A Generic Approach for Large-Scale Ontological Reasoning in the Presence of Access Restrictions to the Ontology's Axioms

Franz Baader[1], Martin Knechtel[2], and Rafael Peñaloza[1]

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

Abstract. The framework developed in this paper can deal with scenarios where selected sub-ontologies of a large ontology are offered as views to users, based on criteria like the user's access right, the trust level required by the application, or the level of detail requested by the user. Instead of materializing a large number of different sub-ontologies, we propose to keep just one ontology, but equip each axiom with a label from an appropriate labeling lattice. The access right, required trust level, etc. is then also represented by a label (called user label) from this lattice, and the corresponding sub-ontology is determined by comparing this label with the axiom labels. For large-scale ontologies, certain consequence (like the concept hierarchy) are often precomputed. Instead of precomputing these consequences for every possible sub-ontology, our approach computes just one label for each consequence such that a comparison of the user label with the consequence label determines whether the consequence follows from the corresponding sub-ontology or not.

In this paper we determine under which restrictions on the user and axiom labels such consequence labels (called boundaries) always exist, describe different black-box approaches for computing boundaries, and present first experimental results that compare the efficiency of these approaches on large real-world ontologies. Black-box means that, rather than requiring modifications of existing reasoning procedures, these approaches can use such procedures directly as sub-procedures, which allows us to employ existing highly-optimized reasoners.

1 Introduction

Assume that you have a large ontology $\mathcal{T}$, but you want to offer different users different views on this ontology, i.e., each user can see only a subset of the actual ontology, which is selected by an appropriate criterion. This criterion could be the access right that this user has, the level of trust (in the axioms of the ontology) that the user requires, the level of details that is deemed to be appropriate for this user, etc. In principle, you could explicitly create a sub-ontology for each (type of) user, but then you might end up with exponentially many different

A. Bernstein et al. (Eds.): ISWC 2009, LNCS 5823, pp. 49–64, 2009.

ontologies, where each is a subset of $\mathcal{T}$. Instead, we propose to keep just the big ontology $\mathcal{T}$, but label the axioms in $\mathcal{T}$ such that a comparison of the axiom label with the user criterion determines whether the axiom belongs to the sub-ontology for this user or not. To be more precise, we use a labeling lattice $(L, \leq)$, i.e., a set of labels L together with a partial order $\leq$ on these labels such that a finite set of labels always has a join (supremum, least upper bound) and a meet (infimum, greatest lower bound) w.r.t. $\leq$.[1]. All axioms $t \in \mathcal{T}$ are now assumed to have a label $\mathsf{lab}(t) \in L$, and the user also receives a label $\ell \in L$ (which can be read as access right, required level of trust, etc.). The sub-ontology that a user with label ℓ can see is then defined to be[2]

$$\mathcal{T}_\ell := \{t \in \mathcal{T} \mid \mathsf{lab}(t) \geq \ell\}.$$

Of course, the user of an ontology should not only be able to see its axioms, but also the consequences of these axioms. Thus, a user with label ℓ should be able to see all the consequences of $\mathcal{T}_\ell$. For large ontologies, certain relevant consequences are often pre-computed. The goal of the pre-computation is that certain user queries can be answered by a simple look-up in the pre-computed consequences, and thus do not require expensive reasoning during the deployment phase of the ontology. For example, in the version of the large medical ontology SNOMED CT[3] that is distributed to hospitals, all the subsumption relationships between the concept names occurring in the ontology are pre-computed. For a labeled ontology as introduced above, it is not enough to pre-compute the relevant consequences of $\mathcal{T}$. In fact, if the relevant consequence α follows from $\mathcal{T}$, then we also need to know for which user labels ℓ it still follows from $\mathcal{T}_\ell$. Otherwise, if a user with label ℓ asks whether α holds, the system could not simply look this up in the pre-computed consequences, but would need to compute the answer on-the-fly by reasoning over the sub-ontology $\mathcal{T}_\ell$. Our solution to this problem is to compute a so-called *boundary* for the consequence α, i.e., an element μ_α of L such that α follows from T_ℓ iff $\ell \leq \mu_\alpha$.

There are basically two approaches for computing a boundary. The *glass-box approach* takes a specific reasoner (or reasoning technique) for an ontology language (e.g., a tableau-based reasoner for OWL DL [20]) and modifies it such that it can compute a boundary. Examples for the application of the glass-box approach to specific instances of the problem of computing a boundary are tableau-based approaches for reasoning in possibilistic Description Logics [15,13] (where the lattice is the interval $[0, 1]$ with the usual order) and glass-box approaches to axiom pinpointing in Description Logics [19,14,12,3,4] (where the lattice consists of (equivalence classes of) monotone Boolean formulae with implication as order [4]). The problem with glass-box approaches is that they

[1] Figure 1 in Section 3 shows a small lattice. A detailed introduction to lattices and orders can, e.g., be found in [9].

[2] To define this sub-ontology, an arbitrary partial order would be sufficient. However, the existence of suprema and infima will be important for the computation of a boundary of a consequence (see below).

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

have to be developed for every ontology language and reasoning technique anew and that optimizations of the original reasoning technique do not always apply to the modified reasoners. In contrast, the *black-box approach* can re-use existing optimized reasoners without modifications, and it can be applied to arbitrary ontology languages: one just needs to plug in a reasoner for this language.

In this paper, we introduce three different black-box approaches for computing a boundary, and compare their performance on real-world ontologies. The first approach uses an axiom pinpointing algorithm as black-box reasoner, whereas the second one modifies the Hitting-Set-Tree-based black-box approach to axiom pinpointing [11,21]. The third uses binary search and can only be applied if the labeling lattice is a linear order. It can be seen as a generalization of the black-box approach to reasoning in possibilistic Description Logics described in [16]. All the proofs omitted in this paper can be found in [2].

2 Basic Definitions and Results

To stay as general as possible, we do not fix a specific ontology language. We just assume that *ontologies* are finite sets of *axioms* such that every subset of an ontology is again an ontology. If $\mathcal{T}'$ is a subset of the ontology $\mathcal{T}$, then $\mathcal{T}'$ is called a *sub-ontology* of $\mathcal{T}$. The ontology language determines which sets of axioms are admissible as ontologies. For a fixed ontology language, a *monotone consequence relation* $\models$ is a binary relation between ontologies $\mathcal{T}$ of this language and *consequences* α such that, for every ontology $\mathcal{T}$, we have that $\mathcal{T}' \subseteq \mathcal{T}$ and $\mathcal{T}' \models \alpha$ imply $\mathcal{T} \models \alpha$. If $\mathcal{T} \models \alpha$, then we say that α *follows from* $\mathcal{T}$ and that $\mathcal{T}$ *entails* α. For instance, given a Description Logic $\mathcal{L}$ (e.g., the DL $\mathcal{SHIN}(D)$ underlying OWL DL), an ontology is an $\mathcal{L}$-TBox, i.e., a finite set of general concept inclusion axioms (GCIs) of the form $C \sqsubseteq D$ for $\mathcal{L}$-concept descriptions C, D. As consequences we can, e.g., consider subsumption relationships $A \sqsubseteq B$ for concept names A, B.

We consider a lattice $(L, \leq)$ and respectively denote by $\bigoplus_{\ell \in S} \ell$ and $\bigotimes_{\ell \in S} \ell$ the *join* (least upper bound) and *meet* (greatest lower bound) of the finite set $S \subseteq L$. A *labeled ontology with labeling lattice* $(L, \leq)$ is an ontology $\mathcal{T}$ together with a labeling function lab that assigns a *label* $\mathsf{lab}(t) \in L$ to every element t of $\mathcal{T}$[4] We denote with L_{lab} the set of all labels occurring in the labeled ontology $\mathcal{T}$, i.e., $L_{\mathsf{lab}} := \{\mathsf{lab}(t) \mid t \in \mathcal{T}\}$. Every element of the labeling lattice $\ell \in L$ defines a sub-ontology $\mathcal{T}_\ell$ that contains the axioms of $\mathcal{T}$ that are labeled with elements greater than or equal to ℓ:

$$\mathcal{T}_\ell := \{t \in \mathcal{T} \mid \mathsf{lab}(t) \geq \ell\}.$$

Conversely, every sub-ontology $\mathcal{S} \subseteq \mathcal{T}$ defines an element $\lambda_\mathcal{S} \in L$, called the *label* of $\mathcal{S}$: $\lambda_\mathcal{S} := \bigotimes_{t \in \mathcal{S}} \mathsf{lab}(t)$. The following lemma states some simple relationships between these two notions.

Lemma 1. *For all $\ell \in L$, $\mathcal{S} \subseteq \mathcal{T}$, it holds that $\ell \leq \lambda_{\mathcal{T}_\ell}$, $\mathcal{S} \subseteq \mathcal{T}_{\lambda_\mathcal{S}}$ and $\mathcal{T}_\ell = \mathcal{T}_{\lambda_{\mathcal{T}_\ell}}$.*

[4] An example of a labeled ontology is given in Example 2 in Section 3.

Notice that, if a consequence α follows from $\mathcal{T}_\ell$ for some $\ell \in L$, it must also follow from $\mathcal{T}_{\ell'}$ for every $\ell' \leq \ell$, since then $\mathcal{T}_\ell \subseteq \mathcal{T}_{\ell'}$. A maximal element of L that still entails the consequence will be called a margin for this consequence.

Definition 1 (Margin). *Let α be a consequence that follows from the ontology $\mathcal{T}$. The label $\mu \in L$ is called a* $(\mathcal{T}, \alpha)$-margin *if $\mathcal{T}_\mu \models \alpha$, and for every ℓ with $\mu < \ell$ we have $\mathcal{T}_\ell \not\models \alpha$.*

If $\mathcal{T}$ and α are clear from the context, we usually ignore the prefix $(\mathcal{T}, \alpha)$ and call μ simply a *margin*. The following lemma shows three basic properties of the set of margins that will be useful throughout this paper.

Lemma 2. *Let α be a consequence that follows from the ontology $\mathcal{T}$. We have:*

1. *If μ is a margin, then $\mu = \lambda_{\mathcal{T}_\mu}$;*
2. *if $\mathcal{T}_\ell \models \alpha$, then there is a margin μ such that $\ell \leq \mu$;*
3. *there are at most $2^{|\mathcal{T}|}$ margins for α.*

If we know that μ is a margin for the consequence α, then we know whether α follows from $\mathcal{T}_\ell$ for all $\ell \in L$ that are comparable with μ: if $\ell \leq \mu$, then α follows from $\mathcal{T}_\ell$, and if $\ell > \mu$, then α does not follow from $\mathcal{T}_\ell$. However, the fact that μ is a margin gives us no information regarding elements that are incomparable with μ. In order to obtain a full picture of when the consequence α follows from $\mathcal{T}_\ell$ for an arbitrary element of l, we can try to strengthen the notion of margin to that of an element ν of L that accurately divides the lattice into those elements whose associated sub-ontology entails α and those for which this is not the case, i.e., ν should satisfy the following: for every $\ell \in L$, $\mathcal{T}_\ell \models \alpha$ iff $\ell \leq \nu$. Unfortunately, such an element need not always exist, as demonstrated by the following example.

Example 1. Consider the distributive lattice $(S_4, \leq_4)$ having the four elements $S_4 = \{0, a_1, a_2, 1\}$, where 0 and 1 are the least and greatest elements, respectively, and a_1, a_2 are incomparable w.r.t. $\leq_4$. Let $\mathcal{T}$ be the set formed by the axioms ax_1 and ax_2, which are labeled by a_1 and a_2, respectively, and let α be a consequence such that, for every $\mathcal{S} \subseteq \mathcal{T}$, we have $\mathcal{S} \models \alpha$ iff $|\mathcal{S}| \geq 1$. It is easy to see that there is no element $\nu \in S_4$ that satisfies the condition described above. Indeed, if we choose $\nu = 0$ or $\nu = a_1$, then a_2 violates the condition, as $a_2 \not\leq \nu$, but $\mathcal{T}_{a_2} = \{\mathrm{ax}_2\} \models \alpha$. Accordingly, if we choose $\nu = a_2$, then a_1 violates the condition. Finally, if $\nu = 1$ is chosen, then 1 itself violates the condition: $1 \leq \nu$, but $\mathcal{T}_1 = \emptyset \not\models \alpha$.

It is nonetheless possible to find an element that satisfies a restricted version of the condition, where we do not impose that the property must hold for every element of the labeling lattice, but only for those elements that are *join prime* relative to the labels of the axioms in the ontology.

Definition 2 (Join prime). *Let $(L, \leq)$ be a lattice. Given a finite set $K \subseteq L$, let $K_\otimes := \{\bigotimes_{\ell \in M} \ell \mid M \subseteq K\}$ denote the closure of K under the meet operator. An element $\ell \in L$ is called* join prime relative to *K if, for every $K' \subseteq K_\otimes$, $\ell \leq \bigoplus_{k \in K'} k$ implies that there is an $k_0 \in K'$ such that $\ell \leq k_0$.*

In Example 1, all lattice elements with the exception of 1 are join prime relative to $\{a_1, a_2\}$.

Definition 3 (Boundary). *Let $\mathcal{T}$ be an ontology and α a consequence. An element $\nu \in L$ is called a* $(\mathcal{T}, \alpha)$-boundary *if for every element $\ell \in L$ that is join prime relative to L_{lab} it holds that $\ell \leq \nu$ iff $\mathcal{T}_\ell \models \alpha$.*

As with margins, if $\mathcal{T}$ and α are clear from the context, we will simply call such a ν a *boundary*. In Example 1, the element 1 is a boundary. Indeed, every join prime element ℓ relative to $\{a_1, a_2\}$ (i.e., every element of L except for 1) is such that $\ell < 1$ and $\mathcal{T}_\ell \models \alpha$. From a practical point of view, our definition of a boundary has the following implication: we must enforce that user labels are always join prime relative to the set L_{lab} of all labels occurring in the ontology.

3 Computing a Boundary

In this section, we describe three black-box approaches for computing a boundary. The first two approaches are based on Lemma 3 below, and the third one, a modification of binary search, can be used if the labeling lattice is a linear order.

Lemma 3. *Let $\mu_1, \ldots, \mu_n$ be all $(\mathcal{T}, \alpha)$-margins. Then $\bigoplus_{i=1}^{n} \mu_i$ is a boundary.*

By Lemma 2, a consequence always has finitely many margins, and thus Lemma 3 shows that a boundary always exists. Note, however, that a consequence may have boundaries different from the one of Lemma 3. To identify the particular boundary of Lemma 3, we will call it the *margin-based boundary*.

3.1 Using Full Axiom Pinpointing

From Lemma 3 we know that the set of all margins yields sufficient information for computing a boundary. The question is now how to compute this set. In this subsection, we show that all margins (and thus the margin-based boundary) can be computed through *axiom pinpointing*. Axiom-pinpointing refers to the task of computing *MinAs* [6]: minimal (w.r.t. set inclusion) sub-ontologies from which a consequence α still follows. More formally, $\mathcal{S} \subseteq \mathcal{T}$ is called a *MinA* for $\mathcal{T}$ and α if $\mathcal{S} \models \alpha$, and $\mathcal{S}' \not\models \alpha$ for every $\mathcal{S}' \subset \mathcal{S}$. The following lemma shows that every margin can be obtained from some MinA.

Lemma 4. *For every margin μ for α there is a MinA $\mathcal{S}$ such that $\mu = \lambda_{\mathcal{S}}$.*

Notice that this lemma does not imply that the label of any MinA $\mathcal{S}$ corresponds to a margin. However, as the consequence follows from every MinA, point 2 of Lemma 2 shows that $\lambda_{\mathcal{S}} \leq \mu$ for some margin μ. The following theorem is an immediate consequence of this fact together with Lemma 3 and Lemma 4.

Theorem 1. *If $\mathcal{S}_1, \ldots, \mathcal{S}_n$ are all MinAs for $\mathcal{T}$ and α, then $\bigoplus_{i=1}^{n} \lambda_{\mathcal{S}_i}$ is the margin-based boundary for α.*

Thus, to compute a boundary, it is sufficient to compute all MinAs. Several methods exist for computing the set of all MinAs, either directly [19,11,7] or through a so-called pinpointing formula [6,4,5], which is a monotone Boolean formula encoding all the MinAs. The main advantage of using the pinpointing-based approach for computing a boundary is that one can simply use existing implementations for computing all MinAs, such as the ones offered by the ontology editor Protégé 4[5] and the CEL system.[6]

3.2 Label-Optimized Axiom Pinpointing

From Lemma 4 we know that every margin is of the form $\lambda_{\mathcal{S}}$ for some MinA $\mathcal{S}$. In the previous subsection we have used this fact to compute a boundary by first obtaining the MinAs and then computing their labels. This process can be optimized if we directly compute the labels of the MinAs, without necessarily computing the actual MinAs. Additionally, not all the labels of MinAs are necessary, but only the maximal ones. We present here a black-box algorithm that uses the labels of the axioms to find the boundary in an optimized way. Our algorithm is a variant of the Hitting-Set-Tree-based [17] method (HST approach) for axiom pinpointing [11,21]. First, we briefly describe the HST approach for computing all MinAs, which will serve as a starting point for our modified version.

The HST algorithm computes one MinA at a time while building a tree that expresses the distinct possibilities to be explored in the search of further MinAs. It first computes an arbitrary MinA $\mathcal{S}_0$ for $\mathcal{T}$, which is used to label the root of the tree. Then, for every axiom t in $\mathcal{S}_0$, a successor node is created. If $\mathcal{T} \setminus \{t\}$ does not entail the consequence, then this node is a dead end. Otherwise, $\mathcal{T} \setminus \{t\}$ still entails the consequence. In this case, a MinA $\mathcal{S}_1$ for $\mathcal{T} \setminus \{t\}$ is computed and used to label the node. The MinA $\mathcal{S}_1$ for $\mathcal{T} \setminus \{t\}$ obtained this way is also a MinA of $\mathcal{T}$, and it is guaranteed to be distinct from $\mathcal{S}_0$ since $t \notin \mathcal{S}_1$. Then, for each axiom s in $\mathcal{S}_1$, a new successor is created, and treated in the same way as the successors of the root node, i.e., it is checked whether $\mathcal{T} \setminus \{t, s\}$ still has the consequence, etc. This process obviously terminates, and the end result is a tree, where each node that is not a dead end is labeled with a MinA, and every MinA appears as the label of at least one node of the tree (see [11,21]).

An important ingredient of the HST algorithm is a procedure that computes a single MinA from an ontology. Such a procedure can, for example, be obtained by going through the axioms of the ontology in an arbitrary order, and removing redundant axioms, i.e., ones such that the ontology obtained by removing this axiom from the current sub-ontology still entails the consequence (see [6] for a description of this and of a more sophisticated logarithmic procedure). As said before, in our modified HST algorithm, we are now not interested in actually computing a MinA, but only its label. This allows us to remove all axioms having a "redundant" label rather than a single axiom. Algorithm 1 describes a black-box method for computing $\lambda_{\mathcal{S}}$ for some MinA $\mathcal{S}$ that is based on this idea. In

[5] http://protege.stanford.edu/
[6] http://code.google.com/p/cel/

Algorithm 1. Compute a minimal label set of one MinA.

Procedure min-lab($\mathcal{T}, \alpha$)
Input: $\mathcal{T}$: ontology; α: consequence
Output: $M_L \subseteq L$: minimal label set for a MinA

```
 1: if T ⊭ α then
 2:    return no MinA
 3: S := T
 4: M_L := ∅
 5: for every k ∈ L_lab do
 6:    if ⨂_{l∈M_L} l ⋦ k then
 7:       if S − k ⊨ α then
 8:          S := S − k
 9:       else
10:          M_L := (M_L \ {l | k < l}) ∪ {k}
11: return M_L
```

fact, the algorithm computes a *minimal label set* of a MinA $\mathcal{S}$, a notion that will also be useful when describing our variant of the HST algorithm.

Definition 4 (Minimal label set). *Let $\mathcal{S}$ be a MinA for α. A set $K \subseteq \{\mathsf{lab}(t) \mid t \in \mathcal{S}\}$ is called a* minimal label set *of $\mathcal{S}$ if distinct elements of K are incomparable and $\lambda_{\mathcal{S}} = \bigotimes_{\ell \in K} \ell$.*

Algorithm 1 removes all the labels that do not contribute to a minimal label set. If $\mathcal{T}$ is an ontology and $\ell \in L$, then the expression $\mathcal{T} - \ell$ appearing at Line 7 denotes the sub-ontology $\mathcal{T} - \ell := \{t \in \mathcal{T} \mid \mathsf{lab}(t) \neq \ell\}$. If, after removing all the axioms labeled with k, the consequence still follows, then there is a MinA none of whose axioms is labeled with k. In particular, this MinA has a minimal label set not containing k; thus all the axioms labeled with k can be removed in our search for a minimal label set. If the axioms labeled with k cannot be removed, then all MinAs of the current sub-ontology need an axiom labeled with k, and hence k is stored in the set M_L. This set is used to avoid useless consequence tests: if a label is greater than or equal to $\bigotimes_{\ell \in M_L} \ell$, then the presence or absence of axioms with this label will not influence the final result, which will be given by the infimum of M_L; hence, there is no need to apply the (possibly complex) decision procedure for the consequence relation.

Theorem 2. *Let $\mathcal{T}$ and α be such that $\mathcal{T} \models \alpha$. There is a MinA $\mathcal{S}_0$ for α such that Algorithm 1 outputs a minimal label set of $\mathcal{S}_0$.*

Once the label of a MinA has been found, we can compute new MinA labels by a successive deletion of axioms from the ontology using the HST approach. Suppose that we have computed a minimal label set $\mathcal{M}_0$, and that $\ell \in \mathcal{M}_0$. If we remove all the axioms in the ontology labeled with ℓ, and compute a new minimal label set $\mathcal{M}_1$ of a MinA of this sub-ontology, then $\mathcal{M}_1$ does not contain ℓ, and thus $\mathcal{M}_0 \neq \mathcal{M}_1$. By iterating this procedure, we could compute all minimal label sets, and hence the labels of all MinAs. However, since our goal is to compute

Algorithm 2. Hitting set tree (HST) algorithm for computing the boundary

Procedure hst-boundary($\mathcal{T}, \alpha$)
Input: $\mathcal{T}$: ontology; α: consequence
Output: boundary ν for α

1: **Global** : $\mathbf{C}, \mathbf{H} := \emptyset; \nu$
2: $\mathcal{M} :=$ min-lab($\mathcal{T}, \alpha$)
3: $\mathbf{C} := \{\mathcal{M}\}$
4: $\nu := \bigotimes_{\ell \in \mathcal{M}} \ell$
5: **for** each label $\ell \in \mathcal{M}$ **do**
6: expand-hst($\mathcal{T}_{\not\leq \ell}, \alpha, \{\ell\}$)
7: **return** ν

Procedure expand-hst($\mathcal{T}, \alpha, H$)
Input: $\mathcal{T}$: ontology; α: consequence; H: list of lattice elements
Side effects: modifications to $\mathbf{C}$, $\mathbf{H}$ and ν

1: **if** there exists some $H' \in \mathbf{H}$ such that $\{h \in H' \mid h \not\leq \nu\} \subseteq H$ **or**
H' contains a prefix-path P with $\{h \in P \mid h \not\leq \nu\} = H$ **then**
2: **return** (early path termination $\diamond$)
3: **if** there exists some $\mathcal{M} \in \mathbf{C}$ such that for all $\ell \in \mathcal{M}, h \in H$, $\ell \not\leq h$ and $\ell \not\leq \nu$ **then**
4: $\mathcal{M}' := \mathcal{M}$ (MinLab reuse)
5: **else**
6: $\mathcal{M}' :=$ min-lab($\mathcal{T}_{\not\leq \nu}, \alpha$)
7: **if** $\mathcal{T}_{\not\leq \nu} \models \alpha$ **then**
8: $\mathbf{C} := \mathbf{C} \cup \{\mathcal{M}'\}$
9: $\nu := \bigoplus\{\nu, \bigotimes_{\ell \in \mathcal{M}'} \ell\}$
10: **for** each label $\ell \in \mathcal{M}'$ **do**
11: expand-hst($\mathcal{T}_{\not\leq \ell}, \alpha, H \cup \{\ell\}$)
12: **else**
13: $\mathbf{H} := \mathbf{H} \cup \{H\}$ (normal termination $\odot$)

the supremum of these labels, the algorithm can be optimized by avoiding the computation of MinAs whose labels will have no impact on the final result. Based on this we can actually do better than just removing the axioms with label ℓ: instead, all axioms with labels $\leq \ell$ can be removed. For an element $\ell \in L$ and an ontology $\mathcal{T}$, $\mathcal{T}_{\not\leq \ell}$ denotes the sub-ontology obtained from $\mathcal{T}$ by removing all axioms whose labels are $\leq \ell$. Now, assume that we have computed the minimal label set $\mathcal{M}_0$, and that $\mathcal{M}_1 \neq \mathcal{M}_0$ is the minimal label set of the MinA $\mathcal{S}_1$. For all $\ell \in \mathcal{M}_0$, if $\mathcal{S}_1$ is not contained in $\mathcal{T}_{\not\leq \ell}$, then $\mathcal{S}_1$ contains an axiom with label $\leq \ell$. Consequently, $\bigotimes_{m \in \mathcal{M}_1} m = \lambda_{\mathcal{S}_1} \leq \bigotimes_{m \in \mathcal{M}_0} m$, and thus $\mathcal{M}_1$ need not be computed. Algorithm 2 describes our method for computing the boundary using a variant of the HST algorithm that is based on this idea.

In the procedure hst-boundary, three global variables are declared: $\mathbf{C}$ and $\mathbf{H}$, initialized with $\emptyset$, and ν. The variable $\mathbf{C}$ stores all the minimal label sets computed so far, while each element of $\mathbf{H}$ is a set of labels such that, when all the axioms with a label less than or equal to any label from the set are removed from the ontology, the consequence does not follow anymore; the variable ν stores the supremum of the labels of all the elements in $\mathbf{C}$ and ultimately corresponds

to the boundary that the method computes. The algorithm starts by computing a first minimal label set $\mathcal{M}$, which is used to label the root of a tree. For each element of $\mathcal{M}$, a branch is created by calling the procedure expand-hst.

The procedure expand-hst implements the ideas of HST construction for pinpointing [11,21] with additional optimizations that help reduce the search space as well as the number of calls to min-lab. First notice that each $\mathcal{M} \in \mathbf{C}$ is a minimal label set, and hence the infimum of its elements corresponds to the label of some MinA for α. Thus, ν is the supremum of the labels of a set of MinAs for α. If this is not yet the boundary, then there must exist another MinA $\mathcal{S}$ whose label is not less than or equal to ν. This in particular means that no element of $\mathcal{S}$ may have a label less than or equal to ν, as the label of $\mathcal{S}$ is the infimum of the labels of the axioms in it. When searching for this new MinA we can then exclude all axioms having a label $\leq \nu$, as done in Line 6 of expand-hst. Every time we expand a node, we extend the set H, which stores the labels that have been removed on the path in the tree to reach the current node. If we reach normal termination, it means that the consequence does not follow anymore from the reduced ontology. Thus, any H stored in $\mathbf{H}$ is such that, if all the axioms having a label less than or equal to an element in H are removed from $\mathcal{T}$, then α does not follow anymore. Lines 1 to 4 of expand-hst are used to reduce the number of calls to the subroutine min-lab and the total search space. We describe them now in more detail. The first optimization, *early path termination*, prunes the tree once we know that no new information can be obtained from further expansion. There are two conditions that trigger this optimization. The first one tries to decide whether $\mathcal{T}_{\not\leq\nu} \models \alpha$ without executing the decision procedure. As said before, we know that for each $H' \in \mathbf{H}$, if all labels less than or equal to any in H' are removed, then the consequence does not follow. Hence, if the current list of removal labels H contains a set $H' \in \mathbf{H}$ we know that enough labels have been removed to make sure that the consequence does not follow. It is actually enough to test whether $\{h \in H' \mid h \not\leq \nu\} \subseteq H$ since the consequence test we need to perform is whether $\mathcal{T}_{\not\leq\nu} \models \alpha$. The second condition for early path termination asks for a prefix-path P of H' such that $P = H$. If we consider H' as a list of elements, then a prefix-path is obtained by removing a final portion of this list. The idea is that, if at some point we have noticed that we have removed the same axioms as in a previous portion of the search, we know that all possibilities that arise from that search have already been tested before, and hence it is unnecessary to repeat the work. Hence we can prune the tree here.

The second optimization avoids a call to min-lab by *reusing* a previously computed minimal label set. Notice that our only requirement on min-lab that it produces a minimal label set. Hence, any minimal label set for the ontology obtained after removing all labels less than or equal to any $h \in H$ or to ν would work. The MinLab reuse optimization checks whether there is such a previously computed minimal label set. If this is the case, it uses this set instead of computing a new one by calling min-lab.

Theorem 3. *Let $\mathcal{T}$ and α be such that $\mathcal{T} \models \alpha$. Then Algorithm 2 computes the margin-based boundary of α.*

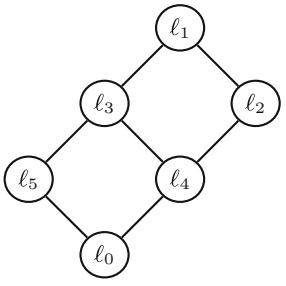

Fig. 1. A lattice

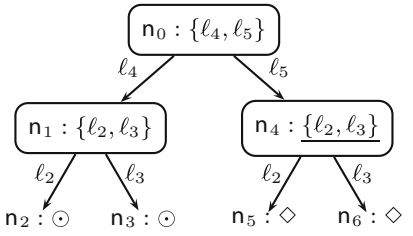

Fig. 2. An expansion of the HST method

A proof of this theorem can be found in [2]. Here, we just illustrate how it works by a small example.

Example 2. Consider the lattice in Figure 1, and let $\mathcal{T}$ be the (Description Logic) ontology consisting of the following five axioms:

$$t_1 : A \sqsubseteq P_1 \sqcap Q_1, \qquad t_2 : P_1 \sqsubseteq P_2 \sqcap Q_2, \qquad t_3 : P_2 \sqsubseteq B,$$
$$t_4 : Q_1 \sqsubseteq P_2 \sqcap Q_2, \qquad t_5 : Q_2 \sqsubseteq B,$$

where each axiom t_i is labeled with $\mathsf{lab}(t_i) = \ell_i$. There are four MinAs for the subsumption relation $A \sqsubseteq B$ w.r.t. $\mathcal{T}$, namely $\{t_1, t_2, t_3\}, \{t_1, t_2, t_5\}, \{t_1, t_3, t_4\}$, and $\{t_1, t_4, t_5\}$. All the elements of the labeling lattice except ℓ_1 and ℓ_3 are join prime relative to L_{lab}. Figure 2 shows a possible run of the hst-boundary algorithm. The algorithm first calls the routine min-lab$(\mathcal{T}, A \sqsubseteq B)$. Consider that the **for** loop of min-lab is executed using the labels $\ell_1, \ldots, \ell_5$ in that order. Thus, we try first to remove t_1 labeled with ℓ_1. We see that $\mathcal{T} - \ell_1 \not\models A \sqsubseteq B$; hence t_1 is not removed from $\mathcal{T}$, and M_L is updated to $M_L = \{\ell_1\}$. We then see that $\mathcal{T} - \ell_2 \models A \sqsubseteq B$, and thus t_2 is removed from $\mathcal{T}$. Again, $\mathcal{T} - \ell_3 \models A \sqsubseteq B$, so t_3 is removed from $\mathcal{T}$. At this point, $\mathcal{T} = \{t_1, t_4, t_5\}$. We test then whether $\mathcal{T} - \ell_4 \models A \sqsubseteq B$ and receive a negative answer; thus, ℓ_4 is added to M_L; additionally, since $\ell_4 < \ell_1$, the latter is removed from M_L. Finally, $\mathcal{T} - \ell_5 \not\models A \sqsubseteq B$, and so we obtain $M_L = \{\ell_4, \ell_5\}$ as an output of min-lab.

The minimal label set $\{\ell_4, \ell_5\}$, is used as the root node n_0, setting the value of $\nu = \ell_4 \otimes \ell_5 = \ell_0$. We then create the first branch on the left by removing all the axioms with a label $\leq \ell_4$, which is only t_4, and computing a new minimal label set. Assume, for the sake of the example, that min-lab returns the minimal label set $\{\ell_2, \ell_3\}$, and ν is accordingly changed to ℓ_4. When we expand the tree from this node, by removing all the axioms below ℓ_2 (left branch) or ℓ_3 (right branch), the subsumption relation $A \sqsubseteq B$ does not follow any more, and hence we have a normal termination, adding the sets $\{\ell_4, \ell_2\}$ and $\{\ell_4, \ell_3\}$ to $\mathbf{H}$. We then create the second branch from the root, by removing the elements below ℓ_5. We see that the previously computed minimal axiom set of node n_1 works also as a minimal axiom set in this case, and hence it can be reused (MinLab reuse), represented as an underlined set. The algorithm continues now by calling expand-hst$(\mathcal{T}_{\not\leq \ell_2}, A \sqsubseteq B, \{\ell_5, \ell_2\})$. At this point, we detect that there is $H' = \{\ell_4, \ell_2\}$

Algorithm 3. Compute a boundary by binary search.

Input: $\mathcal{T}$: ontology; α: consequence
Output: ν: $(\mathcal{T}, \alpha)$-boundary

```
1: if T ⊭ α then
2:    return no boundary
3: ℓ := 0_lab; h := 1_lab
4: while l < h do
5:    set m, ℓ < m ≤ h such that δ(ℓ, m) − δ(m, h) ≤ 1.
6:    if T_m ⊨ α then
7:       ℓ := m
8:    else
9:       h := pred(m)
10: return ν := ℓ
```

satisfying the first condition of early path termination (recall that $\nu = \ell_4$), and hence the expansion of that branch at that point. Analogously, we obtain an early path termination on the second expansion branch of the node n_4. The algorithm then outputs $\nu = \ell_4$, which can be easily verified to be a boundary.

3.3 Binary Search for Linear Ordering

In this subsection, we assume that the labeling lattice $(L, \leq)$ is a linear order, i.e., for any two elements ℓ_1, ℓ_2 of L we have $\ell_1 \leq \ell_2$ or $\ell_2 \leq \ell_1$.

Lemma 5. *Let $\mathcal{T}$ and α be such that $\mathcal{T} \models \alpha$. Then the unique boundary of α is the maximal element μ of L_{lab} with $\mathcal{T}_\mu \models \alpha$.*

A direct way for computing the boundary in this restricted setting thus consists of testing, for every element in $\ell \in L_{\mathsf{lab}}$, in order (either increasing or decreasing) whether $\mathcal{T}_\ell \models \alpha$ until the desired maximal element is found. This process requires in the worst case $n := |L_{\mathsf{lab}}|$ iterations. This can be improved using binary search, which requires a logarithmic number of steps measured in n. Algorithm 3 describes the binary search algorithm. In the description of the algorithm, the following abbreviations have been used: $\mathbf{0}_{\mathsf{lab}}$ and $\mathbf{1}_{\mathsf{lab}}$ represent the minimal and the maximal elements of L_{lab}, respectively; for $\ell_1 \leq \ell_2 \in L_{\mathsf{lab}}$, $\delta(\ell_1, \ell_2) := |\{\ell' \in L_{\mathsf{lab}} \mid \ell_1 < \ell' \leq \ell_2\}|$ is the *distance* function in L_{lab} and for a given $\ell \in L_{\mathsf{lab}}$, $\mathsf{pred}(\ell)$ is the maximal element $\ell' \in L_{\mathsf{lab}}$ such that $\ell' < \ell$.

The variables ℓ and h are used to keep track of the relevant search space. At every iteration of the **while** loop, the boundary is between ℓ and h. At the beginning these values are set to the minimum and maximum of L_{lab} and are later modified as follows: we first find the *middle* element m of the search space; i.e., an element whose distance to ℓ differs by at most one from the distance to h. We then test whether $\mathcal{T}_m \models \alpha$. If that is the case, we know that the boundary must be larger or equal to m, and hence the lower bound ℓ is updated to the value of m. Otherwise, we know that the boundary is strictly smaller than m as m itself cannot be one; hence, the higher bound h is updated to the maximal

element of L_{lab} that is smaller than $m : \mathsf{pred}(m)$. This process terminates when the search space has been reduced to a single point, which must be the boundary.

4 Empirical Evaluation

4.1 Test Data and Test Environment

We test on a PC with 2GB RAM and Intel Core Duo CPU 3.16GHz. We implemented all approaches in Java and used Java 1.6, CEL 1.0, Pellet 2.0.0-rc5 and OWL API trunk revision 1150. The boundary computation with full axiom pinpointing (FP in the following) uses log-extract-mina() (Alg. 2 from [7], which is identical to Alg. 8 from [21]) and the HST based hst-extract-all-minas() (Alg. 9 from [21]). The set of extracted MinAs is then used to calculate the label of the consequence. We break after 10 found MinAs in order to limit the runtime, so there might be non-final label results. The boundary computation with label-optimized axiom pinpointing (LP in the following) with min-lab() and hst-boundary() are implementations of Alg. 1 and Alg. 2 of the present paper. The boundary computation with binary search for linear ordering (BS in the following) implements Alg. 3 of the present paper.

Although we focus on comparing the efficiency of the presented algorithms, and not on practical applications of these algorithms, we have tried to use inputs that are closely related to ones encountered in applications. The two labeling lattices $(L_d, \leq_d)$ and $(L_l, \leq_l)$ are similar to ones encountered in real-world applications. The labeling lattice $(L_d, \leq_d)$ was already introduced in Fig. 1. Lattices of this structure (where the elements correspond to hierarchically organized user roles) can be obtained from a real-world access matrix with the methodology presented in [8]. The set of elements of L_d that are allowed to represent user roles if all elements of the lattice can be used as axiom labels are the elements that are join prime relative to the whole lattice, i.e., $\ell_0, \ell_2, \ell_4, \ell_5$. The labeling lattice $(L_l, \leq_l)$ is a linear order with 6 elements $L_l = L_d = \{\ell_0, \ldots, \ell_5\}$ with $\leq_l := \{(\ell_n, \ell_{n+1}) \mid \ell_n, \ell_{n+1} \in L_l \wedge 0 \leq n \leq 5\}$, which could represent an order of trust values as in [18] or dates from a revision history.

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) $\mathcal{EL}+$. Our version O^{Snomed} is the January/2005 release of the DL version, which contains 379,691 concept names, 62 object property names, and 379,704 axioms. Since more than five million subsumptions are consequences of O^{Snomed}, testing all of them was not feasible and we used the same sample subset as described in [7], i.e., we sampled 0.5% of all concepts in each top-level category of O^{Snomed}. For each sampled concept A, all positive subsumptions $A \sqsubseteq_{O^{\text{Snomed}}} B$ with A as subsumee were considered. Overall, this yielded 27,477 positive subsumptions. Following the ideas of [7], we precomputed the reachability-based module for each sampled concept A with CEL and stored these modules. This module for A was then used as the start ontology when considering subsumptions with subsumee A.

Table 1. Emprical results of FP and LP with lattice $(L_d, \leq_d)$ on a sampled set of 21,001 subsumptions from O^{SNOMED} and on a set of 307 consequences from O^{FUNCT} with less than 10 MinAs (time in ms)

			♯early termination	♯reuse	♯calls to extract MinA (MinLab)	♯MinA (♯MinLab)	♯axioms (♯labels) per MinA (MinLab)	lattice operations time	total labeling time
O^{SNOMED}	FP	avg	81.05	9.06	26.43	2.07	5.40	0.25	143.55
		max	57,188.00	4,850.00	4,567.00	9.00	28.67	45.00	101,616.00
		stddev	874.34	82.00	90.48	1.86	3.80	0.86	1,754.03
	LP	avg	0.01	0.00	2.76	1.03	1.73	0.35	4.29
		max	2.00	1.00	6.00	3.00	3.00	57.00	70.00
		stddev	0.13	0.02	0.59	0.16	0.56	0.98	3.62
O^{FUNCT}	FP	avg	43.59	29.52	26.56	4.26	3.05	0.49	3,403.56
		max	567.00	433.00	126.00	9.00	6.50	41.00	13,431.00
		stddev	92.16	64.04	30.90	2.84	1.01	2.38	3,254.25
	LP	avg	0.09	0.02	2.80	1.33	1.40	0.76	207.32
		max	2.00	1.00	7.00	4.00	3.00	22.00	1,295.00
		stddev	0.34	0.13	0.90	0.54	0.48	1.56	87.29

O^{FUNCT} is an OWL ontology for functional description of mechanical engineering solutions presented in [10]. It has 115 concept names, 47 object property names, 16 data property names, 545 individual names, 3,176 axioms, and the DL expressivity used in the ontology is $\mathcal{SHOIN}(\mathbf{D})$. Its 716 consequences are 12 subsumption and 704 instance relationships (class assertions).

To obtain labeled ontologies, axioms in both labeled ontologies received a random label assignment of elements from $L_l = L_d$. As black-box subsumption and instance reasoner we used the reasoner Pellet since it can deal with the expressivity of both ontologies. For the expressive DL $\mathcal{SHOIN}(\mathbf{D})$ it uses a tableau-based algorithm and for $\mathcal{EL}+$ it uses an optimized classifier for the OWL 2 EL profile, which is based on the algorithm described in [1].

4.2 Results

The results for O^{SNOMED} and $(L_d, \leq_d)$ are given in the upper part of Table 1. LP computed all labels, but since we limit FP to <10 MinAs, only 21,001 subsumptions have a final label, which is guaranteed to be equal to the boundary. The 6,476 remaining subsumptions (31%) have a non-final label which might be too low in the lattice since there might be further MinAs providing a higher label. The overall labeling time for all 21,001 subsumptions with FP was 50.25 minutes, for LP 1.50 minutes which means that LP is about 34 times faster than FP, but again this is only for the subset of subsumptions which were finished by FP. An estimation for the time needed to label all of the more than 5 million subsumptions in O^{SNOMED} with LP would be approximately 6 hours.

The final labels of FP and LP (i.e., the computed boundaries) were identical, the non-final labels of FP were identical to the final labels of LP (i.e., the boundaries) in 6,376 of the 6,476 cases (98%), i.e., in most cases the missing MinAs would not have changed the already computed label. Table 2 provides results for the subsumptions with more than 10 MinAs: FP took 2.5 hours on this set without final results (since it stopped after 10 MinAs), whereas LP took 0.6% of that time and returned final results after 58 seconds. We started a test series

Table 2. Emprical results of FP and LP with lattice $(L_d, \leq_d)$ on a sampled set of 6,476 subsumptions from O^{SNOMED} and on a set of 409 class assertions from O^{FUNCT} with at least 10 MinAs (time in ms)

			♯early termination	♯reuse	♯calls to extract MinA (MinLab)	♯MinA (♯MinLab)	♯axioms (♯labels) per MinA (MinLab)	lattice operations time	total (non-final) labeling time
O^{SNOMED}	FP	avg	432.11	42.25	126.54	10.20	16.38	0.30	1,378.66
		max	42,963.00	5,003.00	4,623.00	16.00	37.80	14.00	148,119.00
		stddev	1,125.06	121.15	186.33	0.49	5.00	0.54	3,493.02
	LP	avg	0.04	0.00	3.12	1.06	2.05	0.32	8.88
		max	3.00	2.00	6.00	3.00	3.00	46.00	86.00
		stddev	0.21	0.04	0.50	0.25	0.44	1.04	4.26
O^{FUNCT}	FP	avg	30.01	16.00	26.44	10.04	4.41	0.56	8,214.91
		max	760.00	511.00	411.00	11.00	6.50	3.00	25,148.00
		stddev	85.33	47.79	40.61	0.20	1.08	0.55	3,428.97
	LP	avg	0.09	0.01	2.76	1.38	1.32	0.77	200.55
		max	3.00	2.00	7.00	4.00	2.00	16.00	596.00
		stddev	0.33	0.12	0.91	0.64	0.43	1.40	61.11

Table 3. Emprical results of LP and BS on a sampled set of 27,477 subsumptions in O^{SNOMED}/ all 716 consequences of O^{FUNCT} with lattice $(L_l, \leq_l)$ (time in ms)

		LP							BS	
		♯early termination	♯reuse	♯calls to extract MinLab	♯MinLab	♯labels per MinLab	lattice operations time	total labeling time	iterations	total labeling time
O^{SNOMED}	avg	0.03	0.00	2.24	1.03	1.23	0.37	4.75	2.41	2.81
	max	1.00	0.00	5.00	3.00	2.00	329.00	330.00	3.00	75.00
	stddev	0.18	0.00	0.45	0.19	0.42	4.85	6.37	0.49	2.94
O^{FUNCT}	avg	0.09	0.00	2.50	1.27	1.24	0.82	186.98	2.55	95.80
	max	1.00	0.00	5.00	3.00	2.00	62.00	1147.00	3.00	877.00
	stddev	0.28	0.00	0.72	0.49	0.40	2.74	69.55	0.50	45.44

limiting runs of FP to <30MinAs, which did not terminate after 90 hours, with 1,572 labels successfully computed and 30 subsumptions skipped since they had ≥30MinAs. Interestingly, in both consequence sets, LP can rarely take advantage of the optimizations early termination and MinA reuse, which might be due to the simple structure of the lattice.

For O^{FUNCT} the comparison between FP and LP is given in the lower part of Tables 1 and 2. Again, the computation of FP was restricted to <10 MinAs. This time, only 363 out of 409 (88%) non-final labels of FP were equal to the final labels of LP (i.e., the boundary). Although the ontology is quite small, LP again behaves much better than FP. The reason could be that in this ontology consequences frequently have a large set of MinAs. From Tables 1 and 2, one can see that LP requires at most three MinLabs for O^{SNOMED}, at most four for O^{FUNCT}, and usually just one MinLab whereas FP usually requires more MinAs.

Table 3 provides results for LP vs. BS with the total order $(L_l, \leq_l)$ as labeling lattice. For O^{SNOMED}, LP takes 130.4 and BS takes 77.1 seconds to label all 27,477 subsumptions. For O^{FUNCT}, LP takes 133.9 and BS takes 68.6 seconds to label all 716 consequences. So BS is about twice as fast as LP. Interestingly, labeling all consequences of O^{FUNCT} and O^{SNOMED} takes roughly the same time, perhaps due to a tradeoff between ontology size and expressivity.

5 Conclusion

We have considered a scenario where ontology axioms are labeled and user labels determine views on the ontology, i.e., sub-ontologies that are obtained by comparing the user label with the axiom labels. Our approach can be used for large-scale ontologies since, on the one hand, it allows to precompute consequences without having to do do this separately for all possible views: once we have computed a boundary for the consequence, checking whether this consequence entailed by a sub-ontology is reduced to a simple label comparison. On the other hand, the fact that we employ a black-box approach for computing the boundary allows us to use existing highly-optimzed reasoners, rather than having to implement a new reasoner from scratch.

Our general framework allows to use any restriction criterion that can be represented using a lattice, such as user roles, levels of trust, granularity, or degrees of uncertainty. In the presence of access restrictions, each user label defines a sub-ontology containing the axioms visible to this user. In the presence of trust restrictions, the user label specifies the trust level required for the ontology axiom. This supports scenarios with axioms from different sources, like company-internal with high trust level and public Web with low trust level. In the presence of uncertainty, e.g. in possibilistic reasoning, each axiom has an associated certainty degree in the interval $[0, 1]$. The user label then specifies the certainty degree required for the axioms and the consequences. Similarly, granularity restrictions (i.e., on how much details the ontology should provide for the user) can be expressed by a total order.

Our experiments have shown that this framework can be applied to large ontologies. From the two black-box algorithms that can deal with arbitrary lattices, the Full Axiom Pinpointing approach is clearly outperformed by the Label-Optimized Axiom Pinpointing approach. For the special case where the labeling lattice is a total order, the latter is again outperformed by the Binary Search approach.

References

1. Baader, F., Brandt, S., Lutz, C.: Pushing the $\mathcal{EL}$ envelope. In: Proc. of 19th Int. Joint Conf. on Art. Int. IJCAI 2005, Edinburgh, UK. Morgan-Kaufmann, San Francisco (2005)
2. Baader, F., Knechtel, M., Peñaloza, R.: Computing boundaries for reasoning in sub-ontologies. Technical Report 09-02, LTCS (2009), `http://lat.inf.tu-dresden.de/research/reports.html`
3. Baader, F., Peñaloza, R.: Axiom pinpointing in general tableaux. In: Olivetti, N. (ed.) TABLEAUX 2007. LNCS (LNAI), vol. 4548, pp. 11–27. Springer, Heidelberg (2007)
4. Baader, F., Peñaloza, R.: Automata-based axiom pinpointing. In: Armando, A., Baumgartner, P., Dowek, G. (eds.) IJCAR 2008. LNCS (LNAI), vol. 5195, pp. 226–241. Springer, Heidelberg (2008)
5. Baader, F., Peñaloza, R.: Axiom pinpointing in general tableaux. Journal of Logic and Computation (To appear, 2009)

6. Baader, F., Peñaloza, R., Suntisrivaraporn, B.: Pinpointing in the description logic $\mathcal{EL}^+$. In: Hertzberg, J., Beetz, M., Englert, R. (eds.) KI 2007. LNCS (LNAI), vol. 4667, pp. 52–67. Springer, Heidelberg (2007)
7. Baader, F., Suntisrivaraporn, B.: Debugging SNOMED CT using axiom pinpointing in the description logic $\mathcal{EL}^+$. In: Proc. of the International Conference on Representing and Sharing Knowledge Using SNOMED (KR-MED 2008), Phoenix, Arizona (2008)
8. Dau, F., Knechtel, M.: Access policy design supported by FCA methods. In: Dau, F., Rudolph, S. (eds.) Proc. of the 17th Int. Conf. on Conceptual Structures, ICCS (2009)
9. Davey, B.A., Priestley, H.A.: Introduction to Lattices and Order, 2nd edn. Cambridge University Press, Cambridge (2002)
10. Gaag, A., Kohn, A., Lindemann, U.: Function-based solution retrieval and semantic search in mechanical engineering. In: Proc. of the 17th Int. Conf. on Engineering Design, ICED 2009 (to appear, 2009)
11. Kalyanpur, A., Parsia, B., Horridge, M., Sirin, E.: Finding all justifications of OWL DL entailments. In: Aberer, K., Choi, K.-S., Noy, N., Allemang, D., Lee, K.-I., Nixon, L.J.B., Golbeck, J., Mika, P., Maynard, D., Mizoguchi, R., Schreiber, G., Cudré-Mauroux, P. (eds.) ASWC 2007 and ISWC 2007. LNCS, vol. 4825, pp. 267–280. Springer, Heidelberg (2007)
12. Kalyanpur, A., Parsia, B., Sirin, E., Hendler, J.A.: Debugging unsatisfiable classes in OWL ontologies. J. Web Sem. 3(4), 268–293 (2005)
13. Lesot, M.-J., Couchariere, O., Bouchon-Meunier, B., Rogier, J.-L.: Inconsistency degree computation for possibilistic description logic: An extension of the tableau algorithm. In: Proc. of NAFIPS 2008, pp. 1–6. IEEE Comp. Soc. Press, Los Alamitos (2008)
14. Meyer, T., Lee, K., Booth, R., Pan, J.Z.: Finding maximally satisfiable terminologies for the description logic ALC. In: Proc. of the 21st Nat. Conf. on Artificial Intelligence (AAAI 2006). AAAI Press/The MIT Press, Menlo Park (2006)
15. Qi, G., Pan, J.Z.: A tableau algorithm for possibilistic description logic. In: Domingue, J., Anutariya, C. (eds.) ASWC 2008. LNCS, vol. 5367, pp. 61–75. Springer, Heidelberg (2008)
16. Qi, G., Pan, J.Z., Ji, Q.: Extending description logics with uncertainty reasoning in possibilistic logic. In: Mellouli, K. (ed.) ECSQARU 2007. LNCS (LNAI), vol. 4724, pp. 828–839. Springer, Heidelberg (2007)
17. Reiter, R.: A theory of diagnosis from first principles. Artificial Intelligence 32(1), 57–95 (1987)
18. Schenk, S.: On the semantics of trust and caching in the semantic web. In: Sheth, A., Staab, S., Dean, M., Paolucci, M., Maynard, D., Finin, T., Thirunarayan, K. (eds.) ISWC 2008. LNCS, vol. 5318, pp. 533–549. Springer, Heidelberg (2008)
19. Schlobach, S., Cornet, R.: Non-standard reasoning services for the debugging of description logic terminologies. In: Gottlob, G., Walsh, T. (eds.) Proc. of the 18th Int. Joint Conf. on Artificial Intelligence (IJCAI 2003), Acapulco, Mexico, pp. 355–362. Morgan Kaufmann, Los Altos (2003)
20. Sirin, E., Parsia, B.: Pellet: An OWL DL reasoner. In: Proc. of the 2004 Description Logic Workshop (DL 2004), pp. 212–213 (2004)
21. Suntisrivaraporn, B.: Polynomial-Time Reasoning Support for Design and Maintenance of Large-Scale Biomedical Ontologies. PhD thesis, Fakultät Informatik, TU Dresden (2009), `http://lat.inf.tu-dresden.de/research/phd/#Sun-PhD-2008`