$, then for every diagnosis $\mathcal{S}'$, $\ell_g < \mu_{\mathcal{S}'}$. But then, under the new labeling $\mathsf{lab}_{\mathcal{S},\ell_g}$ we get that $\mu_{\mathcal{S}} = \ell_g$. And hence, when the greatest lower bound of all $\mu_{\mathcal{S}'}$ is computed, we obtain ℓ_g as a boundary. Using an analogous argument and Lemma 1, it is possible to show that if $\ell_c < \ell_g$, then every MinA is a CS for ℓ_g. The third case can be solved using a combination of the previous two: if ℓ_g and ℓ_c are incomparable, we can first set as a partial goal $\ell'_g := \ell_g \otimes \ell_c$. Thus, we can first solve the first case, to set the boundary to ℓ'_g, and then, using the second approach, modify this new boundary once more to ℓ_g. Rather than actually performing this task as a two-step computation, we can simply compute a MinA and a diagnose. The union of these two sets yields a CS. Unfortunately, the CS computed this way is not necessarily of minimum cardinality, even if the smallest diagnosis or MinA is used, as shown in the following example.

Example 3. Let $\mathcal{O}$,c and lab be as in Example 1. We then know that $\ell_c := \ell_3$ is a boundary for $\mathcal{O}$,c. Suppose now that the goal label is $\ell_g := \ell_4$. Since $\ell_c < \ell_g$, we know that any MinA is a CS. Since all MinAs for $\mathcal{O}$,c have exactly three elements, any CS produced this way will have cardinality three. However, $\{a_2\}$ or $\{a_3\}$ are also valid CS, whose cardinalities are obviously smaller.

To understand why the minimality of MinAs is not sufficient for obtaining a minimum CS, we can look back to Lemma 1. This lemma says that in order to find a boundary, we need to compute the join of all $\lambda_{\mathcal{S}}$, with $\mathcal{S}$ a MinA, and $\lambda_{\mathcal{S}}$ the meet of the labels of all axioms in $\mathcal{S}$. But then, for any axiom $a \in \mathcal{S}$ such that $\ell_g \leq \mathsf{lab}(a)$, modifying this label to ℓ_g will have no influence in the result of $\lambda_{\mathcal{S}}$. In Example 3, there is a MinA $\{a_1, a_2, a_4\}$, where two axioms, namely a_1 and a_4 have a label greater or equal to $\ell_g = \ell_4$. Thus, the only axiom that needs to be relabeled is in fact a_2, which yields the minimum CS $\{a_2\}$ shown in the example. Basically, we consider every axiom $a \in \mathcal{O}$ such that $\ell_g \leq \mathsf{lab}(a)$ as *fixed* in the sense that it is superfluous for any CS. For this reason, we will deal with a generalization of MinAs and diagnoses, that we call IAS and RAS, respectively.

Definition 3 (IAS,RAS). *A* minimal inserted axiom set *(IAS) for ℓ_g is a subset $I \subseteq \mathcal{O}_{\not\geq \ell_g}$ such that $\mathcal{O}_{\geq \ell_g} \cup I \models c$ and for every $I' \subset I : \mathcal{O}_{\geq \ell_g} \cup I' \not\models c$. A* minimal removed axiom set *(RAS) for ℓ_g is a subset $R \subseteq \mathcal{O}_{\not\leq \ell_g}$ such that $\mathcal{O}_{\not\leq \ell_g} \setminus R \not\models c$ and for every $R' \subset R : \mathcal{O}_{\not\leq \ell_g} \setminus R' \models c$.*

The following theorem justifies the use of IAS and RAS when searching for a CS of minimum cardinality.

Theorem 1. *Let ℓ_c be a boundary for $\mathcal{O}$,c, ℓ_g the goal label, and m_R, m_I and m_U the cardinalities of the smallest RAS, the smallest IAS and the smallest union of an IAS and a RAS for ℓ_g, respectively. Then, for every IAS I and RAS R for ℓ_g it holds:*

- *if $\ell_g < \ell_c$ and $|R| = m_R$, then R is a CS of minimum cardinality,*

– *if* $\ell_c < \ell_g$ *and* $|I| = m_I$*, then* I *is a CS of minimum cardinality,*
– *if* ℓ_c *and* ℓ_g *are incomparable and* $|R \cup I| = m_U$*, then* $I \cup R$ *is a CS of minimum cardinality.*

4 Computing a Minimal Change Set

Naïvely the smallest CS can be found by computing all CS and selecting the smallest. As explained above, the task of computing all CS is related to computing all diagnoses and all MinAs, which has been widely studied in recent years, and there exist black-box implementations based on the hitting set tree (HST) algorithm [7, 12]. Our approach to compute a minimal CS follows similar ideas. The HST algorithm makes repeated calls to an auxiliary procedure that computes a single CS. Further CS are found by building a tree, where nodes are labeled with CS and edges with axioms. If the CS labeling a node has n axioms ($\mathcal{S} := \{a_1, \ldots, a_n\}$), then this node will have n children: the edge to the i-th child labeled with a_i, the child labeled with a CS that is not allowed to contain neither a_i nor any ancestor's edge label. This ensures that each node is labeled with a CS distinct from those of its predecessors.

For the auxiliary procedure to compute a single CS, we will use two sub procedures extracting RAS and IAS, respectively. In Algorithm 1 we present a variation of the logarithmic MinA extraction procedure presented in [5] that is able to compute an IAS or stop once this has reached a size n and return a partial IAS. We also show the RAS variant in Algorithm 2. Given a goal label ℓ_g, if we want to compute a IAS or a partial IAS of size exactly n for a consequence c, then we would make a call to $\mathsf{extract\text{-}partial\text{-}IAS}(\mathcal{O}_{\geq \ell_g}, \mathcal{O}_{\not\geq \ell_g}, c, n)$. Similarly, a call to $\mathsf{extract\text{-}partial\text{-}RAS}(\mathcal{O}_{\not\leq \ell_g}, \mathcal{O}_{\not\leq \ell_g}, c, n)$ yields a RAS of size $\leq n$ or a partial RAS of size exactly n. The cardinality limit will be used to avoid unnecessary computations when looking for the smallest CS.

Given the procedures to extract RAS and IAS, Algorithm 3 extracts a CS. In order to label a node, we compute a CS with $\mathsf{extract\text{-}partial\text{-}CS}(\mathcal{O}, \mathsf{lab}, c, \ell_g, H, n)$, where H is the set of all labels attached to edges on the way from the node to the root of the tree. Note that axioms in H are removed from the search space to extract the IAS and RAS. Furthermore, axioms in the IAS are considered as *fixed* for the RAS computation. The returned set is a CS of size $\leq n$ or a partial CS of size n.

Example 4. Returning to our running example, suppose now that we want to modify the label of consequence c to $\ell_g = \ell_5$. Algorithm 3 starts by making a call to $\mathsf{extract\text{-}partial\text{-}IAS}(\mathcal{O}_{\geq \ell_5}, \mathcal{O}_{\not\geq \ell_5}, c)$.[1] A possible output for this call is $I = \{a_3\}$. We can then call $\mathsf{extract\text{-}partial\text{-}RAS}(\mathcal{O}_{\not\leq \ell_5} \setminus I, \mathcal{O}_{\not\leq \ell_5} \setminus I, c)$, which may output e.g. the set $R = \{a_1\}$. Thus, globally the algorithm returns $\{a_3, a_1\}$, which can be easily verified to be a CS for ℓ_5.

One of the advantages of the HST algorithm is that the labels of any node are always ensured not to contain the label of any of its predecessor nodes. In

[1] For this example, we ignore the cardinality limit, as we want to find only one CS.

Algorithm 1 Compute (partial) IAS

Procedure extract-partial-IAS($\mathcal{O}_{\text{fix}}, \mathcal{O}_{\text{test}}, c, n$)
Input: $\mathcal{O}_{\text{fix}}$: fixed axioms; $\mathcal{O}_{\text{test}}$: axioms; c: consequence; n: limit
Output: first n elements of a minimal $\mathcal{S} \subseteq \mathcal{O}_{\text{test}}$ such that $\mathcal{O}_{\text{fix}} \cup \mathcal{S} \models c$

Global $l := 0, n$
return extract-partial-IAS-r($\mathcal{O}_{\text{fix}}, \mathcal{O}_{\text{test}}, c$)

Subprocedure extract-partial-IAS-r($\mathcal{O}_{\text{fix}}, \mathcal{O}_{\text{test}}, c$)

if $n = l$ **then**
  return $\emptyset$
if $|\mathcal{O}_{\text{test}}| = 1$ **then**
  $l := l + 1$
  return $\mathcal{O}_{\text{test}}$
$\mathcal{S}_1, \mathcal{S}_2 :=$ halve($\mathcal{O}_{\text{test}}$) (partition $\mathcal{O}_{\text{test}}$ so that $||\mathcal{S}_1| - |\mathcal{S}_2|| \leq 1$)
if $\mathcal{O}_{\text{fix}} \cup \mathcal{S}_1 \models c$ **then**
  return extract-partial-IAS-r($\mathcal{O}_{\text{fix}}, \mathcal{S}_1, c$)
if $\mathcal{O}_{\text{fix}} \cup \mathcal{S}_2 \models c$ **then**
  return extract-partial-IAS-r($\mathcal{O}_{\text{fix}}, \mathcal{S}_2, c$)
$\mathcal{S}'_1 :=$ extract-partial-IAS-r($\mathcal{O}_{\text{fix}} \cup \mathcal{S}_2, \mathcal{S}_1, c$)
$\mathcal{S}'_2 :=$ extract-partial-IAS-r($\mathcal{O}_{\text{fix}} \cup \mathcal{S}'_1, \mathcal{S}_2, c$)
return $\mathcal{S}'_1 \cup \mathcal{S}'_2$

particular this means that even if we compute a partial CS, the algorithm will still correctly find all CS that do not contain any of the partial CS found during the execution. Since we are interested in finding the CS of minimum cardinality, we can set the limit n to the size of the smallest CS found so far. This limit is initially fixed to the size of the ontology. If extract-partial-CS outputs a set with fewer elements, we are sure that this is indeed a full CS, and our new smallest known CS. The HST algorithm will not find all CS in this way, but we can be sure that one CS with the minimum cardinality will be found. The idea of limiting cardinality for finding the smallest CS can be taken a step further by not expanding each node for all the axioms in it, but rather only on the first $n - 1$, where n is the size of the smallest CS found so far. This further reduces the search space by decreasing the branching factor of the search tree. Notice that the highest advantage of this second optimization appears when the HST is constructed in a depth-first fashion. In that case, a smaller CS found further below in the tree will reduce the branching factor of all its predecessors. So the cardinality limit reduces the search space in two dimensions: (1) the computation of a single CS is limited to n axioms and (2) only $n - 1$ axioms are expanded from each node. The following theorem shows that such a variant of the HST algorithm is correct.

Theorem 2. *Let $\mathcal{O}$ be an ontology, c a consequence with $\mathcal{O} \models c$, and ℓ_g a goal label. If m is the minimum cardinality of all CS for ℓ_g, the HST algorithm described above outputs a CS $\mathcal{S}$ such that $|\mathcal{S}| = m$.*

Algorithm 2 Compute (partial) RAS

Procedure extract-partial-RAS($\mathcal{O}_{\mathsf{nonfix}}, \mathcal{O}_{\mathsf{test}}, c, n$)
Input: $\mathcal{O}_{\mathrm{nonfix}}$: axioms; $\mathcal{O}_{\mathrm{test}} \subseteq \mathcal{O}_{\mathrm{nonfix}}$: axioms; c: consequence; n: limit
Output: first n elements of a minimal $\mathcal{S} \subseteq \mathcal{O}_{\mathrm{test}}$ such that $\mathcal{O}_{\mathrm{nonfix}} \setminus \mathcal{S} \not\models c$

Global $l := 0, \mathcal{O}_{\mathrm{nonfix}}, n$
return extract-partial-RAS-r($\emptyset, \mathcal{O}_{\mathsf{test}}, c$)

Subprocedure extract-partial-RAS-r($\mathcal{O}_{\mathsf{hold}}, \mathcal{O}_{\mathsf{test}}, c$)

if $n = l$ **then**
  return $\emptyset$
if $|\mathcal{O}_{\mathrm{test}}| = 1$ **then**
  $l := l + 1$
  return $\mathcal{O}_{\mathrm{test}}$
$\mathcal{S}_1, \mathcal{S}_2 :=$ halve($\mathcal{O}_{\mathrm{test}}$) (partition $\mathcal{O}_{\mathrm{test}}$ so that $||\mathcal{S}_1| - |\mathcal{S}_2|| \leq 1$)
if $\mathcal{O}_{\mathrm{nonfix}} \setminus (\mathcal{O}_{\mathrm{hold}} \cup \mathcal{S}_1) \not\models c$ **then**
  return extract-partial-RAS-r($\mathcal{O}_{\mathrm{hold}}, \mathcal{S}_1, c$)
if $\mathcal{O}_{\mathrm{nonfix}} \setminus (\mathcal{O}_{\mathrm{hold}} \cup \mathcal{S}_2) \not\models c$ **then**
  return extract-partial-RAS-r($\mathcal{O}_{\mathrm{hold}}, \mathcal{S}_2, c$)
$\mathcal{S}'_1 :=$ extract-partial-RAS-r($\mathcal{O}_{\mathrm{hold}} \cup \mathcal{S}_2, \mathcal{S}_1, c$)
$\mathcal{S}'_2 :=$ extract-partial-RAS-r($\mathcal{O}_{\mathrm{hold}} \cup \mathcal{S}'_1, \mathcal{S}_2, c$)
return $\mathcal{S}'_1 \cup \mathcal{S}'_2$

Proof. The described algorithm outputs a CS since the globally stored and finally returned $\mathcal{S}$ is only modified when the output of extract-partial-CS has size strictly smaller than the limit n, and hence only when this is indeed a CS itself. Suppose now that the output $\mathcal{S}$ is such that $m < |\mathcal{S}|$, and let $\mathcal{S}_0$ be a CS such that $|\mathcal{S}_0| = m$, which exists by assumption. Then, every set obtained by calls to extract-partial-CS has size strictly greater than m, since otherwise, $\mathcal{S}$ and n would be updated. Consider now an arbitrary set $\mathcal{S}'$ found during the execution through a call to extract-partial-CS, and let $\mathcal{S}'_n := \{a_1, \ldots, a_n\}$ be the first n elements of $\mathcal{S}'$. Since $\mathcal{S}'$ is a (partial) CS, it must be the case that $\mathcal{S}_0 \not\subseteq \mathcal{S}'_n$ since every returned CS is minimal in the sense that no axiom might be removed to obtain another CS. Then, there must be an $i, 1 \leq i \leq n$ such that $a_i \notin \mathcal{S}_0$. But then, $\mathcal{S}_0$ will still be a CS after axiom $\{a_i\}$ has been removed. Since this argument is true for all nodes, it is in particular true for all leaf nodes, but then they should not be leaf nodes, since a new CS, namely $\mathcal{S}_0$ can still be found by expanding the HST, which contradicts the fact that $\mathcal{S}$ is the output of the algorithm. □

Example 5. Returning to our running example, suppose that we want to set the label of c to $\ell_g = \ell_0$. Algorithm 3 first calls extract-partial-RAS($\mathcal{O}_{\not\leq \ell_0}, \mathcal{O}_{\not\leq \ell_0}, c, 5$). A possible output of this call is $R = \{a_2, a_3\}$. The tree now branches through a_2 and a_3. In the first case it calls extract-partial-RAS($\mathcal{O}_{\not\leq \ell_0}, \mathcal{O}_{\not\leq \ell_0} \setminus \{a_2\}, c, 2$), which could yield $R = \{a_4, a_5\}$. This might be a partial CS since its size equals the cardinality limit. The next call extract-partial-RAS($\mathcal{O}_{\not\leq \ell_0}, \mathcal{O}_{\not\leq \ell_0} \setminus \{a_2, a_4\}, c, 2$) yields the smallest $R = \{a_1\}$, and the HST terminates. Notice that if $\{a_1\}$ had been the first change set found, the process would have immediately terminated.

Algorithm 3 Compute (partial) Change Set

Procedure extract-partial-CS$(\mathcal{O}, \mathsf{lab}, c, \ell_g, H, n)$

1: $\ell_c :=$ hst-boundary$(\mathcal{O}, c)$ function defined in [2]
2: **return** extract-partial-CS$(\mathcal{O}, \mathsf{lab}, c, \ell_g,$
$\ell_g \not< \ell_c \wedge \mathcal{O}_{\geq \ell_g} \not\models c,$
$\ell_g \not> \ell_c \wedge \mathcal{O}_{\not\leq \ell_g} \models c,\ H, n)$

Procedure extract-partial-CS$(\mathcal{O}, \mathsf{lab}, c, \ell_g, \mathsf{is}_I, \mathsf{is}_R, H, n)$

Input: $\mathcal{O}, \mathsf{lab}$: labeled ontology; c: consequence; ℓ_g: goal label; is_I: decision to compute IAS; is_R: decision to compute RAS; H: HST edge labels; n: limit

Output: first n elements of a minimal CS $\mathcal{S} \subseteq \mathcal{O}$

1: **if** $1 \geq n$ or $\mathrm{is}_I \wedge \mathcal{O}_{\geq \ell_g} \cup (\mathcal{O}_{\not\geq \ell_g} \setminus H) \not\models c$ or $\mathrm{is}_R \wedge H \models c$ **then**
2: **return** $\emptyset$ (HST normal termination)
3: **if** is_I **then**
4: $I :=$ extract-partial-IAS$(\mathcal{O}_{\geq \ell_g}, \mathcal{O}_{\not\geq \ell_g} \setminus H, c, n)$
5: **if** is_R and $\mathcal{O}_{\not\leq \ell_g} \setminus I \models c$ **then**
6: $R :=$ extract-partial-RAS$(\mathcal{O}_{\not\leq \ell_g} \setminus I, \mathcal{O}_{\not\leq \ell_g} \setminus (I \cup H), c, n - |I|)$
7: **return** $I \cup R$

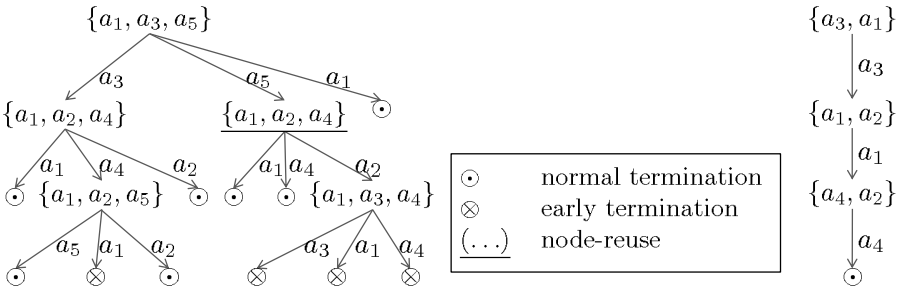

Fig. 2. Hitting Set Trees to compute all MinAs (left) and a minimal change set for $\ell_g = \ell_5$ (right)

Efficient implementations of the original version of the HST algorithm rely on several optimizations. Two standard optimizations described in the literature are node-reuse and early path termination (see, e.g. [7, 12, 2]). Node-reuse keeps a history of all nodes computed so far in order to avoid useless (and usually expensive) calls to the auxiliary procedure that computes a new node. Early path termination, on the other hand, prunes the hitting set tree by avoiding expanding nodes when no new information can be derived from further expansion. In order to avoid unnecessary confusion, we have described the modified HST algorithm without including these optimizations. However, it should be clear that both, node-reuse and early path termination, can be included in the algorithm without destroying its correctness. The implementation used for our experiments contain these two optimizations.

Figure 2 shows the expansion of the HST trees when computing all MinAs and all diagnoses, in comparison with the one obtained for computing a minimal change set for $\ell_g = \ell_5$, using the ontology and consequences of Example 1.

This paper's results are a continuation of work in [9], where we had not one Hitting Set Tree Algorithm but two separately for the smallest IAS and the smallest RAS. This paper's variant is guaranteed to find the smallest CS, as given in the Proof above. For a CS consisting of an IAS and a RAS, computing a smallest of both does not necessarily yield the smallest CS, as the following example shows. Assume $\{a_1, a_2\}, \{a_2, a_3\}$ are the smallest RAS and $\{a_1, a_4\}$ is the smallest IAS, then $\{a_1, a_2, a_4\}$ is the smallest CS, but choosing one smallest IAS and one smallest RAS might yield a CS of cardinality 4. In [9] we also investigated the performance gain by taking not only advantage of fixing a subset of the axioms and limiting cardinality but also by taking the labels of the remaining axioms into account.

5 Empirical Evaluation

We implemented and evaluated our algorithms empirically with large practical ontologies. The test system is identical to one used previous work in [2], so we describe it here very briefly. The two labeling lattices used are $(L_d, \leq_d)$, already introduced in Figure 1, and the linear order $(L_l, \leq_l)$ with 6 elements $L_l = L_d = \{\ell_0, \ldots, \ell_5\}$ with $\leq_l := \{(\ell_n, \ell_{n+1}) \mid \ell_n, \ell_{n+1} \in L_l \wedge 0 \leq n \leq 5\}$. We used the two ontologies O^{SNOMED} and O^{FUNCT} with different expressivity and types of consequences for our experiments. The Systematized Nomenclature of Medicine, Clinical Terms (SNOMED CT) is a comprehensive medical and clinical ontology which is built using the Description Logic (DL) $\mathcal{EL}+$. From the January/2005 release of the DL version, which contains 379,691 concept names, 62 object property names, and 379,704 axioms, and entails more than five million subsumptions, we used a sampled set of 27,477 positive subsumptions. O^{FUNCT} is an OWL-DL ontology for functional description of mechanical engineering solutions [6]. It has 115 concept names, 47 object property names, 16 data property names, 545 individual names, 3,176 axioms, and the DL expressivity is $\mathcal{SHOIN}(\mathbf{D})$. Its 716 consequences are 12 subsumption and 704 instance relationships (class assertions).

We computed the boundary ℓ_c of each consequence c of the ontologies with the algorithms described in [2] and then computed the change set for goal boundary $\ell_g = \ell_3$. Consequences where $\ell_c = \ell_g$ were not considered. Thus, from the 716 consequences in O^{FUNCT}, we have 415 remaining with labeling lattice $(L_d, \leq_d)$ and 474 remaining with $(L_l, \leq_l)$. From the 27,477 consequences in O^{SNOMED} we have 23,695 remaining with labeling lattice $(L_d, \leq_d)$ and 25,897 with $(L_l, \leq_l)$.

Table 1 contains results for the 4 combinations of the two ontologies and the two labeling lattices. For each of them we tested our algorithm against the basic approach of computing all MinAs and diagnoses. We limit the number of computed MinAs and CS to 10, so our algorithms might not find the smallest change set before reaching the limit. We measure the quality of the presented variants given those limitations at execution time. Table 1 lists the ratio of correct solutions where at least 1 correct change set was computed, and the ratio of optimal solutions where the limit was not reached during the computation

Ont.	Lattice	Variant	Runtime Limit per goal	Time (minutes)	Ratio of correct solutions	Ratio of optimal solutions
$\mathcal{O}^{\text{FUNCT}}$	nonlinear	all diagnoses and MinAs	$\leq$ 10 MinA	44.05	96%	47%
		a minimal CS	$\leq$ 10 (partial) CS	8.66	100%	98%
	linear	all diagnoses and MinAs	$\leq$ 10 MinA	54.46	98%	49%
		a minimal CS	$\leq$ 10 (partial) CS	8.61	100%	99%
$\mathcal{O}^{\text{SNOMED}}$	nonlinear	all diagnoses and MinAs	$\leq$ 10 MinA	184.76	100%	75%
		a minimal CS	$\leq$ 10 (partial) CS	10.51	100%	100%
	linear	all diagnoses and MinAs	$\leq$ 10 MinAs	185.35	100%	75%
		a minimal CS	$\leq$ 10 (partial) CS	28.14	100%	98%

Table 1. Results comparing our with the reference approach in 4 test settings

and thus yielded the smallest change set possible. Notice however that the ratio of cases with the minimal change set successfully computed might be higher, including those where the limitation was reached but the minimal change set was already found.

Computing all MinAs is clearly outperformed by our optimized approach. To conclude, fixed sub-ontologies and cardinality limit are optimizations with reasonable impact.

6 Conclusions

Previous work has studied labeled ontologies and methods to compute boundaries for their consequences. In this paper we considered scenarios where a security administrator is not satisfied with the access restriction level computed from the access restriction levels of the implying axioms. Based on ontology repair techniques we developed algorithms to compute a change set of minimal cardinality, which contains axioms to be relabeled in order to yield a consequence's access restriction. The base problem, finding the smallest MinA and diagnosis without computing all of them might be interesting beyond our application domain. Our algorithms take advantage of (1) fixing a subset of the axioms which are not part of the search space, and (2) limiting cardinality of change sets to be computed in the Hitting Set Tree to the smallest known change set. We implemented the algorithms and have first experimental results on large-scale ontologies which show that our ideas yield tangible improvements in both the execution time and the quality of the solution.

As future work we intend to study the problem of finding change sets for several consequences (each with its own goal label) simultaneously. We will also look at more flexible restrictions on the goal label and other criteria for the minimality of change sets for example not counting the amount of changed axiom label assignments but the distance of the new from the old label in the lattice or the amount of other consequence's boundaries changed.

References

1. F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2007.
2. F. Baader, M. Knechtel, and R. Peñaloza. A generic approach for large-scale ontological reasoning in the presence of access restrictions to the ontology's axioms. In *Proc. of ISWC'09*, volume 5823 of *LNCS*, pages 49–64, 2009.
3. F. Baader and R. Peñaloza. Automata-based axiom pinpointing. *Journal of Automated Reasoning*, 2010. Special Issue: IJCAR 2008. To appear.
4. F. Baader and R. Peñaloza. Axiom pinpointing in general tableaux. *Journal of Logic and Computation*, 2010. Special Issue: Tableaux'07. To appear.
5. F. Baader and B. Suntisrivaraporn. Debugging SNOMED CT using axiom pinpointing in the description logic $\mathcal{EL}^+$. In *Proc. of KR-MED'08*, 2008.
6. A. Gaag, A. Kohn, and U. Lindemann. Function-based solution retrieval and semantic search in mechanical engineering. In *Proc. of ICED'09*, 2009.
7. A. Kalyanpur, B. Parsia, M. Horridge, and E. Sirin. Finding all justifications of OWL DL entailments. In *Proc. of ISWC'07 + ASWC'07*, 2007.
8. A. Kalyanpur, B. Parsia, E. Sirin, and J. A. Hendler. Debugging unsatisfiable classes in OWL ontologies. *J. Web Sem.*, 3(4):268–293, 2005.
9. M. Knechtel and R. Peñaloza. A Generic Approach for Correcting Access Restrictions to a Consequence. In *Proc. of ESWC'10*, 2010. To appear.
10. T. Meyer, K. Lee, R. Booth, and J. Z. Pan. Finding maximally satisfiable terminologies for the description logic $\mathcal{ALC}$. In *Proc. of AAAI'06*, 2006.
11. S. Schlobach and R. Cornet. Non-standard reasoning services for the debugging of description logic terminologies. In *Proc. of IJCAI'03*, pages 355–362, 2003.
12. B. Suntisrivaraporn. *Polynomial-Time Reasoning Support for Design and Maintenance of Large-Scale Biomedical Ontologies*. PhD thesis, TU Dresden, 2008.