Context-Dependent Views to Axioms and Consequences of Semantic Web Ontologies

Franz Baader[a], Martin Knechtel[b], Rafael Peñaloza[a]

[a] *Theoretical Computer Science, TU Dresden, Germany*
[b] *SAP AG, SAP Research*

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 contexts like the access rights of a user, 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 context lattice. The different contexts of this ontology are then also expressed by elements of this lattice. For large-scale ontologies, certain consequences (like the subsumption hierarchy) are often pre-computed. Instead of pre-computing these consequences for every context, our approach computes just one label (called a boundary) for each consequence such that a comparison of the user label with the consequence label determines whether the consequence follows from the sub-ontology determined by the context. We 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. Similar to designing ontologies, the process of assigning axiom labels is error-prone. For this reason, we also address the problem of how to repair the labelling of an ontology in case the knowledge engineer notices that the computed boundary of a consequence does not coincide with her intuition regarding in which context the consequence should or should not be visible.

Keywords: Access Restrictions, Views, Contexts, Ontologies

1. Introduction

Description Logics (DL) [1] are a successful family of knowledge representation formalisms, which can be used to represent the conceptual knowledge of an application domain in a structured and formally well-understood way. They are employed in various application domains, such as natural language processing, conceptual modelling in databases, and configuration of technical systems, but their most notable success so far is the adoption of the DL-based language OWL as standard ontology language for the Semantic Web. From the DL point of view, an ontology is a finite set of axioms, which formalize our knowledge about the relevant concepts of the application domain. From this explicitly described knowledge, the reasoners implemented in DL systems can then derive implicit consequence. Application programs or human users interacting with the DL system thus have access not only to the explicitly represented knowledge, but also to its logical consequences. In order to provide fast access to the implicit knowledge, certain consequences (such as the subsumption hierarchy between named concepts) are often pre-computed by DLs systems.

In this paper, we investigate how this sort of pre-computation can be done in an efficient way in a setting where users can access only parts of an ontology, and should see only what follows from these parts. To be more precise, assume that you have a large ontology O, but you want to offer different users different views on this ontology with respect to their context. In other words, each user can see only a subset of the large ontology, which is defined by the context she operates in. The context may be the level of expertise of the user, the access rights that she has been granted, or the level of detail that is deemed to be appropriate for the current setting, etc. More concretely, one could use context-dependent views for reducing information overload by providing only the information appropriate to the experience level of a user. For example, in a medical ontology we might want to offer one view for a patient that has only lay knowledge, one for a general practitioner, one for a cardiologist, one for a pulmonologist, etc. Another example is provided by proprietary commercial ontologies, where access is restricted according to a certain policy. The policy evaluates the context of each user by considering the assigned user roles, and then decides whether some axioms and the implicit consequences that can be derived from them are available to this user or not.

One naïve approach towards dealing with such context-dependent views of ontologies would be to materialize a

Email addresses: `baader@tcs.inf.tu-dresden.de` (Franz Baader), `martin.knechtel@sap.com` (Martin Knechtel), `penaloza@tcs.inf.tu-dresden.de` (Rafael Peñaloza)

separate sub-ontology of the overall large ontology for each possible user context. However, this could potentially lead to an exponential number of ontologies having to be maintained, if we define one user context for each subset of the original ontology. This would imply that any update in the overall ontology needs to be propagated to each of the sub-ontologies, and any change in the context model, such as a new user role hierarchy or a new permission for a user role, may require removing or adding such subsets. Even worse, for each of these sub-ontologies, the relevant implicit consequences would need to be pre-computed and stored separately. To avoid these problems, we propose a different solution in this paper. The idea is to keep just the large ontology O, but assign "labels" to all axioms in the ontology and to all users in such a way that an appropriate comparison of the axiom label with the user label determines whether the axiom belongs to the sub-ontology for this user or not. This comparison will be computationally cheap and can be efficiently implemented with an index structure to look up all axioms with a given label. To be more precise, we use a set of labels L together with a partial order $\leq$ on L and assume that every axiom $a \in O$ has an assigned label $\mathsf{lab}(a) \in L$.[1] The labels $\ell \in L$ are also used to define user contexts (which can be interpreted as access rights, required level of granularity, etc.). The sub-ontology accessible for the context with label $\ell \in L$ is defined to be

$$O_{\geq \ell} := \{a \in O \mid \mathsf{lab}(a) \geq \ell\}.$$

Clearly, the user of a DL-based ontology is not only able to access its axioms, but also the consequences of these axioms. That is, a user whose context has label ℓ should also be allowed to see all the consequences of $O_{\geq \ell}$.

As mentioned already, certain consequences are usually pre-computed by DL systems in order to avoid expensive reasoning during the deployment phase of the ontology. For example, in the version of the large medical ontology SNOMED CT[2] that is distributed to hospitals and doctors, all the subsumption relationships between the concept names occurring in the ontology are pre-computed. For a labelled ontology as introduced above, pre-computing that a certain consequence c follows from the whole ontology O is not sufficient. In fact, a user whose context has label ℓ should only be able to see the consequences of $O_{\geq \ell}$, and since $O_{\geq \ell}$ may be smaller than O, the consequence c of O may not be a consequence of $O_{\geq \ell}$. As said above, pre-computing consequences for all possible user labels is not a good idea since then one might have to compute and store consequences for exponentially many different subsets of O. Our solution to this problem is to compute a so-called *boundary* for the consequence c, i.e., an element ν of L such that c follows from $O_{\geq \ell}$ iff $\ell \leq \nu$. Thus, instead of pre-computing whether this consequence is valid for every possible sub-ontology, our approach computes just one label for each consequence such that a simple comparison of the context label with the consequence label determines whether the consequence follows from the corresponding sub-ontology or not.

There are two main approaches for computing a boundary. The *glass-box approach* takes a specific reasoner (or reasoning technique) for an ontology language 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 [2, 3] (where the lattice is the interval $[0, 1]$ with the usual order), glass-box approaches to axiom pinpointing in Description Logics [4, 5, 6, 7, 8] (where the lattice consists of (equivalence classes of) monotone Boolean formulae with implication as order [8]), and RDFS reasoning over labelled triples with modified inference rules for access control and provenance tracking [9, 10]. The problem with glass-box approaches is that they have to be developed and implemented for every ontology language and reasoning approach anew and optimizations of the original reasoning approach 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. 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, 12]. The third uses binary search and can only be applied if the context 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 [13].

Of course, the boundary computation only yields the correct results if the axiom labels have been assigned in a correct way. Unfortunately, just like creating ontology axioms, appropriately equipping these axioms with context labels is an error-prone task. For instance, in an access control application, several axioms that in isolation may seem innocuous could, together, be used to derive a consequence that a certain user is not supposed to see. If the knowledge engineer detects that a consequence c has an inappropriate boundary, and thus allows access to the consequence by users that should not see it, then she may want to modify the axiom labelling in such a way that the boundary of c is updated to the desired label. This problem is very closely related to the problem of repairing an ontology. Indeed, to correct the boundary of a consequence, one needs to be able to detect the axioms that are responsible for it, since only their labels have an influence on this boundary. In a large-scale ontology, this task needs to be automated, as analysing hundreds of thousands of

[1] We will in fact impose the stronger restriction that $(L, \leq)$ defines a lattice (see Section 2).

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

axioms by hand is not feasible.

To provide for such an automated label repair mechanism, we develop a black-box method for computing minimal sets of axioms that, when relabelled, yield the desired boundary for c; we call these *minimal change sets.* The main idea of this method is again based on the Hitting Set Tree (HST) algorithms that have been developed for axiom-pinpointing. However, we show that the original labelling function can be exploited to decrease the search space. This algorithm can be used to output all minimal change sets. The knowledge engineer can then choose which of them to use for the relabelling, depending on different criteria. Unfortunately, just as in axiom-pinpointing, there may be exponentially many such minimal change sets, and thus analysing them all by hand may not be possible. We thus also develop an algorithm that computes only one change set having the smallest cardinality. This choice is motivated by a desire to make as few changes in the original labelled ontology as possible during the repair. We show that, in this case, a cardinality limit can be used to further optimize the algorithm.

All the algorithms described in this paper have been implemented and tested over large-scale ontologies from real-life applications, and using a context lattice motivated by an access control application scenario. Our experimental results show that our methods perform well in practice.

This paper extends and improves the results previously published in [14, 15]. More precisely, the algorithms for computing the boundaries of consequences were presented in [14], while the problem of repairing the boundaries was addressed in [15]. Here, we (i) provide full proofs for all the theoretical results presented, (ii) present better optimizations to our algorithms, and (iii) provide a thorough comparison of the different algorithmic approaches through our experimental results. In order to make the paper accessible also for practitioners who want to apply the general framework, but are not interested in the full formal details and proofs, we use a running example that should provide enough details to understand the main ideas underlying our approach.

2. Preliminaries

To stay as general as possible, we do not fix a specific ontology language. We just assume that we have one such ontology language determining which finite sets of *axioms* are admissible as *ontologies*, such that every subset of an ontology is itself an ontology. If O' is a subset of the ontology O, then O' is called a *sub-ontology* of O. Consider, for instance, a Description Logic $\mathcal{L}$ (e.g., the DL $\mathcal{SROIQ}(\mathbf{D})$ underlying the OWL 2 web ontology language). 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 *assertional axioms* of the form $C(a)$ and $r(a, b)$, with C an $\mathcal{L}$-concept description, a, b individual names, and r a role name. For a fixed ontology language, a *monotone consequence relation* $\models$ is a binary relation between ontologies O of this language and *consequences* c such that, for every ontology O, it holds that if $O' \subseteq O$ and $O' \models c$, then $O \models c$. Examples of consequences in $\mathcal{SROIQ}(\mathbf{D})$ are subsumption relations $A \sqsubseteq B$ for concept names A, B or assertions $C(a)$. Note that we can abstract from the details of the ontology language and the consequence relation since we intend to use a black-box approach, i.e., all we need is that there is an algorithm that, given an ontology O and a consequence c, is able to deduce whether $O \models c$ holds or not.

If $O \models c$, we may be interested in finding the axioms responsible for this fact. *Axiom-pinpointing* is the task of finding the minimal sub-ontologies that entail a given consequence (MinAs), or dually, the minimal sets of axioms that need to be removed or repaired to avoid deriving the consequence (diagnoses).

Definition 2.1 (MinA, diagnosis)**.** A sub-ontology $S \subseteq O$ is called a *MinA* for O,c if $S \models c$ and for every $S' \subset S$, it holds that $S' \not\models c$.[3]

A *diagnosis* for O,c is a sub-ontology $S \subseteq O$ such that $O \setminus S \not\models c$ and $O \setminus S' \models c$ for all $S' \subset S$.

The sets of MinAs and diagnoses are dual in the sense that from the set of all MinAs, it is possible to compute the set of all diagnoses, and vice versa, through a Hitting Set computation [4].

As a running example, we will use the following scenario of access restrictions, which is part of the research project THESEUS/PROCESSUS [16]. Within this project, semantically annotated documents describe Web services offered and sold on a marketplace in the Web, like traditional goods are sold on Amazon, eBay, and similar Web marketplaces. Different types of users are involved with different permissions that allow them to create, advertise, sell, buy, etc. the services. Access is restricted not only to individual documents but also to a large ontology containing all the semantic annotations at one place.

Example 2.2. Consider an ontology O from a marketplace in the Semantic Web representing knowledge about the Ecological Value Calculator service (*ecoCalc*), EU Ecological Services (*EUecoS*), High Performance Services (*HPerfS*), services with few customers (*SFewCust*), services generating low profit (*LowProfitS*), and services with a price increase (*SPrIncr*) having the following axioms:

$$\begin{array}{ll} a_1: & \mathit{EUecoS} \sqcap \mathit{HPerfS}(\mathit{ecoCalc}) \\ a_2: & \mathit{HPerfS} \sqsubseteq \mathit{SFewCust} \sqcap \mathit{LowProfitS} \\ a_3: & \mathit{EUecoS} \sqsubseteq \mathit{SFewCust} \sqcap \mathit{LowProfitS} \\ a_4: & \mathit{SFewCust} \sqsubseteq \mathit{SPrIncr} \\ a_5: & \mathit{LowProfitS} \sqsubseteq \mathit{SPrIncr} \end{array}$$

The assertion $\mathit{SPrIncr}(\mathit{ecoCalc})$ is a consequence of O that follows from each of the MinAs $\{a_1, a_2, a_4\}$, $\{a_1, a_2, a_5\}$, $\{a_1, a_3, a_4\}$, and $\{a_1, a_3, a_5\}$, and has three diagnoses, namely $\{a_1\}, \{a_2, a_3\}$, and $\{a_4, a_5\}$.

[3] MinAs are sometimes also called *justifications*, e.g. in [6, 11].

As mentioned before, our axiom labels come from an appropriate lattice. A *lattice* $(L, \leq)$ is a set L together with a partial order $\leq$ on L such that a finite subset $S \subseteq L$ always has a join (least upper bound) $\bigoplus S$ and a meet (greatest lower bound) $\bigotimes S$ [17]. The lattice $(L, \leq)$ is *distributive* if the join and meet operators distribute over each other. Another lattice-theoretic notion that will be important for the rest of the paper is that of join-prime elements.

Definition 2.3 (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 $\ell \leq k_0$.

For instance, the lattice $(L, \leq)$ depicted in Figure 1 has four join prime elements relative to L, namely ℓ_0, ℓ_2, ℓ_3, and ℓ_5. The element ℓ_4 is not join prime relative to L since $\ell_4 \leq \ell_4 = \ell_5 \oplus \ell_3$, but $\ell_4 \not\leq \ell_5$ and $\ell_4 \not\leq \ell_3$.

We now explain how lattices can be used to encode contexts, and solve reasoning problems relative to them. From our running example, we want to produce an access control system that regulates the allowed permissions for each user according to her user role. Our example focuses on reading access only. A common representation of user roles and their permissions to access objects is the access control matrix [18]. Using methods from Formal Concept Analysis, as presented in [19], a lattice representation of the access control matrix can be obtained. In fact, the lattice depicted in Figure 1 was derived in this way.

In the general setting, we will use elements of the lattice $(L, \leq)$ to define different *contexts* or *views* of an ontology. Depending on the application in hand, these contexts can have different meanings, such as access rights, level of expertise, trustworthiness, etc.

Given an ontology O, every axiom $a \in O$ is assigned a label $\mathsf{lab}(a) \in L$, which intuitively expresses the contexts from which the axiom a can be accessed. An ontology extended with such a labelling function lab will be called a *labelled ontology*. We will use the expression L_{lab} to denote the set of all labels occurring in the labelled ontology O; that is, $L_{\mathsf{lab}} := \{\mathsf{lab}(a) \mid a \in O\}$. Each element $\ell \in L$ then defines the context sub-ontology[4]

$$O_{\geq \ell} := \{a \in O \mid \mathsf{lab}(a) \geq \ell\}.$$

Conversely, every sub-ontology $S \subseteq O$ defines an element $\lambda_S \in L$, called the *label* of S, given by $\lambda_S := \bigotimes_{a \in S} \mathsf{lab}(a)$.

Some simple relationships between ontologies and their labels are stated in the following lemma.

Lemma 2.4. *Let $(L, \leq)$ be a lattice, O an ontology, and $\mathsf{lab} : O \to L$. For every $\ell \in L$ and $S \subseteq O$, it holds that*

1. $\ell \leq \lambda_{O_{\geq \ell}}$,
2. $S \subseteq O_{\geq \lambda_S}$, *and*
3. $O_{\geq \ell} = O_{\geq \lambda_{O_{\geq \ell}}}$.

Proof. For the first statement, by definition $\ell \leq \mathsf{lab}(a)$ holds for all $a \in O_{\geq \ell}$. Thus, $\ell \leq \bigotimes_{a \in O_{\geq \ell}} \mathsf{lab}(a) = \lambda_{O_{\geq \ell}}$. Regarding the second claim, for every $a \in S$ it holds that $\lambda_S = \bigotimes_{s \in S} \mathsf{lab}(s) \leq \mathsf{lab}(a)$, which implies that $a \in O_{\geq \lambda_S}$. Now, consider the last claim. First, as $\ell \leq \lambda_{O_{\geq \ell}}$, it holds trivially that $O_{\geq \lambda_{O_{\geq \ell}}} \subseteq O_{\geq \ell}$. From the second claim it also follows that $O_{\geq \ell} \subseteq O_{\geq \lambda_{O_{\geq \ell}}}$. $\square$

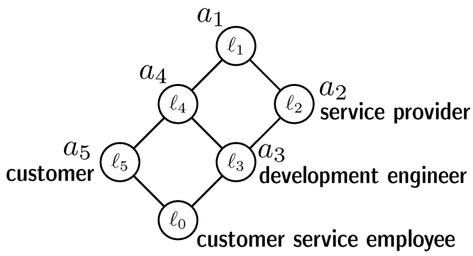

Figure 1: A lattice with 4 contexts and 5 axioms assigned to it

Example 2.5. Let $(L, \leq)$ be the lattice shown in Figure 1, where elements $\ell_0, \ell_2, \ell_3, \ell_5$ represent the different kinds of users (that is, contexts) that have access to an ontology. Let lab be the labelling function assigning to each axiom a_i of the ontology O from Example 2.2 the label ℓ_i, as depicted also in Figure 1. The label ℓ_3 defines the context of a development engineer for which the sub-ontology $O_{\geq \ell_3} = \{a_1, a_2, a_3, a_4\}$, along with all its consequences, is visible.

Notice that labels that are lower in the lattice define larger context sub-ontologies. In other words, a user assigned to a context sub-ontology lower in the lattice will have access to more axioms (and thus, consequences) than a user belonging to a context above her.

3. Pre-Computing Context-Dependent Implicit Knowledge

Just as every axiom is accessible only for certain contexts, a consequence of the ontology will only be derivable in those contexts that have access to enough axioms to deduce it. We are interested in computing adequate labels (called boundaries) for such implicit consequences, which express, just as the labels of the axioms, which contexts are capable of deducing them from their visible axioms.

Notice that, if a consequence c follows from $O_{\geq \ell}$ for some $\ell \in L$, it must also follow from $O_{\geq \ell'}$ for every $\ell' \leq \ell$, since then $O_{\geq \ell} \subseteq O_{\geq \ell'}$. A maximal element of L that still entails the consequence will be called a margin for this consequence.

[4] To define this sub-ontology, an arbitrary partial order would suffice. However, the existence of suprema and infima will be important for the computation of a boundary of a consequence (see Section 4).

Definition 3.1 (Margin)**.** Let c be a consequence that follows from the ontology O. The label $\mu \in L$ is called a (O, c)-*margin* if $O_{\geq \mu} \models c$, and for every ℓ with $\mu < \ell$ we have $O_{\geq \ell} \not\models c$.

If O and c are clear from the context, we usually ignore the prefix (O, c) and call μ simply a *margin.* The following lemma shows three basic properties of the set of margins, which will be useful throughout this paper.

Lemma 3.2. *Let c be a consequence that follows from the ontology O. We have:*

1. *If μ is a margin, then $\mu = \lambda_{O_{\geq \mu}}$;*
2. *if $O_{\geq \ell} \models c$, then there is a margin μ such that $\ell \leq \mu$;*
3. *there are at most $2^{|O|}$ margins for c.*

Proof. To show 1, let $\mu \in L$. Lemma 2.4 yields $\mu \leq \lambda_{O_{\geq \mu}}$ and $O_{\geq \mu} = O_{\geq \lambda_{O_{\geq \mu}}}$, and thus $O_{\geq \lambda_{O_{\geq \mu}}} \models c$. If $\mu < \lambda_{O_{\geq \mu}}$, then this $\lambda_{O_{\geq \mu}}$ contradicts our assumption that μ is a margin; hence $\mu = \lambda_{O_{\geq \mu}}$. Point 3 is a trivial consequence of 1: since every margin has to be of the form λ_S for some $S \subseteq O$, there are at most as many margins as there are subsets of O.

For the remaining point, let $\ell \in L$ be such that $O_{\geq \ell} \models c$. Let $m := \lambda_{O_{\geq \ell}}$. From Lemma 2.4, it follows that $\ell \leq m$ and $O_{\geq m} = O_{\geq \ell}$, and hence $O_{\geq m} \models c$. If m is a margin, then the result holds; suppose to the contrary that m is not a margin. Then, there must exist an $\ell_1, m < \ell_1$, such that $O_{\geq \ell_1} \models c$. As $m = \lambda_{O_{\geq m}}$, there must exist an axiom $a \in O$ such that $m \leq \mathsf{lab}(a)$, but $\ell_1 \not\leq \mathsf{lab}(a)$. In fact, if $m \leq \mathsf{lab}(a) \Rightarrow \ell_1 \leq \mathsf{lab}(a)$ would hold for all $a \in O$, then $m = \lambda_{O_{\geq \ell}} = \lambda_{O_{\geq m}} = \bigotimes_{\mathsf{lab}(a) \geq m} \mathsf{lab}(a) \geq \ell_1$, contradicting our choice of ℓ_1. The existence of this axiom a implies that $O_{\geq \ell_1} \subset O_{\geq m}$. Let $m_1 := \lambda_{O_{\geq \ell_1}}$; then $m < \ell_1 \leq m_1$. If m_1 is not a margin, then we can repeat the same process to obtain a new m_2 with $m < m_1 < m_2$ and $O_{\geq m} \supset O_{\geq m_1} \supset O_{\geq m_2}$, and so on. As O is finite, there exists a finite k where this process stops, and hence m_k is a margin. ☐

If we know that μ is a margin for the consequence c, then we know whether c follows from $O_{\geq \ell}$ for all $\ell \in L$ that are comparable with μ: if $\ell \leq \mu$, then c follows from $O_{\geq \ell}$, and if $\ell > \mu$, then c does not follow from $O_{\geq \ell}$. However, this gives us no information regarding elements that are incomparable with μ. In order to obtain a full picture of when the consequence c follows from $O_{\geq \ell}$ for an arbitrary element ℓ 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 c and those for which this is not the case, i.e., ν should satisfy the following: for every $\ell \in L$, $O_{\geq \ell} \models c$ iff $\ell \leq \nu$. Unfortunately, such an element need not always exist, as demonstrated by the following example.

Example 3.3. Consider the lattice $(L, \leq)$ depicted in Figure 1 and let O' be an ontology consisting of axioms b_1 and b_2, labelled with ℓ_4 and ℓ_2, respectively. Let now c be a consequence such that, for every $S \subseteq O'$, we have $S \models c$ iff $|S| \geq 1$. It is easy to see that there is no element $\nu \in L$ that satisfies the condition described above. Indeed, if we choose $\nu \in \{\ell_0, \ell_3, \ell_4, \ell_5\}$, then ℓ_2 violates the condition, as $\ell_2 \not\leq \nu$, but $O'_{\geq \ell_2} = \{b_2\} \models c$. Similarly, if we choose $\nu = \ell_2$, then ℓ_1 violates the condition. Finally, if $\nu = \ell_1$ is chosen, then ℓ_1 itself violates the condition: $\ell_1 \leq \nu$, but $O'_{\geq \ell_1} = \emptyset \not\models c$.

It is nonetheless possible to find an element that satisfies a restricted version of the condition, where we do not impose that the property (i.e. $O_{\geq \ell} \models c$ iff $\ell \leq \nu$) must hold for every element of the context lattice, but only for those elements that are *join prime* relative to the labels of the axioms in the ontology.

Definition 3.4 (Boundary)**.** Let O be an ontology and c a consequence. An element $\nu \in L$ is called a (O, c)-*boundary* if for every element $\ell \in L$ that is join prime relative to L_{lab} it holds that $\ell \leq \nu$ iff $O_{\geq \ell} \models c$.

As with margins, if O and c are clear from the context, we will simply call such a ν a *boundary.* When it is clear that the computed boundary and no assigned label is meant, we also often call it *consequence label.* In Example 3.3, the element ℓ_1 is a boundary. Indeed, every join prime element ℓ relative to $\{\ell_4, \ell_2\}$ (i.e., every element of L except for ℓ_1) is such that $\ell < \ell_1$ and $O'_{\geq \ell} \models c$.

From a practical point of view, our definition of a boundary has the following implication: we must enforce that contexts are always defined through labels that are join prime relative to the set L_{lab} of all labels occurring in the ontology. In Example 2.5, all the elements of the context lattice except ℓ_1 and ℓ_4 are join prime relative to L_{lab} and for this reason $\ell_0, \ell_2, \ell_3, \ell_5$ are all valid context labels and can thus be used to represent user roles as illustrated. Given a context label ℓ_u, we will say that a consequence c is in the context if $\ell_u \leq \nu$ for some boundary ν.

Notice however that the boundary is not guaranteed to be unique, as shown in the following example.

Example 3.5. Consider the lattice L obtained from the lattice in Figure 1 by removing the element ℓ_4 and keeping the order relation unchanged. Let now $O = \{a_1, a_2\}$ and c be such that $S \models c$ iff $a_1 \in S$. If we set $\mathsf{lab}(a_1) = \ell_3$, $\mathsf{lab}(a_2) = \ell_5$, it then follows that (i) ℓ_0, ℓ_3, ℓ_5 are all join-prime elements relative to L_{lab}, and (ii) $O_{\geq \ell} \models c$ iff $\ell \leq \ell_3$. But notice that $\ell_3 \leq \ell_2$ and $\ell_5 \not\leq \ell_2$; thus, ℓ_2 and ℓ_3 are both (O, c)-boundaries.

Before formally describing how to compute (Section 4) and correct (Section 5) boundaries for consequences of an ontology, we briefly describe what are the requirements and benefits of our method from a knowledge engineering point of view.

As a prerequisite, we assume that the context lattice L is known, and that every axiom of the ontology is labelled with an element of L expressing the set of contexts that have access to it. To obtain this lattice and labeling, the knowledge engineer can first build a *context matrix* relating every relevant context to the sub-ontology that it can access. The knowledge engineer only needs to "tag" every axiom with the corresponding contexts; tagging elements is already a common task in Web 2.0 applications, and no further effort is required from our framework. Formal Concept Analysis [20] can then be used to obtain a lattice representation of this matrix, together with a labelling function. This labelling function is ensured to be the least restrictive possible satisfying all the restrictions specified by the knowledge engineer in the context matrix. Indeed, the context lattice depicted in Figure 1 was derived in this way [19].

Given a labelled ontology, computing a boundary corresponds to reasoning with respect to *all* contexts simultaneously, modulo an inexpensive label comparison: given a boundary ν for a consequence c, every context below ν in the lattice can derive c, while all others cannot.

Boundaries also simplify the work of verifying the correctness of the labelling function, since the knowledge engineer needs only compare the boundary of implicit consequences with the set of contexts that should access them, rather than analysing every context independently. If a consequence has an undesired boundary, then our method provides suggestions for correcting it, while keeping the changes in the labelling function to the minimum. In the same manner, our approach is helpful for the maintenance of labelled ontologies.

4. Computing a Boundary

We now focus on the problem of computing a boundary. We first present an algorithm based on axiom-pinpointing, which introduces the main ideas for the computation of a boundary. We then improve on these ideas by taking the labels of the axioms into account during the computation. Finally, we show that, if the lattice is a total order, then a modification of binary search can be used to compute a boundary. All these algorithms are based on the following lemma.

Lemma 4.1. *Let $\mu_1, \ldots, \mu_n$ be all (O, c)-margins. Then $\bigoplus_{i=1}^n \mu_i$ is a boundary for O, c.*

Proof. Let $\ell \in L$ be join prime relative to L_{lab}. We need to show that $\ell \leq \bigoplus_{i=1}^n \mu_i$ iff $O_{\geq \ell} \models c$. Assume first that $O_{\geq \ell} \models c$. Then, from 2 of Lemma 3.2, it follows that there is a margin μ_j such that $\ell \leq \mu_j$, and thus $\ell \leq \bigoplus_{i=1}^n \mu_i$.

Conversely, let $\ell \leq \bigoplus_{i=1}^n \mu_i$. From 1 of Lemma 3.2, it follows that $\mu_i \in (L_{\mathsf{lab}})_\otimes$ for every $i, 1 \leq i \leq n$. As ℓ is join prime relative to L_{lab}, it then holds that there is a j such that $\ell \leq \mu_j$ and hence, by the definition of a margin and the monotonicity of the consequence relation, $O_{\geq \ell} \models c$. □

By Lemma 3.2, a consequence always has finitely many margins, and thus Lemma 4.1 shows that a boundary always exists. As shown in Example 3.5, a consequence may have boundaries different from the one of Lemma 4.1. To identify the particular boundary of Lemma 4.1, we will call it the *margin-based boundary*. For the rest of this section, we will focus on computing this boundary.

4.1. Using Full Axiom Pinpointing

From Lemma 4.1 we know that the set of all margins yields sufficient information for computing a boundary. The question is thus how to compute this set. We now show that every margin can be obtained from some MinA.

Lemma 4.2. *For every margin μ for c there is a MinA S such that $\mu = \lambda_S$.*

Proof. If μ is a margin, then $O_{\geq \mu} \models c$ by definition. Thus, there exists a MinA $S \subseteq O_{\geq \mu}$. Since $\mu \leq \mathsf{lab}(a)$ for every $a \in O_{\geq \mu}$, this in particular holds also for every axiom in S, and hence $\mu \leq \lambda_S$. Additionally, as $S \subseteq O_{\geq \lambda_S}$, we have $O_{\geq \lambda_S} \models c$. This implies $\mu = \lambda_S$ since otherwise $\mu < \lambda_S$, and then μ would not be a margin. □

Notice that this lemma does not imply that the label of any MinA S corresponds to a margin. Indeed, for the ontology and consequence of Example 2.5, two of the four MinAs are $\{a_1, a_2, a_5\}, \{a_1, a_2, a_4\}$ whose labels are ℓ_0 and ℓ_3, respectively, and hence the label of the former cannot be a margin (since $\ell_0 < \ell_3$). However, as the consequence follows from every MinA S, Point 2 of Lemma 3.2 shows that $\lambda_S \leq \mu$ for some margin μ. The following theorem is an immediate consequence of this fact together with Lemma 4.1 and Lemma 4.2.

Theorem 4.3. *If $S_1, \ldots, S_n$ are all MinAs for O and c, then $\bigoplus_{i=1}^n \lambda_{S_i}$ is the margin-based boundary for c.*

Example 4.4. We continue Example 2.5 where each axiom a_i is labelled with $\mathsf{lab}(a_i) = \ell_i$. We are interested in the boundary for the consequence *SPrIncr*(*ecoCalc*), which has the MinAs $\{a_1, a_2, a_4\}, \{a_1, a_2, a_5\}, \{a_1, a_3, a_4\}$, and $\{a_1, a_3, a_5\}$. From Theorem 4.3, it follows that the margin-based boundary for c is $\ell_3 \oplus \ell_0 \oplus \ell_3 \oplus \ell_0 = \ell_3$. This in particular shows that only the contexts of development engineers and customer service employees, defined through the labels ℓ_3 and ℓ_0, respectively, can derive the consequence.

According to the above theorem, to compute a boundary, it is sufficient to compute all MinAs. Several methods exist for computing the set of all MinAs, either directly [4, 11, 21] or through a so-called pinpointing formula [22, 8, 7], 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] However, since not

[5] http://protege.stanford.edu/

[6] http://code.google.com/p/cel/

all MinAs may really contribute to computing the boundary, first computing *all* MinAs may require extensive superfluous work.

4.2. Using Label-Optimized Axiom Pinpointing

From Lemma 4.2 we know that every margin is of the form λ_S for some MinA S. In the previous subsection we have used this fact to compute a boundary by first obtaining the MinAs and then computing their labels. However, this idea ignores that the relevant part of the computation of a boundary are the labels of the MinAs, rather than the MinAs *per se*. This process can be optimized if we directly compute the labels of the MinAs, without necessarily computing the actual MinAs. Additionally, it is not necessary to compute the label of every MinA, but only of those that correspond to margins, that is, those that are maximal w.r.t. the lattice ordering $\leq$. For instance, in Example 4.4, we could avoid computing the two MinAs that have label ℓ_0.

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 [23] method (HST approach) for axiom pinpointing [11, 12]. First, we briefly describe the HST approach for computing all MinAs, which will serve as a starting point for our modified version.

The HST-based method for axiom pinpointing 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 S_0 for O, which is used to label the root of the tree. Then, for every axiom a in S_0, a successor node is created. If $O \setminus \{a\}$ does not entail the consequence, then this node is a dead end. Otherwise, $O \setminus \{a\}$ still entails the consequence. In this case, a MinA S_1 for $O \setminus \{a\}$ is computed and used to label the node. The MinA S_1 for $O \setminus \{a\}$ obtained this way is also a MinA of O, and it is guaranteed to be distinct from S_0 since $a \notin S_1$. Then, for each axiom a' in 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 $O \setminus \{a, a'\}$ still has the consequence, etc. This process obviously terminates since O is a finite set of axioms, and the end result is a tree, where each node that is not a dead end is labelled with a MinA, and every existing MinA appears as the label of at least one node of the tree (see [11, 12] for further details).

An important ingredient of the HST algorithm is a procedure that computes a single MinA from an ontology. Such a procedure can, e.g., 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 [21] for a description of this and of a more sophisticated logarithmic procedure for computing one MinA).

We will use this same idea as a basis for computing the margin-based boundary for a consequence. As said before,

Algorithm 1 Compute a MinLab of one MinA

Procedure $\mathsf{min\text{-}lab}(O, c)$
Input: O: ontology; c: consequence
Output: $M_L \subseteq L$: a MinLab

1: **if** $O \not\models c$ **then**
2:     **return** no MinA
3: $S := O$
4: $M_L := \emptyset$
5: **for** every $k \in L_{\mathsf{lab}}$ **do**
6:     **if** $\bigotimes_{l \in M_L} l \not\leq k$ **then**
7:         **if** $S_{\neq k} \models c$ **then**
8:             $S := S_{\neq k}$
9:         **else**
10:             $M_L := (M_L \setminus \{l \mid k < l\}) \cup \{k\}$
11: **return** M_L

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 the label of some MinA S based on this idea. More precisely, the algorithm does not compute a single label, but rather a *minimal label set (MinLab)* of a MinA S.

Definition 4.5 (Minimal label set)**.** Let S be a MinA for c. A set $K \subseteq \{\mathsf{lab}(a) \mid a \in S\}$ is called a *MinLab* of S if the elements of K are pairwise incomparable and $\lambda_S = \bigotimes_{\ell \in K} \ell$.

Algorithm 1 removes all the labels that do not contribute to a MinLab. If O is an ontology and $\ell \in L$, then the expression $O_{\neq \ell}$ appearing at Line 7 denotes the sub-ontology $O_{\neq \ell} := \{a \in O \mid \mathsf{lab}(a) \neq \ell\}$. If, after removing all the axioms labelled with k, the consequence still follows, then there is a MinA none of whose axioms is labelled with k. In particular, this MinA has a MinLab not containing k; thus, all the axioms labelled with k can be removed in our search for a MinLab. If the axioms labelled with k cannot be removed, then all MinAs of the current sub-ontology need an axiom labelled with k, and hence k is stored in the set M_L. This set is also 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 (Line 6).

Theorem 4.6. *Let O and c be such that $O \models c$. There is a MinA S_0 for c such that Algorithm 1 outputs a MinLab of S_0.*

Proof. As $O \models c$, the algorithm will enter the **for** loop. This loop keeps the following two invariants: (i) $S \models c$ and (ii) for every $\ell \in M_L, S_{\neq \ell} \not\models c$. The invariant (i) is ensured by the condition in Line 7 that must be satisfied before S is modified. Otherwise, that is, if $S_{\neq \ell} \not\models c$, then ℓ

is added to M_L (Line 10) which, together with the fact that S is always modified to a smaller set (Line 8), ensures (ii). Hence, when the loop finishes, the sets S and M_L satisfy both invariants. As $S \models c$, there is a MinA $S_0 \subseteq S$ for c. For each $\ell \in M_L$, there must be an axiom $a \in S_0$ such that $\mathsf{lab}(a) = \ell$, otherwise, $S_0 \subseteq S_{\neq \ell}$ and hence $S_{\neq \ell} \models c$, which contradicts invariant (ii); thus, $M_L \subseteq \{\mathsf{lab}(a) \mid a \in S_0\}$ and in particular $\lambda_{S_0} \leq \bigotimes_{\ell \in M_L} \ell$.

It remains to show that the inequality in the other direction holds as well. Consider now $k \in \{\mathsf{lab}(a) \mid a \in S\}$ and let M_L^k be the value of M_L when the **for** loop was entered with value k. We have that $\bigotimes_{\ell \in M_L} \ell \leq \bigotimes_{\ell \in M_L^k} \ell$. If $\bigotimes_{\ell \in M_L} \ell \not\leq k$, then also $\bigotimes_{\ell \in M_L^k} \ell \not\leq k$, and thus it fulfills the test in Line 6, and continues to Line 7. If that test is satisfied, then all the axioms with label k are removed from S, contradicting the assumption that $k = \mathsf{lab}(a)$ for some $a \in S$. Otherwise, k is added to M_L, which contradicts the assumption that $\bigotimes_{\ell \in M_L} \ell \not\leq k$. Thus, for every axiom a in S, $\bigotimes_{\ell \in M_L} \ell \leq \mathsf{lab}(a)$; hence $\bigotimes_{\ell \in M_L} \ell \leq \lambda_S \leq \lambda_{S_0}$. □

Once the label of a MinA has been found, we can compute new MinLabs by a successive deletion of axioms from the ontology using the HST approach. Suppose that we have computed a MinLab $\mathcal{M}_0$, and that $\ell \in \mathcal{M}_0$. If we remove all the axioms in the ontology labelled with ℓ, and compute a new MinLab $\mathcal{M}_1$ 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 MinLabs, and hence the labels of all MinAs. However, since our goal is to compute the supremum of these labels, the algorithm can be further optimized by avoiding the computation of those 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 O, $O_{\not\leq \ell}$ denotes the sub-ontology obtained from O by removing all axioms whose labels are $\leq \ell$. Now, assume that we have computed the MinLab $\mathcal{M}_0$, and that $\mathcal{M}_1 \neq \mathcal{M}_0$ is the MinLab of the MinA S_1. For all $\ell \in \mathcal{M}_0$, if S_1 is not contained in $O_{\not\leq \ell}$, then S_1 contains an axiom with label $\leq \ell$. Consequently, $\bigotimes_{m \in \mathcal{M}_1} m = \lambda_{S_1} \leq \bigotimes_{m \in \mathcal{M}_0} m$, 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 $\mathsf{HST\text{-}boundary}$, three global variables are declared: $\mathbf{C}$, $\mathbf{H}$ (initialized with $\emptyset$), and ν. The variable $\mathbf{C}$ stores all the MinLabs 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 MinLab $\mathcal{M}$, which is used to label the root of a tree. For each element of $\mathcal{M}$, a branch is created by calling the procedure $\mathsf{expand\text{-}HST}$.

Algorithm 2 Compute a boundary by a HST algorithm

Procedure $\mathsf{HST\text{-}boundary}(O, c)$
Input: O: ontology; c: consequence
Output: boundary ν for c
1: **Global** : $\mathbf{C}, \mathbf{H} := \emptyset; \nu$
2: $\mathcal{M} := \mathsf{min\text{-}lab}(O, c)$
3: $\mathbf{C} := \{\mathcal{M}\}$
4: $\nu := \bigotimes_{\ell \in \mathcal{M}} \ell$
5: **for** each label $\ell \in \mathcal{M}$ **do**
6: $\quad$ $\mathsf{expand\text{-}HST}(O_{\not\leq \ell}, c, \{\ell\})$
7: **return** ν

Procedure $\mathsf{expand\text{-}HST}(O, c, H)$
Input: O: ontology; c: consequence; H: list of lattice elements
Side effects: modifies $\mathbf{C}$, $\mathbf{H}$, 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: $\quad$ **return** (early path termination ⊗)
3: **if** there exists $\mathcal{M} \in \mathbf{C}$ such that for all $\ell \in \mathcal{M}, h \in H$, $\ell \not\leq h$ and $\ell \not\leq \nu$ **then**
4: $\quad$ $\mathcal{M}' := \mathcal{M}$ (MinLab reuse)
5: **else**
6: $\quad$ $\mathcal{M}' := \mathsf{min\text{-}lab}(O_{\not\leq \nu}, c)$
7: **if** $O_{\not\leq \nu} \models c$ **then**
8: $\quad$ $\mathbf{C} := \mathbf{C} \cup \{\mathcal{M}'\}$
9: $\quad$ $\nu := \bigoplus \{\nu, \bigotimes_{\ell \in \mathcal{M}'} \ell\}$
10: $\quad$ **for** each label $\ell \in \mathcal{M}'$ **do**
11: $\quad\quad$ $\mathsf{expand\text{-}HST}(O_{\not\leq \ell}, c, H \cup \{\ell\})$
12: **else**
13: $\quad$ $\mathbf{H} := \mathbf{H} \cup \{H\}$ (normal termination ⊙)

The procedure $\mathsf{expand\text{-}HST}$ implements the ideas of HST construction for computing all MinAs [11, 12] with additional optimizations that help reduce the search space as well as the number of calls to $\mathsf{min\text{-}lab}$. First notice that each $\mathcal{M} \in \mathbf{C}$ is a MinLab, and hence the infimum of its elements corresponds to the label of some MinA for c. Thus, ν is the supremum of the labels of a set of MinAs for c. If this is not yet the boundary, then there must exist another MinA S whose label is not less than or equal to ν. This in particular means that no element of S may have a label less than or equal to ν, as the label of 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 $\mathsf{expand\text{-}HST}$. 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 O, then c does not follow anymore. Lines 1 to 4 of $\mathsf{expand\text{-}HST}$ are used to reduce the number of calls to the subroutine $\mathsf{min\text{-}lab}$ 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 $O_{\not\leq \nu} \models c$ 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 $O_{\not\leq \nu} \models c$. 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 branch 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. The tree can then be pruned at this node. As an example, consider a subtree reachable from the root by going along the edges ℓ_1, ℓ_2 which has been expanded completely. Then all Hitting Sets of its leaf nodes share the common prefix-path $P = \{\ell_1, \ell_2\}$. Now suppose the tree is expanded by $\mathsf{expand\text{-}HST}(O, c, H)$ with $H = \{\ell_2, \ell_1\}$. The expansion stops with early termination since $P = H$.

The second optimization avoids a possibly expensive call to $\mathsf{min\text{-}lab}$ by *reusing* a previously computed minimal label set. Notice that our only requirement on $\mathsf{min\text{-}lab}$ is that it produces a MinLab. Hence, any MinLab 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 MinLab. If this is the case, the algorithm uses this set instead of computing a new one by calling $\mathsf{min\text{-}lab}$. If we left out the prefix-path condition for early termination, the MinLab reuse condition would still hold. That means leaving out the prefix-path condition leads to no more $\mathsf{min\text{-}lab}$ calls but leads to copying several branches in the tree without obtaining new information.

Before showing that the algorithm is correct, we illustrate its execution through a small example.

Example 4.7. We continue Example 4.4 with the same consequence *SPrIncr*(*ecoCalc*). Figure 2 shows a possible run of the $\mathsf{HST\text{-}boundary}$ algorithm. The algorithm first calls the routine $\mathsf{min\text{-}lab}(O, c)$. Consider that the **for** loop of $\mathsf{min\text{-}lab}$ is executed using the labels in the order $\ell_1, \ell_2, \ell_4, \ell_3, \ell_5$ since Line 5 requires no specific order. Thus, we try first to remove a_1 labelled with ℓ_1. We see that $O_{\neq \ell_1} \not\models c$; hence a_1 is not removed from O, and M_L is updated to $M_L = \{\ell_1\}$. We then see that $O_{\neq \ell_2} \models c$, and thus a_2 is removed from O. Again, $O_{\neq \ell_4} \models c$, so a_4 is removed from O. At this point, $O = \{a_1, a_3, a_5\}$. We

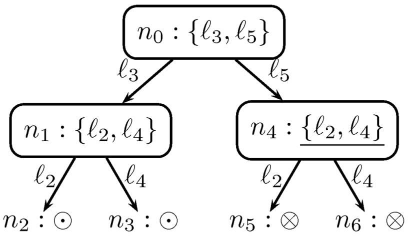

Figure 2: An expansion of the HST method

test then whether $O_{\neq \ell_3} \models c$ and receive a negative answer; thus, ℓ_3 is added to M_L; additionally, since $\ell_3 < \ell_1$, the latter is removed from M_L. Finally, $O_{\neq \ell_5} \not\models c$, and so we obtain $M_L = \{\ell_3, \ell_5\}$ as an output of $\mathsf{min\text{-}lab}$.

The MinLab $\{\ell_3, \ell_5\}$, is used as the root node n_0, setting the value of $\nu = \ell_3 \otimes \ell_5 = \ell_0$. We then create the first branch on the left by removing all the axioms with a label $\leq \ell_3$, which is only a_3, and computing a new MinLab. Assume, for the sake of the example, that $\mathsf{min\text{-}lab}$ returns the MinLab $\{\ell_2, \ell_4\}$, and ν is accordingly changed to ℓ_3. When we expand the tree from this node, by removing all the axioms below ℓ_2 (left branch) or ℓ_4 (right branch), the instance relation c does not follow any more, and hence we have a normal termination, adding the sets $\{\ell_3, \ell_2\}$ and $\{\ell_3, \ell_4\}$ 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 label set of node n_1 works also as a MinLab in this case, and hence it can be reused (MinLab reuse), represented in the figure as an underlined set. The algorithm continues now by calling $\mathsf{expand\text{-}HST}(O_{\not\leq \ell_2}, c, \{\ell_5, \ell_2\})$. At this point, we detect that there is $H' = \{\ell_3, \ell_2\}$ satisfying the first condition of early path termination (recall that $\nu = \ell_3$), and hence the expansion of that branch stops at that point. Analogously, we obtain an early path termination on the second expansion branch of the node n_4. The algorithm then outputs $\nu = \ell_3$, which is the margin-based boundary as computed before.

Theorem 4.8. *Let O and c be such that $O \models c$. Then Algorithm 2 computes the margin-based boundary of c.*

Proof. Let η be the margin-based boundary which, by Lemma 4.1, must exist. Notice first that the procedure $\mathsf{expand\text{-}HST}$ keeps as invariant that $\nu \leq \eta$ as whenever ν is modified, it is only to join it with the infimum of a MinLab (Line 9), which by definition is the label of a MinA and, by Theorem 4.3, is $\leq \eta$. Thus, when the algorithm terminates, we have that $\nu \leq \eta$. Assume now that $\nu \neq \eta$. Then, there must exist a MinA S such that $\lambda_S \not\leq \nu$; in particular, this implies that none of the axioms in S has a label $\leq \nu$ and thus $S \subseteq O_{\not\leq \nu}$. Let $\mathcal{M}_0$ be the MinLab obtained in Line 2 of $\mathsf{HST\text{-}boundary}$. There must then be a $h_0 \in \mathcal{M}_0$ such that $S \subseteq O_{\not\leq h_0}$; otherwise, $\lambda_S \leq \bigotimes_{\ell \in \mathcal{M}_0} \ell \leq \nu$. There will then be a call to the process $\mathsf{expand\text{-}HST}$ with parameters $O_{\not\leq h_0}, c$, and $\{h_0\}$. Suppose first that early path termination is not triggered. A MinLab $\mathcal{M}_1$ is then obtained, either by MinLab reuse

(Line 4) or by a call to min-lab (Line 6). As before, there is a $h_1 \in \mathcal{M}_1$ with $S \subseteq (O_{\not\geq h_0})_{\not\geq h_1}$. Additionally, since $O_{\not\geq h_0}$ does not contain any axiom labelled with h_0, we know $h_0 \notin \mathcal{M}_1$. While iterating this algorithm, we can find a sequence of MinLabs $\mathcal{M}_0, \mathcal{M}_1, \ldots, \mathcal{M}_n$ and labels $h_0, h_1, \ldots, h_n$ such that (i) $h_i \in \mathcal{M}_i$, (ii) $S \subseteq O_{\not\geq h_i}$, and (iii) $h_i \notin \mathcal{M}_j$ for all i, j, $1 \leq i < j \leq n$. In particular, this means that the $\mathcal{M}_i$s are all different, and since there are only finitely many MinLabs, this process must terminate. Let $\mathcal{M}_n$ be the last set found this way. Then, when expand-HST is called with $\mathcal{R} := (((O_{\not\geq h_0})_{\not\geq h_1})_{\ldots})_{\not\geq h_n}, c$ and $H = \{h_1, \ldots, h_n\}$, no new MinLab is found. Suppose first that this is due to a normal termination. Then, $\mathcal{R}_{\not\leq \nu} \not\models c$. But that contradicts the fact that S is a MinA for c since $S \subseteq \mathcal{R}_{\not\leq \nu}$. Hence, it must have finished by early termination.

Early termination can be triggered by two different causes. Suppose first that there is a $H' \in \mathbf{H}$ such that $\{h \in H' \mid h \not\leq \nu\} \subseteq H$. Then it is also the case that, for every $h \in H'$ and $S \subseteq O_{\not\geq h}$ the following holds: if $h \in H$, then $\mathcal{R} \subseteq O_{\not\geq h}$; otherwise, $h \leq \nu$ and hence $O_{\not\leq \nu} \subseteq O_{\not\geq h}$. Let $\mathcal{R}' := \{a \in O \mid \text{there is no } h \in H' \text{ with } \mathsf{lab}(a) \leq h\}$. As $H' \in \mathbf{H}$, it was added after a normal termination; thus, c does not follow from $\mathcal{R}'_{\not\leq \nu}$. As $S \subseteq \mathcal{R}_{\not\leq \nu}$, we obtain once again a contradiction.

The second cause for early path termination is the existence of a prefix-path P with $\{h \in P \mid h \not\leq \nu\} = H$. This means that in a previously explored path we had concluded that $\mathcal{R}_{\not\leq \nu} \models c$, and a new MinLab $\mathcal{M}_{n+1}$ was found. As in the beginning of this proof, we can then compute sets $\mathcal{M}_{n+1}, \ldots, \mathcal{M}_m$ and $h_{n+1}, \ldots, h_m$ $(n < m)$ such that $S \subseteq O_{\not\geq h_i}$ for all $i, 1 \leq i \leq m$ and the $\mathcal{M}_i$s are all different. Hence this process terminates. As before, the cause of termination cannot be normal termination, nor the first condition for early path termination. Thus, there must exist a new $H'' \in \mathbf{H}$ that fulfills the second condition for early termination. As $\mathbf{H}$ is a finite set, and each of its elements is itself a finite list, this process also terminates. When that final point is reached, there are no further causes of termination that do not lead to a contradiction, which means that our original assumption that $\nu \neq \eta$ cannot be true. Hence, ν is the margin-based boundary of c. ☐

4.3. Using Binary Search for Linear Ordering

Assume now that the context lattice $(L, \leq)$ is a linear order, i.e., for any two elements ℓ_1, ℓ_2 of L either $\ell_1 \leq \ell_2$ or $\ell_2 \leq \ell_1$. We show that in this case, the computation of the boundary can be further optimized through a variant of binary search. First, we give a characterization of the boundary in this setting.

Lemma 4.9. *Let O and c be such that $O \models c$. Then the unique boundary of c is the maximal element μ of L_{lab} with $O_{\geq \mu} \models c$.*

Proof. Let μ be the maximal element of L_{lab} such that $O_{\geq \mu} \models c$. Such a maximal element exists since L_{lab} is a finite total order. We need to show that $\ell \leq \mu$ iff $O_{\geq \ell} \models c$. Obviously, $\ell \leq \mu$ implies $O_{\geq \ell} \supseteq O_{\geq \mu}$, and thus $O_{\geq \mu} \models c$ yields $O_{\geq \ell} \models c$. Assume now that $O_{\geq \ell} \models c$. Then the fact that μ is maximal with this property together with the fact that $\leq$ is a linear order implies $\ell \leq \mu$. Thus, μ is a boundary. ☐

Algorithm 3 Compute a boundary by binary search.

Input: O: ontology; c: consequence
Output: ν: (O, c)-boundary

```
if O ⊭ c then
    return no boundary
ℓ := 0_lab; h := 1_lab
while l < h do
    set m, ℓ < m ≤ h, such that |δ(ℓ, m) − δ(m, h)| ≤ 1
    if O_{≥m} ⊨ c then
        ℓ := m
    else
        h := pred(m)
return ν := ℓ
```

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 $O_{\geq \ell} \models c$ 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 $O_{\geq m} \models c$. 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; i.e., $\mathsf{pred}(m)$. This process terminates when the search space has been reduced to a single point, which must be the boundary.

We have thus shown methods to compute a boundary and different optimizations techniques that can be used to improve their efficiency, as will be later shown in Section 6 in an empirical evaluation. Once this boundary has been computed, the knowledge engineer may notice that the consequence belongs to an unwanted set of contexts. In that case, she would like to change the labelling function to correct the contexts to which this consequence belongs. In the next section we will describe methods for finding minimal changes for obtaining the desired boundary.

5. Repairing a Boundary

Just as ontology development and maintenance is an error prone activity, so is the adequate labelling of axioms. Indeed, several seemingly harmless axioms might possibly be combined to deduce knowledge that is considered to be out of the scope of a context. On the other hand, an over-restrictive labelling of axioms may cause harmless or fundamental knowledge to be inaccessible to some contexts.

Example 5.1. We continue Example 2.5. The ontology entails the consequence $c = \mathit{SPrIncr}(\mathit{ecoCalc})$ and the computed boundary of c is ℓ_3 (see Example 4.4), which implies that only for contexts labelled with ℓ_0 and ℓ_3, c is visible. That means 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 customer service employees and development engineers, but rather to customer service employees and customers. In that case, we wish to modify the boundary of c to ℓ_5.

If the knowledge engineer notices that the boundary for a given consequence differs from the desired one, then it would be helpful if she could use automatically generated suggestions for how to modify the labelling function in order to correct this error. This problem can be formalized and approached in several different ways. Here, we assume that the knowledge engineer knows the exact boundary ℓ_g that the consequence c should receive, and we try to find a set S of axioms of minimal cardinality such that, if all the axioms in S are relabelled to ℓ_g, then the boundary of c will be ℓ_g.

Definition 5.2 (Change set)**.** Let O be an ontology, c a consequence, lab a labelling function, $S \subseteq O$ and $\ell_g \in L$ the *goal label*. The *modified assignment* lab_{S,ℓ_g} is given by

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

A sub-ontology $S \subseteq O$ is called a *change set (CS)* for ℓ_g if the boundary for O,c under the labelling function lab_{S,ℓ_g} equals ℓ_g. It is a *minimal CS (MinCS)* if the set is minimal (w.r.t. set inclusion) with this property.

Obviously, the original ontology O is always a change set for any goal label if $O \models c$. However, we are interested in performing minimal changes to the labelling function. Hence, we search first for minimal change sets, and later for a change set of minimum cardinality. A change set of minimum cardinality, or *smallest CS* for short, is obviously also a MinCS. However, the reverse is not necessarily true. A MinCS is minimal with respect to set inclusion but is not necessarily a smallest CS since there might be several MinCS of different cardinality. This is similar to the minimality of MinA (see Definition 2.1), where a MinA is also not necessarily a MinA of minimum cardinality. It follows from results in [22] that it is NP-complete to determine whether the cardinality of a smallest CS is equal to a given natural number, and thus smallest change sets cannot be computed in polynomial time (unless P=NP).

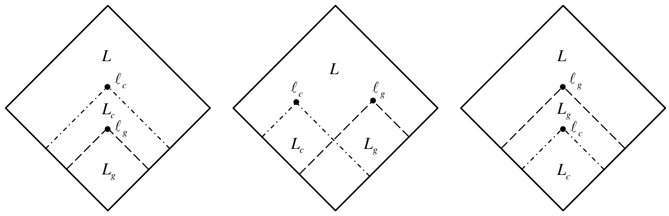

Figure 3: Hide consequence from some contexts (left), allow additional contexts to see consequence (right), and both at the same time (middle)

Let ℓ_g denote the goal label and ℓ_c the margin-based boundary for c. If $\ell_g \neq \ell_c$, we have three cases which are illustrated in Figure 3: either (1) $\ell_g < \ell_c$ (left), (2) $\ell_c < \ell_g$ (right), or (3) ℓ_g and ℓ_c are incomparable (middle). In our example, where $\ell_c = \ell_3$, the three cases can be obtained by ℓ_g being ℓ_0, ℓ_4, and ℓ_5, respectively. The sets L_c and L_g contain the labels defining contexts that can respectively deduce the consequence before and after the label changes. Consider first the case where $\ell_c < \ell_g$. From Theorem 4.3 it follows that any MinA S is a change set for ℓ_g: since $\ell_c < \ell_g$, then for every MinA S', it follows that $\lambda_{S'} < \ell_g$. But then, under the new labelling lab_{S,ℓ_g} it follows that

$$\bigotimes_{a \in S} \mathsf{lab}_{S,\ell_g}(a) = \bigotimes_{a \in S} \ell_g = \ell_g,$$

and hence when the least upper bound of all the labels of all MinAs is computed, we obtain the boundary ℓ_g, as desired.

For the case where $\ell_g < \ell_c$, we will use a similar argument as before, based on a result dual to the result in Theorem 4.3:

Theorem 5.3. *If $S_1, \dots, S_n$ are all diagnoses for O,c, then $\bigotimes_{i=1}^n (\bigoplus_{a \in S_i} \mathsf{lab}(a))$ is a boundary for c.*

Proof. Let first $\ell \in L$ be such that $O_{\geq \ell} \models c$, and let $S_i, 1 \leq i \leq n$ be a diagnosis for O,c. Since $O_{\geq \ell} \models c$, there must be an axiom $a \in S_i$ such that $a \in O_{\geq \ell}$. This means that $\mathsf{lab}(a) \geq \ell$ and hence $\bigoplus_{a \in S_i} \mathsf{lab}(a) \geq \ell$. As this is true for each diagnosis, it holds that $\bigotimes_{i=1}^n (\bigoplus_{a \in S_i} \mathsf{lab}(a)) \geq \ell$.

For the converse, let $\ell \in L$ be a join prime element relative to L_{lab} such that $\ell \leq \bigotimes_{i=1}^n (\bigoplus_{a \in S_i} \mathsf{lab}(a))$. This in particular means that, for every diagnosis S_i for O,c, $\ell \leq \bigoplus_{a \in S_i} \mathsf{lab}(a)$. But since ℓ is join prime relative to L_{lab} and for each $a \in S_i$ $\mathsf{lab}(a)$ is an element of L_{lab}, it holds that there must exist some $a_i \in S_i$ such that $\ell \leq \mathsf{lab}(a_i)$. Thus, $O_{\geq \ell} \cap S_i \neq \emptyset$ for every $i, 1 \leq i \leq n$. Since $S_1, \dots, S_n$ are all diagnoses for O,c, it follows that $O_{\geq \ell} \models c$. □

Notice that, due to the duality between MinAs and diagnoses, if the lattice L is distributive, then the boundary given by this theorem is the same as the margin-based boundary. From Theorem 5.3, it follows that, if $\ell_g < \ell_c$, then every diagnosis is a change set for ℓ_g.

The third case can be addressed using a combination of the previous two approaches: 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 apply the method dealing with 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 diagnosis. The union of these two sets yields a CS.

Unfortunately, the CS computed as described above is not necessarily a MinCS, even if a smallest diagnosis or a smallest MinA is used, as shown in the following example.

Example 5.4. Let O,c and lab be as in Example 2.5 with the consequence *SPrIncr*(*ecoCalc*). We then know that $\ell_c := \ell_3$ is a boundary for O,c. Suppose now that c shall remain visible for those who see it already and additionally made available to customers, i.e. the goal label is $\ell_g := \ell_4$. Since $\ell_c < \ell_g$, we know that any MinA is a change set. Since all MinAs for O,c have exactly three elements, any change set produced this way will have cardinality three. However, $\{a_2\}$ is also a CS. More precisely it is a MinCS.

To understand why the minimality of MinAs is not sufficient for obtaining a MinCS, we can look back to Theorem 4.3. This theorem states that, in order to find a boundary, we need to compute the join of all λ_S, with S a MinA, and λ_S the meet of the labels of all axioms in S. But then, for any axiom $a \in S$ such that $\ell_g \leq \mathsf{lab}(a)$, modifying this label to ℓ_g will have no influence on the result of λ_S. In Example 5.4, 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 relabelled is in fact a_2, which yields the MinCS $\{a_2\}$ shown in the example. Basically, we can consider every axiom $a \in O$ such that $\ell_g \leq \mathsf{lab}(a)$ as *fixed* in the sense that it is superfluous for any change set. Analogously, one can view some of the axioms in a diagnosis as being fixed when trying to compute a change set for decreasing the boundary. For this reason, we will introduce generalizations of MinAs and diagnoses, which we call IAS and RAS, respectively.

Definition 5.5 (IAS, RAS)**.** A *minimal inserted axiom set (IAS)* for ℓ_g is a subset $I \subseteq O_{\not\geq \ell_g}$ such that $O_{\geq \ell_g} \cup I \models c$ and for every $I' \subset I : O_{\geq \ell_g} \cup I' \not\models c$.

A *minimal removed axiom set (RAS)* for ℓ_g is a subset $R \subseteq O_{\not\leq \ell_g}$ such that $O_{\not\leq \ell_g} \setminus R \not\models c$ and for every $R' \subset R$: $O_{\not\leq \ell_g} \setminus R' \models c$.

In the following, we will say that a set S is a *minimal union of a RAS and an IAS* if (i) there exist a RAS R and an IAS I such that $S = R \cup I$ and (ii) for every RAS R' and IAS I', $R' \cup I'$ is not strictly contained in S. The following theorem justifies the use of IAS and RAS when searching for the minimal change sets and a smallest change set.

Theorem 5.6. *Let ℓ_c be a boundary for O,c, ℓ_g the goal label, and $S \subseteq O$. Then, the following holds:*

- *if $\ell_c < \ell_g$ then S is a MinCS iff S is an IAS,*
- *if $\ell_g < \ell_c$ then S is a MinCS iff S is a RAS,*
- *if ℓ_c and ℓ_g are incomparable then S is a MinCS iff S is a minimal union of a RAS and an IAS.*

Proof. We prove only the first result. The other two can be shown analogously. Let first S be a MinCS. From Theorem 4.3 it follows that $S \subseteq O_{\not\geq \ell_g}$ since otherwise S would not be minimal. Since S is a change set, there is a MinA S' such that $\bigotimes_{a \in S'} \mathsf{lab}_{S,\ell_g}(a) \geq \ell_g$; that is, $\mathsf{lab}_{S,\ell_g}(a) \geq \ell_g$ for every $a \in S'$. This means that $S' \subseteq O_{\geq \ell} \cup S$ and thus $O_{\geq \ell} \cup S \models c$. Hence S is an IAS.

Conversely, let S be an IAS; then S is clearly also a change set. If it was not a MinCS, then there would exist an axiom $a \in S$ such that $S \setminus \{a\}$ is also a change set, but as shown before, this would imply that $S \setminus \{a\}$ is an IAS, which violates the minimality condition. □

Obviously, this theorem also yields a direct approach for computing a CS of minimal cardinality.

Corollary 5.7. *Let ℓ_c be a boundary for O,c, and ℓ_g the goal label. Then a CS of minimal cardinality can be found by computing a RAS, an IAS and a union of an IAS and a RAS of minimal cardinality.*

The cardinality of a smallest union of an IAS and a RAS cannot be computed from the cardinalities of a smallest RAS and a smallest IAS since combining the smallest IAS and RAS does not necessarily yield a smallest CS. The following example illustrates this.

Example 5.8. 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 and has cardinality 3. However, combining a smallest IAS and a smallest RAS might yield a MinCS (but not a smallest CS) of cardinality 4.

We now describe how to compute a smallest change set. As in the previous section, we first present the most obvious approach that is based on the computation of all MinAs and diagnoses. Afterwards, we show how this idea can be improved by considering fixed portions of the ontology and computing the set of IAS and RAS, as described before. These methods compute all minimal change sets, from which those with the smallest cardinality can be easily extracted. If one is only interested in a smallest CS, then we can further improve this approach showing that it suffices to compute only partial MinCS by putting a cardinality limit, thus reducing the search space and execution time of our method.

Although we have shown in Example 5.4 that MinAs and diagnoses do not yield MinCS or even smallest CS directly, both of these change sets can still be deduced from the set of all MinAs and diagnoses, as shown by the following lemma.

Lemma 5.9. *Let I (R) be an IAS (RAS) for ℓ_g, then there is a MinA (diagnosis) S such that $I = S \setminus O_{\geq \ell_g}$ ($R = S \setminus O_{\leq \ell_g}$).*

Proof. Let I be an IAS. Then $O_{\geq \ell_g} \cup I \models c$, and hence there is a MinA $S \subseteq O_{\geq \ell_g} \cup I$. As $O_{\geq \ell_g} \cap I = \emptyset$ it follows that $I = S \setminus O_{\geq \ell_g}$. The case for RAS is analogous. □

Lemma 5.9 shows that we can compute the set of all IAS by first computing all MinAs and then removing the set of fixed elements $O_{\geq \ell_g}$ from it. Thus, the most naïve approach for computing a change set of minimum cardinality is to first find all MinAs, then compute the set of all IAS by removing all elements in $O_{\geq \ell_g}$, and finally search for the IAS having the least elements. The same procedure applies to RAS, using diagnoses instead of MinAs.

As explained before, all MinAs can be computed using a HST-based algorithm. Although not stated explicitly in the axiom pinpointing literature, it is clear that the same HST algorithm can be used for computing all diagnoses. The only variant necessary is to have a subroutine capable of computing one such diagnosis, which can be obtained by dualizing the algorithm for computing one MinA (see Algorithms 4 and 5 for an example on how this dualization works). In our experiments, we used this approach as a basis to measure the improvement achieved by the optimizations that will be introduced next.

Naïvely a CS with the lowest cardinality can be found by computing all MinCS and selecting one of minimal size. To find all MinCS, we can use a HST algorithm that uses an auxiliary procedure that computes a single MinCS. For this auxiliary procedure, we can use two subprocedures extracting RAS and IAS, respectively, as evidenced by Theorem 5.6. We now describe an approach for computing a smallest CS directly, which again uses a variant of the HST algorithm.

In Algorithm 4 we present a variation of the logarithmic MinA extraction procedure presented in [21] that is able to compute an IAS or stop once this has reached a size n, in which case it returns the partial IAS computed so far. In this algorithm, the auxiliary procedure halve partitions an ontology into two disjoint subsets of axioms whose difference in cardinality is at most 1. We also show the dual variant for computing a RAS in Algorithm 5.

Given a goal label ℓ_g, if we want to compute an IAS or a partial IAS of size at most n for a consequence c, then we would make a call to extract-partial-IAS($O_{\geq \ell_g}, O_{\not\geq \ell_g}, c, n$). Similarly, a call to extract-partial-RAS($O_{\not\leq \ell_g}, O_{\not\leq \ell_g}, c, n$) yields a RAS of size $\leq n$ or a partial RAS of size exactly

Algorithm 4 Compute a (partial) IAS

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

Global $l := 0, n$
return extract-partial-IAS-r($O_{\mathsf{fix}}, O_{\mathsf{test}}, c$)

Subprocedure extract-partial-IAS-r($O_{\mathsf{fix}}, O_{\mathsf{test}}, c$)

if $n = l$ **then**
  return $\emptyset$
if $|O_{\text{test}}| = 1$ **then**
  $l := l + 1$
  return O_{test}
$S_1, S_2 :=$ halve(O_{test})
if $O_{\text{fix}} \cup S_1 \models c$ **then**
  return extract-partial-IAS-r(O_{fix}, S_1, c)
if $O_{\text{fix}} \cup S_2 \models c$ **then**
  return extract-partial-IAS-r(O_{fix}, S_2, c)
$S'_1 :=$ extract-partial-IAS-r($O_{\text{fix}} \cup S_2, S_1, c$)
$S'_2 :=$ extract-partial-IAS-r($O_{\text{fix}} \cup S'_1, S_2, c$)
return $S'_1 \cup S'_2$

n. The cardinality limit will be used to avoid unnecessary computations when looking for a smallest CS.

With the help of the procedures to extract RAS and IAS, Algorithm 6 describes how to compute a MinCS with a cardinality limit. In the first lines of this algorithm, lbl(c) expresses the margin-based boundary of the consequence c. In order to label a node, we compute a MinCS with extract-partial-MinCS(O, lab, c, ℓ_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 all the axioms in H are removed from the search space to extract the new IAS and RAS. Furthermore, axioms in the IAS computed in Line 4 of this algorithm are considered as *fixed* for the RAS computation. The returned set is a MinCS of size $\leq n$ or a partial MinCS of size n.

Example 5.10. Returning to our running example, suppose now that we want to hide c from development engineers and make it available to customers, i.e. modify the label of consequence c to $\ell_g = \ell_5$. Algorithm 6 starts by making a call to extract-partial-IAS($O_{\geq \ell_5}, O_{\not\geq \ell_5}, c, \infty$).[7] A possible output for this call is $I = \{a_3\}$. We can then call extract-partial-RAS($O_{\not\leq \ell_5} \setminus I, O_{\not\leq \ell_5} \setminus I, c, \infty$), 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 MinCS for ℓ_5.

One of the advantages of the HST algorithm is that the labels of any node are always ensured not to contain

[7]For the sake of this example, we ignore the cardinality limit, as we want to describe only how one MinCS is computed.

Algorithm 5 Compute a (partial) RAS

Procedure $\mathsf{extract\text{-}partial\text{-}RAS}(O_{\mathsf{nonfix}}, O_{\mathsf{test}}, c, n)$

Input: O_{nonfix}: axioms; $O_{\text{test}} \subseteq O_{\text{nonfix}}$: axioms; c: consequence; n: limit

Output: first n elements of a minimal $S \subseteq O_{\text{test}}$ such that $O_{\text{nonfix}} \setminus S \not\models c$

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

Subprocedure $\mathsf{extract\text{-}partial\text{-}RAS\text{-}r}(O_{\mathsf{hold}}, O_{\mathsf{test}}, c)$

if $n = l$ **then**
  return $\emptyset$
if $|O_{\text{test}}| = 1$ **then**
  $l := l + 1$
  return O_{test}
$S_1, S_2 := \mathsf{halve}(O_{\text{test}})$
if $O_{\text{nonfix}} \setminus (O_{\text{hold}} \cup S_1) \not\models c$ **then**
  return $\mathsf{extract\text{-}partial\text{-}RAS\text{-}r}(O_{\text{hold}}, S_1, c)$
if $O_{\text{nonfix}} \setminus (O_{\text{hold}} \cup S_2) \not\models c$ **then**
  return $\mathsf{extract\text{-}partial\text{-}RAS\text{-}r}(O_{\text{hold}}, S_2, c)$
$S_1' := \mathsf{extract\text{-}partial\text{-}RAS\text{-}r}(O_{\text{hold}} \cup S_2, S_1, c)$
$S_2' := \mathsf{extract\text{-}partial\text{-}RAS\text{-}r}(O_{\text{hold}} \cup S_1', S_2, c)$
return $S_1' \cup S_2'$

the label of any of its predecessor nodes. In particular this means that even if we compute a partial MinCS, the algorithm will still correctly find all MinCS that do not contain any of the partial MinCS found during the execution. Since we are interested in finding the MinCS 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-MinCS outputs a set with fewer elements, we are sure that this is indeed a full MinCS, and our new smallest known CS. The HST algorithm will not find all MinCS in this way, but we can be sure that one MinCS with the minimum cardinality will be found. The idea of limiting the cardinality in order to find a smallest MinCS 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 MinCS found further below in the tree will reduce the branching factor of all its predecessors. Hence, the cardinality limit reduces the search space in two dimensions: (1) the computation of a single MinCS is limited to n axioms and (2) only $n - 1$ axioms are expanded from each node. Algorithm 7 is the resulting HST algorithm. The following theorem states that it is correct.

Theorem 5.11. *Let O be an ontology, c a consequence with $O \models c$, and ℓ_g a goal label. If m is the minimum cardinality of all CS for ℓ_g, then Algorithm 7 outputs a CS S such that $|S| = m$.*

Algorithm 6 Compute a (partial) MinCS

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

$\mathrm{i}_I := \ell_g \not< \mathsf{lbl}(c) \wedge O_{\geq \ell_g} \not\models c$
$\mathrm{i}_R := \ell_g \not> \mathsf{lbl}(c) \wedge O_{\not\leq \ell_g} \models c$
return $\mathsf{extract\text{-}partial\text{-}MinCS}(O, \mathsf{lab}, c, \ell_g, \mathrm{i}_I, \mathrm{i}_R, H, n)$

Procedure $\mathsf{extract\text{-}partial\text{-}MinCS}(O, \mathsf{lab}, c, \ell_g, \mathrm{i}_I, \mathrm{i}_R, H, n)$

Input: O, lab: labelled ontology; c: consequence; ℓ_g: goal label; i_I: decision to compute IAS; i_R: decision to compute RAS; H: HST edge labels; n: limit

Output: first n elements of a MinCS $S \subseteq O$

if $\mathrm{i}_I \wedge O_{\geq \ell_g} \cup (O_{\not\geq \ell_g} \setminus H) \not\models c$ or $\mathrm{i}_R \wedge H \models c$ **then**
  return $\emptyset$ (HST normal termination)
if i_I **then**
  $I := \mathsf{extract\text{-}partial\text{-}IAS}(O_{\geq \ell_g}, O_{\not\geq \ell_g} \setminus H, c, n)$
if i_R and $O_{\not\leq \ell_g} \setminus I \models c$ **then**
  $R := \mathsf{extract\text{-}partial\text{-}RAS}(O_{\not\leq \ell_g} \setminus I, O_{\not\leq \ell_g} \setminus (I \cup H), c, n - |I|)$
return $I \cup R$

Proof. The described algorithm outputs a CS since the globally stored and finally returned S is only modified when the output of extract-partial-MinCS has size strictly smaller than the limit n, and hence only when this is indeed a CS itself. Suppose now that the output S is such that $m < |S|$, and let S_0 be a MinCS such that $|S_0| = m$, which exists by assumption. Then, every set obtained by calls to extract-partial-MinCS has size strictly greater than m, since otherwise, S and n would be updated. Consider now an arbitrary set S' found during the execution through a call to extract-partial-MinCS, and let $S_n' := \{a_1, \ldots, a_n\}$ be the first n elements of S'. Since S' is a (partial) MinCS, it must be the case that $S_0 \not\subseteq S_n'$ since every returned MinCS is minimal in the sense that no axiom might be removed to obtain another MinCS. Then, there must be an $i, 1 \leq i \leq n$ such that $a_i \notin S_0$. But then, S_0 will still be a MinCS 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 MinCS, namely S_0 can still be found by expanding the HST, which contradicts the fact that S is the output of the algorithm. $\square$

Example 5.12. Coming back to our running example, suppose that we want to hide c from development engineers, i.e. set the label of c to $\ell_g = \ell_0$. Algorithm 6 first calls $\mathsf{extract\text{-}partial\text{-}RAS}(O_{\not\leq \ell_0}, O_{\not\leq \ell_0}, c, 5)$. 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$(O_{\not\leq \ell_0}, O_{\not\leq \ell_0} \setminus \{a_2\}, c, 2)$, which could yield the RAS $R = \{a_4, a_5\}$. This might be a partial MinCS since its size equals the cardinality limit. The next call extract-partial-RAS$(O_{\not\leq \ell_0}, O_{\not\leq \ell_0} \setminus \{a_2, a_4\}, c, 2)$ yields a smallest $R = \{a_1\}$, and the HST terminates. Notice that if $\{a_1\}$ had been the first MinCS found, the process would have immediately terminated.

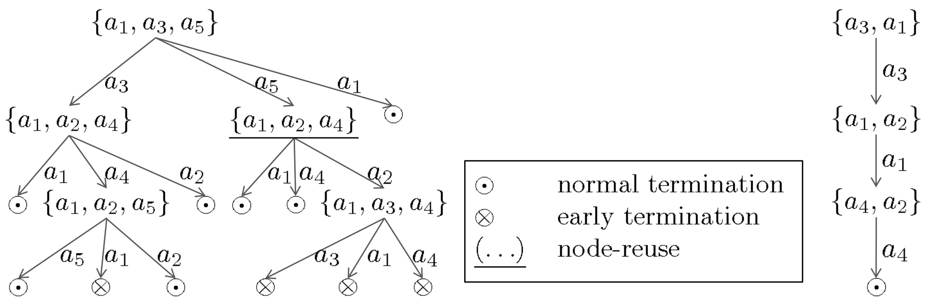

Figure 4: Hitting Set Trees to compute all MinAs (left) and a smallest change set for $\ell_g = \ell_5$ (right)

Algorithm 7 Compute a smallest CS by a HST algorithm

Procedure HST-extract-smallest-CS$(O, \mathsf{lab}, (L, \leq), c, \ell_g)$
Input: O,lab: labelled ontology; $(L, \leq)$: lattice; c: consequence; ℓ_g: goal boundary
Output: a smallest CS S

Global $\mathbf{C}, \mathbf{H}, S := O, n := |O|, c,$
 $\text{is}_I := \ell_g \not\leq \mathsf{lbl}(c) \wedge O_{\geq \ell_g} \not\models c;$
 $\text{is}_R := \ell_g \not\geq \mathsf{lbl}(c) \wedge O_{\not\leq \ell_g} \models c$
expand-HST-CS$(\emptyset)$
return S

Procedure expand-HST-CS(H)
Input: H: list of edge labels
Side effects: modifications to $\mathbf{C}$ and $\mathbf{H}$

if there exists some $H' \in \mathbf{H}$ such that $H' \subseteq H$ **or** H' contains a prefix-path P with $P = H$ **then**
  return (early termination $\otimes$)
else if there exists some $Q' \in \mathbf{C}$ such that $H \cap Q' = \emptyset$ **then**
  $Q := Q'$ (MinCS reuse)
else
  $Q :=$ extract-partial-MinCS$(O, \mathsf{lab}, c, \ell_g, \text{is}_I, \text{is}_R, H, n)$
  if $\emptyset = Q$ **then**
    $\mathbf{H} := \mathbf{H} \cup \{H\}$ (normal termination $\odot$)
    return
  if $|Q| < |S|$ **then**
    $n := |Q|$
    $S := Q$
  $\mathbf{C} := \mathbf{C} \cup \{Q\}$
for the first $(n-1)$ axioms $a \in Q$ **do**
  expand-HST-CS$(H \cup \{a\})$

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. [11, 12, 14]). 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 in our experiments applies these two optimizations.

Example 5.13. We continue Example 2.5 with the same consequence *SPrIncr*(*ecoCalc*). For goal label $\ell_g = \ell_5$, Figure 4 shows the expansion of the HST trees computing all MinAs and all diagnoses (left), in comparison with the one obtained for computing a smallest change set using both optimizations: fixed axioms and cardinality limit (right). Obviously, the number of nodes, the node cardinality and the number of tree expansions is lower.

6. Empirical Evaluation

On large real-world ontologies, we empirically evaluated implementations of the algorithms to (1) compute a boundary for a consequence and (2) repair this boundary if needed. The following sections describe the test data and the test environment first, and then present the empirical results, which show that our algorithms perform well in practical scenarios.

6.1. Test Data and Test Environment

We performed our tests on a PC with 2GB RAM and Intel Core Duo CPU 3.16GHz. We implemented all approaches in Java 1.6 and for convenient OWL file format parsing and reasoner interaction we used the OWL API for OWL 2 [24] in trunk revision 1150 from 21.5.2009.[8]

6.1.1. Context Lattices

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 context lattices $(L_d, \leq_d)$ and $(L_l, \leq_l)$ are similar to ones encountered in real-world applications. The context lattice $(L_d, \leq_d)$, already introduced in Figure 1, was developed and applied in an access policy scenario [19].

[8] Subversion Repository `https://owlapi.svn.sourceforge.net/svnroot/owlapi/owl1_1/trunk`

The context lattice $(L_l, \leq_l)$ is a linear order with 6 elements $L_l = L_d = \{\ell_0, \ldots, \ell_5\}$ with the obvious ordering $\leq_l := \{(\ell_n, \ell_{n+1}) \mid \ell_n, \ell_{n+1} \in L_l \wedge 0 \leq n \leq 5\}$. This lattice could represent an order of trust values as in [25] or dates from a revision history, to name just two applications.

6.1.2. Ontologies, Label Assignment and Reasoners

We used the two ontologies O^{Snomed} and O^{Funct} with different expressivities 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 DL $\mathcal{EL}^{++}$. Our version of 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 [21], i.e., we sampled 0.5% of all concepts in each top-level category of O^{Snomed}. For each sampled concept A, all subsumptions $A \sqsubseteq B$ following from O^{Snomed} with A as subsumee were considered. Overall, this yielded 27,477 subsumptions. Following the ideas of [21], we pre-computed the reachability-based module for each sampled concept A with the reasoner CEL 1.0 [26] and stored these modules. The module for A is guaranteed to contain all axioms of any MinA and any diagnosis, thus also any IAS and RAS, for each subsumption $A \sqsubseteq B$ with A the considered subsumee. This module was then used as the start ontology when considering subsumptions with subsumee A, rather than using the complete ontology.

The OWL ontology O^{Funct} has been designed for functional descriptions of mechanical engineering solutions and was presented in [27, 28]. 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 (concept assertions).

To obtain labelled ontologies, axioms in both ontologies received a random label assignment of elements from the set $L_l = L_d$. Our test suite provides a great variety of cases: consequences with many or few MinAs, with very large or very small MinAs, cases with a high or low boundary, etc. Hence, our results should not differ greatly if a different label assignment is used. As black-box subsumption and instance reasoner we used Pellet 2.0 [29], 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 that is based on the algorithm described in [30].

6.1.3. Test Setting for Computing a Boundary

The boundary computation with full axiom pinpointing (FP) uses log-extract-MinA (Algorithm 2 from [21], which is identical to Algorithm 8 from [12]) and the HST based HST-extract-all-MinAs procedure (Algorithm 9 from [12]). The set of extracted MinAs is then used to calculate the label of the consequence. We stop the execution after 10 MinAs have been found in order to limit the runtime; thus, some of the labels found may not be the final result. The boundary computation with label-optimized axiom pinpointing (LP) with min-lab and HST-boundary are implementations of Algorithm 1 and Algorithm 2. The boundary computation with binary search for linear ordering (BS in the following) implements Algorithm 3. We tested 8 combinations resulting from the 2 ontologies O^{Snomed} and O^{Funct}, with the two approaches FP and LP over the lattice $(L_d, \leq_d)$ and the two approaches LP and BS over the lattice $(L_l, \leq_l)$.

6.1.4. Test Setting for Repairing a Boundary

We tested repairing access restrictions to implicit knowledge in the following setting. We took the computed boundary ℓ_c of each consequence c of the ontologies from the first experiment and then computed the MinCS to reach the goal boundary ℓ_g which is constantly ℓ_3 in all experiments. Consequences were not considered if $\ell_c = \ell_g$. Thus, from the 716 consequences in O^{Funct}, we have 415 remaining with context 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 context lattice $(L_d, \leq_d)$ and 25,897 with $(L_l, \leq_l)$. The MinCS computation with FP uses the procedures log-extract-MinA and the HST based HST-extract-all-MinAs, implemented by the algorithms mentioned above. The MinCS computation with extract-partial-MinCS and the smallest CS computation with HST-extract-smallest-CS including optimizations for fixed axioms and cardinality limit are implementations of Algorithm 6 and Algorithm 7. The required IAS and RAS extraction with extract-partial-IAS, extract-partial-RAS are implementations of Algorithms 4 and 5, respectively. We stop after 10 MinAs (or respectively MinCS or partial MinCS) have been found in order to limit the runtime; hence there might be no computed MinCS at all or a non-smallest MinCS returned. We tested 12 combinations resulting from the two ontologies O^{Snomed} and O^{Funct}, two context lattices $(L_d, \leq_d)$ and $(L_l, \leq_l)$ and three algorithm variants (FP, fixed axioms, and fixed axioms in combination with cardinality limit). Running the fixed axioms optimization without cardinality limit can be done easily by skipping Line 11 in Algorithm 7.

6.2. Experimental Results

Our experiments show that our algorithms perform well on practical large-scale ontologies. In the following we describe our empirical results for each of the two discussed tasks with labelled ontologies, i.e. computing a boundary to discover access restrictions to a given consequence and repair the boundary of a single consequence by changing axiom labels.

6.2.1. Computing a Boundary

The results for boundary computation by FP and LP, using lattice $(L_d, \leq_d)$ and the two ontologies O^{Snomed} and O^{Funct} are given in Table 1. The table is divided into two parts. The upper part contains a set of consequences that are "easy," in the sense that each consequence has fewer than 10 MinAs. This contains 21,001 subsumptions from O^{Snomed} and 307 consequences from O^{Funct}. The lower part contains a set of consequences that are "hard," in the sense that each consequence has at least 10 MinAs. This contains 6,476 subsumptions from O^{Snomed} and 409 consequences from O^{Funct}.

While LP computed the boundary for each consequence following from the easy and the hard set, FP computed the boundary for each consequence following from the easy but not for each following from the hard set. As described above, we stop the execution after 10 MinAs have been found. A label computed for a consequence following from the hard set, called non-final label, might be lower than the boundary since there might be further MinAs providing a higher label. For a practical system, a lower label puts an unnecessarily strong access restriction to a consequence, resulting in an overrestrictive policy.

For the easy set of O^{Snomed}, the overall labelling time for all 21,001 subsumptions with FP was 50.25 minutes. For LP it was 1.50 minutes, which means that LP was about 34 times faster than FP. For the hard set of O^{Snomed}, the non-final labels of FP were identical to the boundaries of LP 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. FP took 2.5 hours without final results, whereas LP took 0.6% (a factor of 155) of that time and returned final results after 58 seconds. We started a test series limiting runs of FP to <30 MinAs, which did not terminate after 90 hours, with 1,572 labels successfully computed and 30 subsumptions skipped since they had ≥30 MinAs. Interestingly, in both the easy and the hard set, LP rarely takes advantage of the optimizations early termination and reuse, which might be due to the simple structure of the lattice.

Similar results have been observed for the easy and the hard sets of O^{Funct}. 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 boundaries of LP. Although the ontology is quite small, LP again performs much better than FP. The reason could be that, in this ontology, consequences frequently have a large set of MinAs.

For a system designer, a question to decide could be to use either (a) our approach of keeping one large ontology with labelled axioms and pre-compute all consequences[9] and their labels or (b) the naïve approach of managing separate ontologies and computing all consequences of each separate ontology independently. We can make the following rough estimate while we assume that the lattice is nonlinear but the details of its structure would not have any influence. Based on our test results with O^{Snomed}, an estimate for the time needed to compute a label for all of the more than 5 million subsumptions in O^{Snomed} with LP would be $2.47 \cdot \frac{5 \cdot 10^6}{27477} \approx 449$ minutes. Assuming 20 minutes to compute all consequences with the current CEL reasoner [12], our approach to compute and label all consequences would be as expensive as computing all consequences $\frac{20+449}{20} \approx 23$ times. However, two remarks need to be made here:

1. Taking the fast computation time of 20 minutes, achieved by CEL is to some extent unfair, since the slower (see [12] for a comparison) Pellet is used in our experiments for reasons explained above. However, at the time of these experiments, Pellet fails to classify the complete O^{Snomed} because of memory exhaustion. For this reason we process only the reachability-based modules with Pellet and not the complete O^{Snomed}, as described above. On average, our reachability-based modules contain 53.21 axioms, with a maximum size of 146 axioms; that is, their size is 0.01% (maximum 0.04%) of the total size of O^{Snomed} [21]. Presumably, the realistic ratio is actually below 23.
2. The preparation step computing the reachability-based modules as described above took < 0.21 seconds [21] and can be neglected here.

Our approach is as expensive as computing 23 views if we assume that computing all consequences for each of the views requires 20 minutes. For incomparable user labels, e.g. representing user roles which do not inherit permissions from each other while one user can have several roles, already the considerably low number of 5 incomparable user labels implies $2^5 = 32$ sub-ontologies (views), and thus our approach is already faster. For fewer user labels, the naïve approach is faster, but then separate subsumption hierarchies would need to be stored for each of the views, whereas in our approach only one labelled hierarchy needs to be stored. Based on our test results with O^{Funct}, a similar estimate can be made. Computing all consequences requires 5 seconds and labelling all consequences with LP requires 146 seconds. In this case our approach is as expensive as computing all consequences 30 times. Again with 5 or more user labels, our approach is faster.

In statistics, histograms are often used to roughly assess probability distributions. The range of values on the x-axis is divided into non-overlapping intervals and the y-axis provides the number of observations. The histogram in

[9]When we say "all consequences" we always mean "all consequences the user is interested in." These might be, e.g., all concept assertions of named individuals to named concepts and all subsumptions between two named concepts. This restriction is necessary since already from simple axioms, infinitely many nonequivalent consequences may follow, e.g. from $M \sqsubseteq \exists l.M$ it follows that $M \sqsubseteq \exists l.M, M \sqsubseteq \exists l.(\exists l.M)$, etc.

			♯early termination	♯reuse	♯calls to extract MinA (MinLab)	♯MinA (♯MinLab)	♯axioms (♯labels) per MinA (MinLab)	Lattice operations time in ms	Total labelling time in ms
O^{Snomed} easy	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} easy	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^{Snomed} hard	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} hard	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 1: Boundary computation by FP vs. LP, using lattice $(L_d, \leq_d)$ and two ontologies with an easy and a hard set of consequences

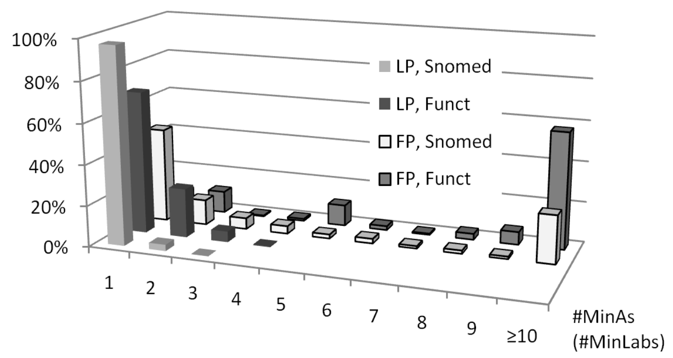

Figure 5: Histogram of required ♯MinAs (♯MinLabs) to compute a boundary (respectively non-final label)

Figure 5 shows, for all 4 combinations of the 2 ontologies and the 2 computation methods, the number of MinAs (respectively MinLabs) required to compute a boundary or non-final label. From this histogram and also from Table 1, 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.

The histograms in Figure 6 compare the distribution of time needed with FP vs. LP to compute a boundary of a consequence from the union of the above described easy and hard sets. The required time is given on the logarithmic x-axis, where the number below each interval defines the maximum contained value. As can be seen, FP takes more time than LP in general. Note that moving to the left on the x-axis means a relatively high performance improvement, due to the logarithmic scale. It can be further seen that LP covers a few intervals while FP covers more. This indicates that FP has a higher variability and the standard deviation values in Table 1 confirm this.

Table 2 provides results for LP and BS using the total order $(L_l, \leq_l)$ as context 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. Interestingly, labelling all consequences of O^{Funct} or all consequences of O^{Snomed} takes roughly the same time, perhaps due to a trade-off between ontology size and expressivity. Roughly, BS is twice as fast (factor 1.7 with O^{Snomed}, 1.9 with O^{Funct}) compared to LP.

Above, we already discussed the decision of a system designer whether to use our approach or the naïve approach for nonlinear lattices. Based on our test results a similar estimate can be made for linear lattices. Labelling all consequences of O^{Snomed} would require $\frac{77.1}{60} \cdot \frac{5 \cdot 10^6}{27477} \approx 234$ minutes. Similar to the explanation above, our approach is as expensive as computing all consequences $\frac{20+234}{20} \approx 13$

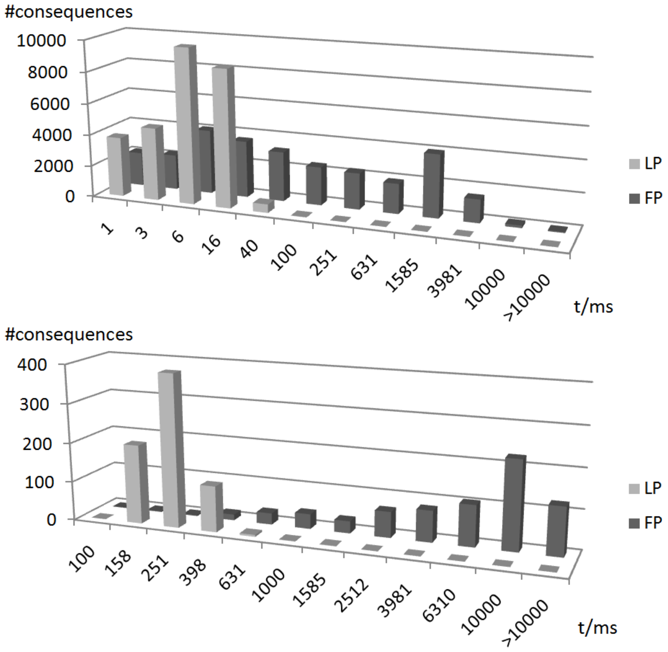

Figure 6: Histograms of time needed to compute a consequence's boundary in O^{Snomed} (upper part) and O^{Funct} (lower part) with the methods FP vs. LP

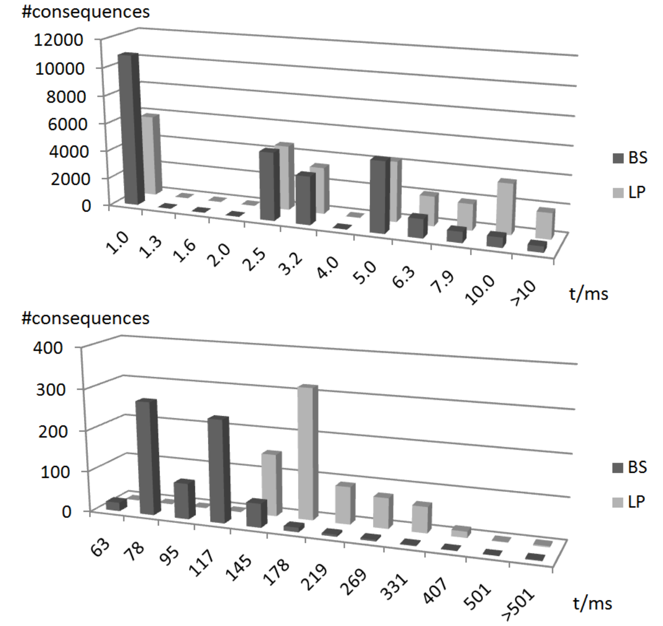

Figure 7: Histograms of time needed to compute a boundary for a consequence in O^{Snomed} (upper part) and O^{Funct} (lower part) with the methods BS vs. LP

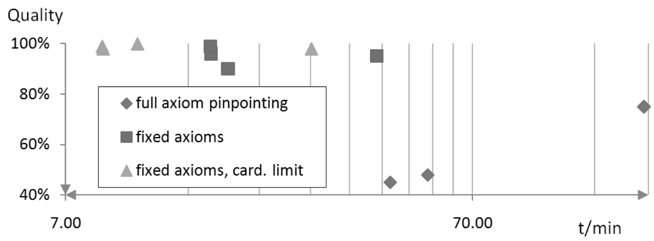

Figure 8: Time-quality diagram comparing variants to compute a smallest CS

times, i.e. with 13 or more context labels our approach is faster. For O^{Funct}, our approach is as expensive as computing all consequences $\frac{68.8}{5} \approx 14$ times, i.e. with 14 or more user labels our approach is faster.

The histograms in Figure 7 compare the distribution of time needed to compute a boundary with BS and LP. Again the required time is given on the logarithmic x-axis. They show that the performance gain with BS over LP is higher with O^{Funct} compared to O^{Snomed}, as discussed already. They further show that there is no clear winner with respect to variability, as was the case comparing FP with LP; Table 2 confirms that observation.

6.2.2. Repairing a Boundary

Table 3 contains results for the 4 combinations of the 2 ontologies and the 2 context lattices. For each of them we tested 3 variants, leading to 12 test series overall. As described above, we limit the number of computed MinAs and MinCS to 10, so our algorithms might not find any, or not a smallest change set before reaching the limit. We measure the quality of the presented variants given this limitation at execution time in the following sense. Table 3 lists the ratio of correct solutions where at least 1 correct MinCS was computed, and the ratio of optimal solutions where the limit was not reached during the computation and thus yielded the smallest change set possible. Notice however that the ratio of cases with the smallest change set successfully computed might be higher, including those where the limitation was reached but the smallest change set was already found.

Figure 8 depicts a time-quality diagram of all variants from Table 3, where quality is the ratio of correct solutions multiplied by the ratio of optimal ones. Obviously, a desirable variant is in the upper left corner yielding maximum quality in minimal time. It can be seen that FP is clearly outperformed by our optimizations. The experiment shows that fixed axioms and cardinality limit, especially in their combination, are optimizations yielding significantly higher quality and lower runtime.

Instead of providing histograms of time needed to repair a boundary for a consequence, we provide the cumulative distribution in Figures 9 and 10. The difference to the histograms is that not discrete intervals, but instead the continuous spectrum of time needed is depicted, and the number of consequences is cumulated over time until it reaches the number of all considered consequences. For this reason, the maximum values on the y-axis are 415, 474, 23695 and 25897. The reason for those numbers of consequences has been explained above. The x-axis is again logarithmic, as it has been the case with the previous histograms. It can be seen that in general a consequence from O^{Snomed}

		LP							BS	
		♯early term.	♯reuse	♯calls to extract MinLab	♯MinLab	♯labels per MinLab	Lattice oper. time	Total labelling time	Iterations	Total labelling 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

Table 2: Boundary computation by LP vs. 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)

Ont.	Lattice	Variant	Runtime limit per goal	Time in minutes	Ratio of correct solutions	Ratio of optimal solutions
O^{Funct}	$(L_d, \leq_d)$	FP	$\leq$ 10 MinA	44.05	96%	47%
		fixed axioms	$\leq$ 10 MinCS	17.56	100%	90%
		fixed axioms, card. lim.	$\leq$ 10 (partial) MinCS	8.65	100%	98%
	$(L_l, \leq_l)$	FP	$\leq$ 10 MinA	54.46	98%	49%
		fixed axioms	$\leq$ 10 MinCS	15.97	100%	96%
		fixed axioms, card. lim.	$\leq$ 10 (partial) MinCS	8.61	100%	99%
O^{Snomed}	$(L_d, \leq_d)$	FP	$\leq$ 10 MinA	184.76	100%	75%
		fixed axioms	$\leq$ 10 MinCS	15.87	100%	99%
		fixed axioms, card. lim.	$\leq$ 10 (partial) MinCS	10.51	100%	100%
	$(L_l, \leq_l)$	FP	$\leq$ 10 MinAs	185.35	100%	75%
		fixed axioms	$\leq$ 10 MinCS	40.83	100%	95%
		fixed axioms, card. lim.	$\leq$ 10 (partial) MinCS	28.14	100%	98%

Table 3: Results comparing variants to compute a smallest CS

is repaired much faster than one from O^{Funct}. Interestingly, even at the start of the logarithmic x-axis, some consequences are repaired already. In our test result log, the reported time for several consequences was even 0 ms. The reason is Java's limitation with respect to measuring fractions of milliseconds. It can be seen, as already known from Table 3, that for the two diagrams of O^{Snomed} and the upper right diagram of O^{Funct}, most of the consequences are repaired roughly one order of magnitude faster with both of our optimizations enabled compared to the naïve FP approach.

7. Conclusions

We have presented a general approach for defining and reasoning with contexts in large scale ontologies. Our approach assumes that every axiom in the ontology is labelled with an element of a context lattice. The different contexts of this ontology are then expressed by elements of this lattice: each[10] such element ℓ yields a sub-ontology consisting of the axioms whose label is greater or equal to ℓ. This general framework can be instantiated to any notion of context that can be expressed using a context lattice. Examples of such instances are access control to ontological knowledge, trust management, provenance, and granularity, among many others. The main advantage of this approach is that it allows the knowledge engineer to maintain only one large ontology that is usable in all different contexts, rather than separate sub-ontologies for each context.

We have shown that we can extend the labelling function to arbitrary consequences of the ontology, in the sense that every implicit consequence can be assigned a label, called its boundary, which fully characterizes the set of all contexts in which the consequence can be derived. In this way, one can solve reasoning tasks for all contexts simultaneously. For instance, if one is interested in computing the concept hierarchy, one can compute the boundary for each of the subsumption relations between concept names holding in the full ontology. From this information one can easily deduce the concept hierarchy derivable in each of the contexts. Our algorithms are inspired by ideas from

[10]For technical reasons, only elements that are join prime relative to the axiom labels can be used as context labels.

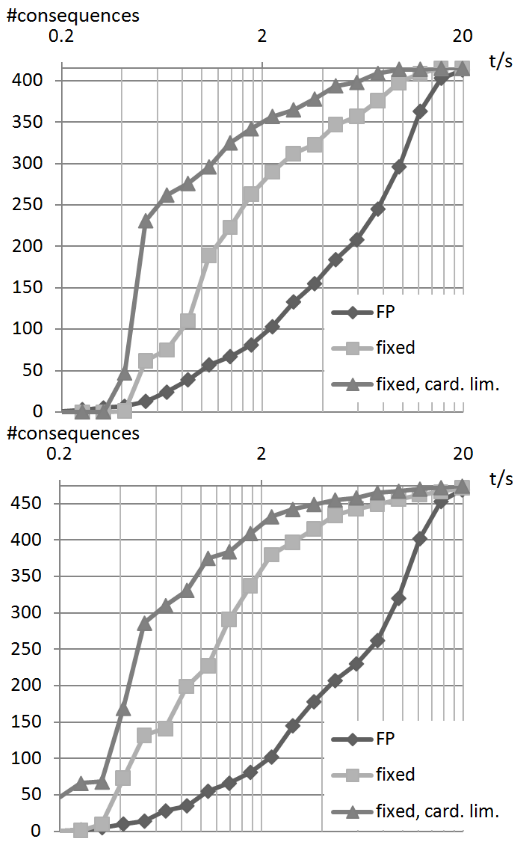

Figure 9: Cumulative distribution of time needed to repair a boundary in O^{FUNCT} with lattices $(L_d, \leq_d)$ and $(L_l, \leq_l)$

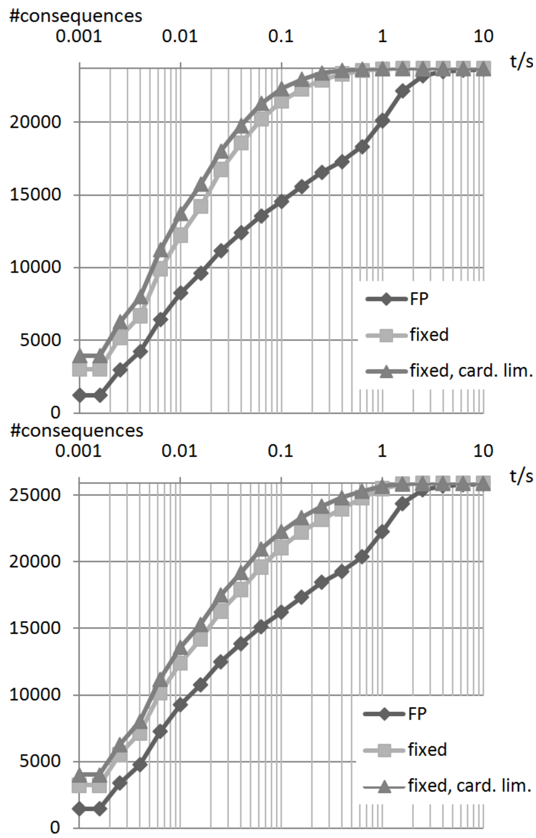

Figure 10: Cumulative distribution of time needed to repair a boundary in O^{SNOMED} with lattices $(L_d, \leq_d)$ and $(L_l, \leq_l)$

axiom pinpointing, but are optimized to take advantage of the additional information provided by the lattice and the labelling function.

The principal assumption for our framework is that the axioms have been assigned the correct label. However, assigning these labels to all the axioms in the ontology is an error-prone task. Indeed, a small change in the labelling function may hide relevant consequences from a context, or apparently innocuous axioms may in combination produce consequences not intended to be visible in some contexts. To alleviate this problem, we propose a method for finding minimal changes that should be made to the labelling function, in order to repair the boundary of a given consequence. Here, we use two different notions of minimality. First we consider the task of finding all the minimal (w.r.t. set inclusion) sets of axioms that need to be relabelled to correct the boundary, i.e. all minimal change sets. This information can be then given to the knowledge engineer, who can decide which change set is the best option. However, the large number of minimal change sets available may be overwhelming for the knowledge engineer, and hence finding only one of these sets of minimal size can sometimes be more desirable. We show how to improve the algorithms to find only one change set of minimal cardinality, by including a cardinality limit in the Hitting Set Tree construction.

An interesting property of our framework is that it is independent of the ontology language used. That is, it can be used for finding the boundary of subsumption relations in the DL $\mathcal{SROIQ}(\mathbf{D})$, as well as unsatisfiability of concepts w.r.t. acyclic TBoxes in $\mathcal{ALC}$, for example. This is the case since all our algorithms follow a black-box approach. This means that they do not depend on a specific implementation of a reasoner, but can be used together with any reasoner available. In particular, this allows us to take advantage of the many optimizations of state-of-the-art DL reasoners.

We have evaluated implementations of our algorithms empirically, using two large-scale ontologies that are used in real-life scenarios, and a context lattice developed for an access control application. Our experimental results show that our implementations perform well in practical scenarios with large-scale ontologies.

For computing a boundary for a consequence, the full axiom pinpointing approach is clearly outperformed by the label-optimized axiom pinpointing approach, which is faster up to a factor of 155. For the special case where the context lattice is a total order, label-optimized axiom pinpointing is itself outperformed by the Binary Search approach by roughly a factor of 2. We have provided an estimate from the point of view of a system designer, comparing our approach of labelled ontologies and consequences to a naïve approach of reasoning over separate ontologies. It showed that our approach is faster when more than four incomparable user labels are present in a nonlinear lattice or when more than 12 user labels are present in a linear lattice.

For repairing a boundary, which is only possible by changing axiom labels, our experiments show that our algorithms and optimizations yield tangible improvements in both the execution time and the quality of the proposed smallest CS defining a new axiom labelling. In order to compute a CS of minimal cardinality, the approach of computing all MinAs is outperformed up to a factor of 12 by our optimized approach of computing IAS and RAS. Limiting cardinality further reduces computation time by up to a factor of 2. In combination we observed a performance increase by up to a factor of 18. But not only performance is improved, at the same time both optimizations increase the quality of the computed smallest CS under limited resources at runtime.

As future work, we plan to extend our methods for repairing a boundary. In the current setting, the knowledge engineer must specify the exact boundary that the consequence must receive. However, it is sometimes desirable to set a constraint in the form of an inequality, for instance, specifying that the consequence should be visible from a given context, but without restricting its visibility w.r.t. other contexts. Additionally, we plan to explore other notions of minimality of the change sets, like the distance that a label is moved, the number of users affected, or the number of consequences that receive a new boundary.

[1] F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, P. F. Patel-Schneider (Eds.), The Description Logic Handbook: Theory, Implementation and Applications, 2nd Edition, Cambridge University Press, 2007.

[2] G. Qi, J. Z. Pan, A tableau algorithm for possibilistic description logic, in: J. Domingue, C. Anutariya (Eds.), Proceedings of the 3rd Asian Semantic Web Conference (ASWC 2008), Vol. 5367 of Lecture Notes in Computer Science, Springer-Verlag, 2008, pp. 61–75.

[3] M.-J. Lesot, O. Couchariere, B. Bouchon-Meunier, J.-L. Rogier, Inconsistency degree computation for possibilistic description logic: An extension of the tableau algorithm, in: Proceedings of the Annual Meeting of the North American Fuzzy Information Processing Society (NAFIPS 2008), IEEE Computer Society Press, 2008, pp. 1–6.

[4] S. Schlobach, R. Cornet, Non-standard reasoning services for the debugging of description logic terminologies, in: Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI 2003), 2003, pp. 355–362.

[5] T. Meyer, K. Lee, R. Booth, J. Z. Pan, Finding maximally satisfiable terminologies for the description logic $\mathcal{ALC}$, in: Proceedings of the 21st National Conference on Artificial Intelligence (AAAI 2006), AAAI Press/The MIT Press, 2006.

[6] A. Kalyanpur, B. Parsia, E. Sirin, J. A. Hendler, Debugging unsatisfiable classes in OWL ontologies, Journal of Web Semantics: Science, Services and Agents on the World Wide Web 3 (4) (2005) 268–293.

[7] F. Baader, R. Peñaloza, Axiom pinpointing in general tableaux, Journal of Logic and Computation 20 (1) (2010) 5–34.

[8] F. Baader, R. Peñaloza, Automata-based axiom pinpointing, in: A. Armando, P. Baumgartner, G. Dowek (Eds.), Proceedings of the 4th International Joint Conference on Automated Reasoning (IJCAR 2008), no. 5195 in Lecture Notes in Artificial Intelligence, Springer, 2008, pp. 226–241.

[9] A. Jain, C. Farkas, Secure resource description framework: an access control model, in: Proceedings of the 11th ACM symposium on Access control models and technologies (SACMAT 2006), ACM, New York, NY, USA, 2006, pp. 121–129. doi:http://doi.acm.org/10.1145/1133058.1133076.

[10] G. Flouris, I. Fundulaki, P. Pediaditis, Y. Theoharis, V. Christophides, Coloring RDF triples to capture provenance, in: A. Bernstein, D. R. Karger, T. Heath, L. Feigenbaum, D. Maynard, E. Motta, K. Thirunarayan (Eds.), Proceedings of the 8th International Semantic Web Conference (ISWC 2009), Vol. 5823 of Lecture Notes in Computer Science, Springer, 2009, pp. 196–212.

[11] A. Kalyanpur, B. Parsia, M. Horridge, E. Sirin, Finding all justifications of OWL DL entailments, in: Proceedings of the 6th International Semantic Web Conference and 2nd Asian Semantic Web Conference (ISWC+ASWC 2007), Vol. 4825 of Lecture Notes in Computer Science, Springer-Verlag, Busan, Korea, 2007, pp. 267–280.

[12] B. Suntisrivaraporn, Polynomial-time reasoning support for design and maintenance of large-scale biomedical ontologies, Ph.D. thesis, Technische Universität Dresden, available at `http://nbn-resolving.de/urn:nbn:de:bsz:14-ds-1233830966436-59282` (2008).

[13] G. Qi, J. Z. Pan, Q. Ji, Extending description logics with uncertainty reasoning in possibilistic logic, in: K. Mellouli (Ed.), Proceedings of the 9th European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty (ECSQARU 2007), Vol. 4724 of Lecture Notes in Computer Science, Springer-Verlag, 2007, pp. 828–839.

[14] F. Baader, M. Knechtel, R. Peñaloza, A generic approach for large-scale ontological reasoning in the presence of access restrictions to the ontology's axioms, in: A. Bernstein, D. R. Karger, T. Heath, L. Feigenbaum, D. Maynard, E. Motta, K. Thirunarayan (Eds.), Proceedings of the 8th International Semantic Web Conference (ISWC 2009), Vol. 5823 of Lecture Notes in Computer Science, 2009, pp. 49–64.

[15] M. Knechtel, R. Peñaloza, A generic approach for correcting access restrictions to a consequence, in: L. Aroyo, G. Antoniou, E. Hyvönen, A. ten Teije, H. Stuckenschmidt, L. Cabral, T. Tudorache (Eds.), Proceedings of the 7th Extended Semantic Web Conference (ESWC 2010), Vol. 6088 of Lecture Notes in Computer Science, 2010, pp. 167–182.

[16] THESEUS Joint Research, THESEUS application scenario PROCESSUS, `http://www.theseus.joint-research.org/processus-en/` (2010).

[17] B. A. Davey, H. A. Priestley, Introduction to Lattices and Order, 2nd Edition, Cambridge University Press, 2002.

[18] B. Lampson, Protection, in: Proceedings of the 5th Annual Princeton Conference on Information Sciences and Systems, 1971, pp. 437–443.

[19] F. Dau, M. Knechtel, Access policy design supported by FCA methods, in: F. Dau, S. Rudolph (Eds.), Proceedings of the 17th International Conference on Conceptual Structures (ICCS 2009), Vol. 5662 of Lecture Notes in Computer Science, 2009, pp. 141–154.

[20] B. Ganter, R. Wille, Formal Concept Analysis - mathematical foundations, Springer, 1999.

[21] F. Baader, B. Suntisrivaraporn, Debugging SNOMED CT using axiom pinpointing in the description logic $\mathcal{EL}^+$, in: Proceedings of the International Conference on Representing and Sharing Knowledge Using SNOMED (KR-MED 2008), Phoenix, Arizona, 2008.

[22] F. Baader, R. Peñaloza, B. Suntisrivaraporn, Pinpointing in the description logic $\mathcal{EL}$, in: Proceedings of the 30th German Conference on Artificial Intelligence (KI 2007), Vol. 4667 of Lecture Notes in Artificial Intelligence, Springer-Verlag, Osnabrück, Germany, 2007, pp. 52–67.

[23] R. Reiter, A theory of diagnosis from first principles, Artificial

Intelligence 32 (1) (1987) 57–95.

[24] M. Horridge, S. Bechhofer, The owl api: A java api for owl ontologies, Semantic Web 2 (1) (2011) 11–21.

[25] S. Schenk, On the semantics of trust and caching in the Semantic Web, in: Proceedings of the 7th International Semantic Web Conference (ISWC 2008), 2008, pp. 533–549.

[26] F. Baader, C. Lutz, B. Suntisrivaraporn, CEL—a polynomial-time reasoner for life science ontologies, in: U. Furbach, N. Shankar (Eds.), Proceedings of the 3rd International Joint Conference on Automated Reasoning (IJCAR 2006), Vol. 4130 of Lecture Notes in Artificial Intelligence, Springer-Verlag, 2006, pp. 287–291.

[27] A. Gaag, A. Kohn, U. Lindemann, Function-based solution retrieval and semantic search in mechanical engineering, in: Proceedings of the 17th International Conference on Engineering Design (ICED 2009), 2009.

[28] A. Gaag, Entwicklung einer Ontologie zur funktionsorientierten Lösungssuche in der Produktentwicklung, Ph.D. thesis, Munich University of Technology (2010).

[29] E. Sirin, B. Parsia, B. C. Grau, A. Kalyanpur, Y. Katz, Pellet: A practical OWL-DL reasoner, Journal of Web Semantics 5 (2) (2007) 51–53.

[30] F. Baader, S. Brandt, C. Lutz, Pushing the $\mathcal{EL}$ envelope, in: Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI 2005), Morgan-Kaufmann Publishers, Edinburgh, UK, 2005.