Elimination of Redundancy in Ontologies

Stephan Grimm and Jens Wissmann

FZI – Research Center for Information Technology, Karlsruhe, Germany
{grimm,wissmann}@fzi.de

Abstract. Ontologies may contain redundancy in terms of axioms that logically follow from other axioms and that could be removed for the sake of consolidation and conciseness without changing the overall meaning. In this paper, we investigate methods for removing such redundancy from ontologies. We define notions around redundancy and discuss typical cases of redundancy and their relation to ontology engineering and evolution. We provide methods to compute irredundant ontologies both indirectly by calculating justifications, and directly by utilising a hitting set tree algorithm and module extraction techniques for optimization. Moreover, we report on experimental results on removing redundancy from existing ontologies available on the Web.

1 Introduction

Ontologies are subject to an engineering lifecycle as any other technical artifact in an information system, and methods for their development and maintenance over time are researched under the label of *ontology evolution*. Ontologies also provide features for automated deduction that allow for deriving implicit knowledge from its explicitly stated axioms. These features give rise to a notion of redundancy in ontologies, meaning the presence of axioms that implicitly follow from other axioms present in the ontology. One important technique in an ontology evolution toolbox is the automated identification and elimination of such redundancy on demand to provide a means for the consolidation and compactification of ontologies in the course of their development.

Although there might be scenarios where redundancy can arguably be a desirable feature, we consider cases in which ontology engineers want to identify and eliminate redundancy to consolidate and clean up their ontologies for the sake of maintenance. Techniques for keeping ontologies irredundant have many use-cases in ontology evolution scenarios as redundancy can cause various problems. Due to the non-locality of redundant information distributed over several axioms in different places, an ontology can be hard to understand, to maintain and to be split into separate independent modules. Moreover, deletion or update of axioms might be semantically ineffective in case they are implied by others. Furthermore, in scenarios where techniques of automated knowledge acquisition are utilized, redundancy elimination can reduce the amount of acquired statements for optimising local storage and reasoning. Similarly, it can be used for undoing materialization.

G. Antoniou et al. (Eds.): ESWC 2011, Part I, LNCS 6643, pp. 260–274, 2011.

Elimination of redundancy can also support other techniques for processing ontologies. While module extraction techniques typically single out relevant parts of an ontology for reuse, they do not ensure minimality in axioms, whereas a combination with redundancy elimination allows for obtaining irredundant modules. Another use case for irredundant ontologies is the handling of access rights to axioms of an ontology. There, the problem of concealed axioms being derived by accessible ones can be avoided when using irredundant ontologies.

In this paper, we study methods for the elimination of redundancy in OWL DL ontologies. We define various notions around redundancy in the axioms of an ontology and identify typical cases in which redundancy is introduced unanticipatedly in the course of an ontology's evolution over time. We provide two methods for computing all irredundant versions of an ontology: one by means of calculating justifications combined with internalization, and one by direct identification of redundant axioms in an optimized hitting-set-tree algorithm that makes calls to an underlying description logic reasoner. Furthermore, we present empirical results for eliminating redundancy in existing ontologies based on an implementation of the above methods, where we report on both the efficiency of the algorithms and the effect of redundancy elimination applied to prevalent ontologies. By this, we contribute an effective and efficient means for consolidating ontologies on demand to any ontology engineering toolbox.

The paper is organized in sections. After recalling preliminaries and related work in Section 2, we introduce our notions around redundancy in ontologies and discuss typical cases of redundancy in Section 3. Then, we describe methods for an automated elimination of redundancy in Section 4 before reporting on experimental results in Section 5. We conclude with future work in Section 6.

2 Preliminaries

We introduce some basic notions around ontologies and review related work.

OWL and Description Logics. As a language for ontologies we consider the prominent Web Ontology Language (OWL) [11], which is based on the description logic (DL) formalism [2]. In description logics, an *ontology* $\mathcal{O}$ is a set of axioms that express either terminological (T-Box) or assertional (A-Box) knowledge. For details about types of axioms and the way complex concepts are constructed from individual, concept and role names, we refer to [2]. Our results are largely independent from the concrete DL used.

Inference with OWL ontologies builds on the notion of logical consequence, and we write $\mathcal{O} \models \alpha$ to mean that an ontology $\mathcal{O}$ logically *entails* an axiom α, and $\mathcal{O} \models \{\alpha_1, \ldots, \alpha_n\}$ to express entailment of several axioms.

The *signature* of an ontology $\mathcal{O}$, denoted by $\sigma(\mathcal{O})$, is the set of all individual, concept and role names occurring in $\mathcal{O}$, and thus, its vocabulary.

Moreover, by $\langle\mathcal{O}\rangle$ we denote the *deductive closure* of an ontology $\mathcal{O}$, i.e. the set $\{\alpha \mid \mathcal{O} \models \alpha\}$ of all DL axioms α that are logical consequences of $\mathcal{O}$.

Internalization. A technique called *internalization* can be used to express an ontology in form of a single concept inclusion axiom [2]. The internalization of an ontology $\mathcal{O}$ results in an axiom $\alpha_{\mathcal{O}}$ that contains all semantic information in $\mathcal{O}$, i.e. $\langle \mathcal{O} \rangle = \langle \{\alpha_{\mathcal{O}}\} \rangle$. However, OWL ontologies cannot be fully internalized in their complete expressivity. Certain role axioms cannot be internalized as they do not syntactically fit the form of the concept inclusion axiom $\alpha_{\mathcal{O}}$ (see e.g. [8]).

Justifications. For debugging ontologies, *justifications* are used to provide explanations for entailments. A justification is a minimal subset of an ontology that supports a given entailment, captured by the following definition from [9].

Definition 1 (justification). *For an ontology $\mathcal{O}$ and an axiom α with $\mathcal{O} \models \alpha$, a set $\mathcal{J}$ of axioms is a* justification for α in $\mathcal{O}$ *if $\mathcal{J} \subseteq \mathcal{O}$, $\mathcal{J} \models \alpha$ and there is no set $\mathcal{J}'$ such that $\mathcal{J}' \subset \mathcal{J}$ and $\mathcal{J}' \models \alpha$.*

Module Extraction. Techniques of module extraction are used to obtain a fragment of an ontology that is semantically relevant for entailments over a given signature. Despite containing only a subset of the original ontology's axioms, a *module* preserves all entailments with regard to this signature. We adapt the definition of a module based on the notion of conservative extension from [4].

Definition 2 (module). *Let $\mathcal{O}$ and $\mathcal{O}_m$ be ontologies with $\mathcal{O}_m \subseteq \mathcal{O}$ and Σ be a signature. Then, $\mathcal{O}_m$ is a* module for Σ in $\mathcal{O}$ *if for every ontology $\mathcal{O}'$ with $\sigma(\mathcal{O}') \cap \sigma(\mathcal{O}) \subseteq \Sigma$, we have that $\mathcal{O}' \cup \mathcal{O} \models \alpha$ if and only if $\mathcal{O}' \cup \mathcal{O}_m \models \alpha$ for any axiom α with $\sigma(\alpha) \subseteq \Sigma$.*

Related Work. A different notion of redundancy used in the DL literature, e.g. [2], refers to concept names in ontologies that are entailed to be equivalent to others, and thus are redundant vocabulary. Our notion of redundancy in axioms has not been extensively investigated in the context of OWL ontologies or description logics, and we mainly build on ideas introduced in [10] about redundant clauses in propositional logic formulas. Moreover, our main method for eliminating redundancy is largely inspired by the work on the use of a hitting-set-tree algorithm [12] for finding all justifications in [9].

In various works on normal forms, DL knowledge bases are transformed into a more compact form, such as prime implicate normal form [3] or linkless concept descriptions [5]. In contrast to yielding irredundant ontologies, these works are targeted to pre-processing ontologies for more efficient reasoning.

In [6], reductions of RDFS ontolgies are investigated in terms of RDF graphs that do not contain entailed triples explicitly, and their uniqueness is related to subsumption acyclicity of the original RDF graphs. Due to the simpler semantics of RDFS, however, these results only cover explicit class subsumption.

In [1], the compactification of ontologies has more been studied more with an application in mind and less focused on a grounding in the formal semantics underlying OWL.

3 Redundancy in Ontologies

In many ontology evolution scenarios, redundancy is unconsciously introduced in the axioms of an ontology, which can make it hard to maintain. In the following, we introduce useful notions and discuss cases of redundancy in ontologies.

3.1 Notion of Redundancy

For an ontology it is often desirable to derive a minimal version that does not contain any redundancy in its axioms but has the same semantical "meaning". An ontology contains redundancy in terms of expressed axioms if any of the axioms it contains is entailed by other axioms contained. The removal of such a redundant axiom would preserve the deductive closure of the ontology due to this entailment. Accordingly, we define *redundancy* in ontologies as follows.

Definition 3 (redundancy). *An ontology $\mathcal{O}$ is* redundant *if it contains an axiom α such that $\mathcal{O} \setminus \{\alpha\} \models \alpha$.*

We also call an irredundant subset of an ontology $\mathcal{O}$ that preserves the deductive closure of $\mathcal{O}$ a *reduction* of $\mathcal{O}$, as defined next.

Definition 4 (reduction). *Let $\mathcal{O}$ and $\hat{\mathcal{O}}$ be ontologies such that $\hat{\mathcal{O}} \subseteq \mathcal{O}$. Then, $\hat{\mathcal{O}}$ is a* reduction of $\mathcal{O}$ *if $\hat{\mathcal{O}}$ is irredundant and $\langle \hat{\mathcal{O}} \rangle = \langle \mathcal{O} \rangle$.*

In case an ontology is already irredundant, it is its own reduction. Hence, every ontology always has a reduction, which, however, need not be unique. For a focused removal of redundancy from parts of an ontology only, any part can be replaced by any of its reductions without loosing entailments due to monotonicity of DLs. For studying cases of an ontology having several reductions, we classify its axioms according to their level of *dispensability*, similar as in [10].

Definition 5 (dispensability of axioms). *For $\mathcal{O}$ an ontology and α an axiom,*

- *α is* indispensable in $\mathcal{O}$ *if it is contained in all reductions of $\mathcal{O}$*
- *α is* unconditionally dispensable in $\mathcal{O}$ *if it is in no reduction of $\mathcal{O}$;*
- *α is* conditionally dispensable in $\mathcal{O}$ *if there are two different reductions $\hat{\mathcal{O}}_1, \hat{\mathcal{O}}_2$ of $\mathcal{O}$ such that $\alpha \in \hat{\mathcal{O}}_1$ and $\alpha \notin \hat{\mathcal{O}}_2$.*

Indispensable axioms are those required in an ontology, unconditionally dispensable axioms those that can safely be removed, and conditionally dispensable axioms those that are interchangeably replaceable. Clearly, the existence of several reductions for an ontology is connected to the presence of conditionally dispensable axioms, as expressed in a proposition adapted from results in [10].

Proposition 1. *An ontology $\mathcal{O}$ has a unique reduction if and only if the following interchangeable conditions hold:*

1. *$\mathcal{O}$ has no conditionally dispensable axioms;*
2. *$\langle \mathcal{O} \rangle = \langle \mathcal{O}_i \rangle$ for $\mathcal{O}_i$ the axioms indispensable in $\mathcal{O}$;*
3. *there are no distinct sets S_1, S_2 of axioms conditionally dispensable in $\mathcal{O}$ such that $\langle \mathcal{O} \rangle = \langle \mathcal{O} \setminus S_1 \rangle = \langle \mathcal{O} \setminus S_2 \rangle \neq \langle \mathcal{O} \setminus (S_1 \cup S_2) \rangle$.*

Unfortunately, there is no easy way for distinguishing the conditionally dispensable axioms from the unconditionally dispensable ones. However, we can at least easily identify those axioms as unconditionally dispensable that follow from the indispensable axioms, according to the following lemma.

Lemma 1. *For an ontology $\mathcal{O}$ with indispensable axioms $\mathcal{O}_i$, any axiom $\alpha \in \mathcal{O}$ with $\mathcal{O}_i \models \alpha$ is unconditionally dispensable in $\mathcal{O}$, and thus, an ontology $\hat{\mathcal{O}}$ is a reduction of $\mathcal{O}$ if and only if it is a reduction of $\mathcal{O}_i \cup \{\alpha \in \mathcal{O} \mid \mathcal{O}_i \not\models \alpha\}$.*

A special case of redundancy covered by Lemma 1 is the presence of tautologies. Obviously, any tautological axiom in an ontology is unconditionally dispensable, and thus, no reduction can contain a tautology.

For the computation of reductions, we will be interested in a gradual removal of dispensable axioms from an ontology one-by-one. Removing single dispensable axioms from an ontology results in a strict decrease of the set of dispensable axioms in the respective sub-ontologies, such that reductions do not cease to be reductions by such removal, as stated in the following lemma.

Lemma 2. *Let $\mathcal{O}$ be an ontology with dispensable axioms $\mathcal{O}_d$ and α be an axiom with $\alpha \in \mathcal{O}_d$. Then, $\mathcal{O}'_d \subset \mathcal{O}_d$ for $\mathcal{O}'_d$ the axioms dispensable in $\mathcal{O} \setminus \{\alpha\}$, and thus, any reduction of $\mathcal{O} \setminus \{\alpha\}$ is also a reduction of $\mathcal{O}$.*

3.2 Cases of Redundancy

Since we look at redundancy in ontologies primarily from an ontology engineering and evolution point of view, we are interested in cases of redundancy as they occur in an ontology, and in the way redundancy is introduced in such cases.

Ontologies typically evolve over time with their different parts being developed and maintained in different contexts and separately from each other, which is a potential source for redundancy if such parts are combined. Moreover, big ontology development projects involve multiple ontology engineers who might introduce redundancy in the course of concurrent modeling activities.

Table 1 shows an example of a redundant ontology $\mathcal{O}_{hum}$ about human relationships that illustrates the notions introduced above. The axioms in Table 1 are listed in the order they were introduced, starting with basic notions about humans being either male or female, having children and being mother or father. These are followed by the notion of parent and more precise definitions of what it means to be father or mother, reflecting the successive expansion of an ontology in its evolution process. Overall, $\mathcal{O}_{hum}$ has two reductions and occurrences of both dispensable and indispensable axioms.

We identify cases in which redundancy is introduced in an unanticipated way.

Definition Over Subsumption. Definitions of concept names bear the potential to overwrite previously introduced subsumption axioms that involve these names, as they often succeed the introduction of new concepts by simple subsumption axioms. Axioms like $A \sqsubseteq B_1, A \sqsubseteq B_2$ become dispensable when adding a definition $A \equiv B_1 \sqcap B_2$ for A due to $\{A \equiv B_1 \sqcap B_2\} \models \{A \sqsubseteq B_1, A \sqsubseteq B_2\}$. In our example from Table 1, this case applies to the entailment $\{(10)\} \models (8)$.

Table 1. An example ontology about human relationships

$\mathcal{O}_{hum}$		
(1)	$Human \sqsubseteq \exists\, hasChild^{-}\, .Human$	humans are children of humans
(2)	$Human \equiv Male \sqcup Female$	humans are defined as either male or female
(3)	$Male \sqcap Female \sqsubseteq \bot$	males and females are distinct
(4)	$Father \sqsubseteq Human$	fathers are human
(5)	$Mother \sqsubseteq Human$	mothers are human
(6)	$Father \sqcap Mother \sqsubseteq \bot$	fathers and mothers are distinct
(7)	$Parent \equiv Human \sqcap \exists\, hasChild\, .Human$	parents are just humans with human children
(8)	$Father \sqsubseteq Parent$	fathers are parents
(9)	$Human \sqsubseteq= 2\, hasChild^{-}\, .Human$	humans have exactly two human parents
(10)	$Father \equiv Male \sqcap Parent$	fathers are male parents
(11)	$Mother \equiv Female \sqcap Parent$	mothers are female parents
(12)	$Parent(Peter)$	Peter is a parent
(13)	$Male(Peter)$	Peter is male
(14)	$Father(Peter)$	Peter is a father

indispensable:$\{2, 3, 7, 9, 10, 11\}$, conditionally disp.: $\{12, 13, 14\}$, uncond. disp.:$\{1, 4, 5, 6, 8\}$
reductions for $\mathcal{O}_{hum}$: $\hat{\mathcal{O}}_1 = \{2, 3, 7, 9, 10, 11, 14\}$, $\hat{\mathcal{O}}_2 = \{2, 3, 7, 9, 10, 11, 13, 12\}$

Subsumption Transitivity. Also simple subsumption axioms between concept names can introduce redundancy when explicating transitive parts of the subsumption hierarchy, e.g. by introducing additional intermediate subclasses. If $A \sqsubseteq C$ is stated about A first, and later on the intermediate subclass B is introduced by means of $A \sqsubseteq B$, $B \sqsubseteq C$ then $A \sqsubseteq C$ becomes dispensable. In our example from Table 1, this is the case for the entailment $\{(7), (8)\} \models (4)$.

Conjunctive Strengthening. Simple subsumption axioms between concept names bear the potential to be overwritten by more complex ones that strengthen the restrictions on the subsumed class by means of conjunctions. When placing new concept names in an ontology's subsumption hierarchy by means of axioms like $A \sqsubseteq B_1$, and then adding more restrictions by means of conjunction, such as $A \sqsubseteq B_1 \sqcap B_2$, at a later stage, the former axiom becomes dispensable. The entailment $\{(10)\} \models (8)$ with regard to Table 1 is also an example for this case.

Disjunctive Weakening. Disjunctions on the right-hand side of subsumption axioms bear the potential to become redundant when some of the disjunctive cases are excluded later on. An example are so called coverage axioms of the form $A \sqsubseteq B_1 \sqcup B_2$, which are a common design pattern and thus likely to be introduced early in the evolution of an ontology. If at a later stage more restrictive subsumptions about A are added without replacing the original coverage axiom, such as $A \sqsubseteq B_1$, then this one becomes dispensable due to $A \sqsubseteq B_1 \models A \sqsubseteq B_1 \sqcup B_2$.

Cardinality Inclusion. Also numbers in cardinality restrictions might introduce redundancy when applied to the same role but at different places or stages. The restriction $A \sqsubseteq\leq 3\ r$, for example, is overwritten by $A \sqsubseteq\leq 2\ r$, due to $A \sqsubseteq\leq 2\ r \models A \sqsubseteq\leq 3\ r$. For our example from Table 1, we get e.g. $\{(9)\} \models (1)$.

Inherited Disjointness. Subclasses of respective disjoint classes are again disjoint from each other. Hence, disjointness axioms introduced deep down in class

hierarchies become dispensable when the respective parent classes are stated to be disjoint. In our example from Table 1, we have that $\{(3)\} \models (6)$.

Assertional Redundancy. Also concept or role assertion axioms in ABoxes can be redundant in combination with TBox information. This might either happen in case a knowledge engineer describes a situation in an ABox very explicitly, not having the full information regarding all TBox axioms in mind that derive part of this situation, or in case a TBox is extended after situations have been described accurately in the ABox in a way that makes ABox statements obsolete. In our example from Table 1, axiom (14) is equivalent to $\{(12), (13)\}$.

For our experiments with OWL ontologies, we focus on scenarios in which ontologies are primarily hand-crafted and no techniques of automated knowledge acquisition are used. In such scenarios, we assume "typical" domain ontologies as used in the Semantic Web to contain rather little redundancy, since ontology engineers would most likely model their ontologies in a rather concise way, not repeating statements over and over in different ways. Therefore, we work with the following hypothesis for the subsequent computation of reductions.[1]

Hypothesis 1 (low redundancy). *An ontology is expected to contain significantly less dispensable axioms $\mathcal{O}_d$ than indispensable axioms $\mathcal{O}_i$: $\#\mathcal{O}_d \ll \#\mathcal{O}_i$.*

4 Computing Reductions

Starting from the notion of redundancy introduced above, we investigate how irredundant ontologies can be produced from redundant ones in an automated way. We propose two methods for computing reductions, one that builds on both internalization and the computation of justifications, and one that computes reductions directly based on the hitting set algorithm for diagnosis problems.

4.1 Finding Reductions by Computing Justifications

Recall the notion of a justification, which is a minimal subset of axioms of an ontology that support a particular entailment. As such, a justification has the property of not containing any redundancy at the level of expressed axioms, since excluding any of its axioms would give up the entailment. If we expand a single entailment to cover the whole ontology, we can extend this property from the restricted set of axioms in the justification to the ontology as a whole. We can do so by using the mechanism of internalization in order to encode all information of the original ontology in a single entailed axiom – giving up this entailment amounts to loosing information with regard to the original ontology. Hence, algorithms for computing justifications can be used to produce reductions.

Since there can be several justifications for an entailment, this approach results in several solutions for eliminating redundancy in an ontology. In fact, algorithms that compute all justifications produce all reductions of an ontology internalized by the respective entailed axiom, as stated next.

[1] In Section 5 we will support this hypothesis with some empirical evidence from experiments with existing ontologies.

Algorithm 1. computeRedJust $(\mathcal{O};\ R)$ – Compute reductions via justifications

Require: an ontology $\mathcal{O}$
Ensure: R is the set of all reductions of $\mathcal{O}$

```
O' := R := J := ∅
extractInternalisablePart(O; O')
internalize(O'; α_O')
computeJustifications(O, α_O'; J)
for all J_i ∈ J do
   R := R ∪ {J_i ∪ (O \ O')}
end for
```

Theorem 1. *Let $\mathcal{O}$ be an ontology and $\alpha_\mathcal{O}$ be the internalization of $\mathcal{O}$ such that $\langle\mathcal{O}\rangle = \langle\{\alpha_\mathcal{O}\}\rangle$. Any justification $\mathcal{J}_\mathcal{O}$ for $\alpha_\mathcal{O}$ in $\mathcal{O}$ is a reduction of $\mathcal{O}$ such that $\langle\mathcal{O}\rangle = \langle\mathcal{J}_\mathcal{O}\rangle$.*

Proof. Let $\mathcal{J}_\mathcal{O}$ be a justification for $\alpha_\mathcal{O}$ in $\mathcal{O}$. Then, the entailment $\mathcal{J}_\mathcal{O} \models \alpha_\mathcal{O}$ holds, and since $\mathcal{O}$ and $\alpha_\mathcal{O}$ are semantically equivalent, also $\mathcal{J}_\mathcal{O} \models \mathcal{O}$. Moreover, due to $\mathcal{J}_\mathcal{O} \subseteq \mathcal{O}$ we also have that $\mathcal{O} \models \mathcal{J}_\mathcal{O}$. This implies $\langle\mathcal{O}\rangle = \langle\mathcal{J}_\mathcal{O}\rangle$.

We have just seen that $\alpha_\mathcal{O} \in \langle\mathcal{J}_\mathcal{O}\rangle$ due to $\mathcal{J}_\mathcal{O} \models \alpha_\mathcal{O}$. Since $\mathcal{J}_\mathcal{O}$ is a minimal set of axioms that entail $\alpha_\mathcal{O}$, we get that $\mathcal{J}' \not\models \alpha_\mathcal{O}$ for any $\mathcal{J}' \subset \mathcal{J}_\mathcal{O}$. Hence, we get $\langle\mathcal{J}'\rangle \neq \langle\mathcal{J}_\mathcal{O}\rangle$ for any $\mathcal{J}' \subset \mathcal{J}_\mathcal{O}$, and by Definition 3, $\mathcal{J}_\mathcal{O}$ is irredundant. □

One drawback of this method is that in OWL not all axioms can in general be internalized, and axioms not covered, such as transitivity or other role properties, are not taken into account for the computation of reductions. However, for cases in which ontologies are expressed in a language fragment that contains such axioms, at least the internalizable part of an ontology can be freed of redundancy. Algorithm 1 provides a procedure for this that makes use of procedures for internalization and computation of justifications like the ones in [9].[2] By this, it provides a way to eliminate redundancy in ontologies by means of the readily available techniques for internalization and for computing justifications.

Another drawback of this method is its low efficiency. Internalization of the whole ontology in an axiom to be checked for entailment doubles the input for the underlying DL reasoner, which has a significant impact on the run time of the typically exponential time DL reasoning problem. Therefore, we investigate the direct computation of reductions, next.

4.2 Finding Reductions by Direct Diagnosis

A straightforward method for computing a single reduction of an ontology is to successively remove dispensable axioms from the ontology, which requires linearly many entailment checks for verifying dispensability of single axioms in the successive sub-ontologies. In case the ontology's dispensable axioms have been

[2] In our procedural notation, parameters before the ;-symbol are read-only and passed by value, while those after the ;-symbol are read/write and passed by reference.

Algorithm 2. computeSingleReduction $(\mathcal{O}, \mathcal{O}_d; \hat{\mathcal{O}})$ – Compute a single reduction for an ontology with pre-identified dispensable axioms

Require: an ontology $\mathcal{O}$, the set $\mathcal{O}_d$ of axioms dispensable in $\mathcal{O}$
Ensure: $\hat{\mathcal{O}}$ is a reduction of $\mathcal{O}$

```
Ô := O
for all α ∈ O_d do
  if Ô \ {α} ⊨ α then Ô := Ô \ {α}
end for
```

Algorithm 3. determineDispensableAxioms $(\mathcal{O}, \mathcal{O}^*; \mathcal{O}_d)$ – Find dispensable axioms

Require: an ontology $\mathcal{O}$, an ontology $\mathcal{O}^*$ with $\mathcal{O}^* \subseteq \mathcal{O}$ and $\mathcal{O}^*$ contains all axioms dispensable in $\mathcal{O}$
Ensure: $\mathcal{O}_d$ is the set of axioms that are dispensable in $\mathcal{O}$

```
O_d := ∅
for all α ∈ O* do
  if O \ {α} ⊨ α then O_d := O_d ∪ {α}
end for
```

pre-identified, this can be optimized verifying only dispensable axioms, since indispensable axioms do not need to be checked for removal, according to Definition 5. Algorithm 2 provides a procedure for computing a single reduction of an ontology with possibly pre-identified dispensable axioms in its second parameter; in case of not knowing the dispensable axioms in advance, this parameter is initialized with the whole ontology.

For the task of finding all (or more than one) reductions of an ontology, it is beneficial to spend the effort of pre-identifying dispensable axioms, since for all reductions to be computed the indispensable axioms can be neglected, and due to Hypotheses 1 their relative number is expected to be high. Algorithm 3 provides a procedure for this pre-identification, taking as its second parameter a subset of the original ontology for optimization in case some axioms can be excluded from being dispensable, which applies for repeated calls when only a subset of formerly dispensable axioms needs to be checked due to Lemma 2.

Based on pre-identified dispensable axioms, techniques for ontology module extraction can be applied to optimize the computation of all reductions of an ontology. Observe from Algorithm 2 that calls to a DL reasoner for entailment checking are restricted to axioms dispensable in the original ontology, which can expected to be few according to Hypothesis 1. Instead of checking entailment with respect to the full ontology, it is therefore sufficient to only take into account an ontology module that is computed by means of the signature of all dispensable axioms to preserve their (non-)entailment according to Definition 2. The following proposition provides the basis for this optimization.

Proposition 2. *Let $\mathcal{O}$ be an ontology with dispensable axioms $\mathcal{O}_d$ and $\mathcal{O}_m$ be a module for $\sigma(\mathcal{O}_d)$ in $\mathcal{O} \setminus \mathcal{O}_d$. Then, $\hat{\mathcal{O}}' \cup (\mathcal{O} \setminus (\mathcal{O}_m \cup \mathcal{O}_d)$ is a reduction of $\mathcal{O}$ for any reduction $\hat{\mathcal{O}}'$ of $\mathcal{O}_m \cup \mathcal{O}_d$.*

Proof. $\hat{\mathcal{O}}' \cup (\mathcal{O} \setminus (\mathcal{O}_m \cup \mathcal{O}_d)) \subseteq \mathcal{O}$ holds due to $\hat{\mathcal{O}}' \subseteq \mathcal{O}_m \cup \mathcal{O}_d \subseteq \mathcal{O}$.

Since $\hat{\mathcal{O}}'$ is a reduction of $\mathcal{O}_m \cup \mathcal{O}_d$, we have that $\langle \hat{\mathcal{O}}' \rangle = \langle \mathcal{O}_m \cup \mathcal{O}_d \rangle$, and thus, $\langle \hat{\mathcal{O}}' \cup (\mathcal{O} \setminus (\mathcal{O}_m \cup \mathcal{O}_d)) \rangle = \langle (\mathcal{O}_m \cup \mathcal{O}_d) \cup (\mathcal{O} \setminus (\mathcal{O}_m \cup \mathcal{O}_d)) \rangle = \langle \mathcal{O} \rangle$.

Finally, assume that $\hat{\mathcal{O}}' \cup (\mathcal{O} \setminus (\mathcal{O}_m \cup \mathcal{O}_d))$ is redundant. Then, there is an axiom $\alpha \in \hat{\mathcal{O}}' \cup (\mathcal{O} \setminus (\mathcal{O}_m \cup \mathcal{O}_d))$ with $\hat{\mathcal{O}}' \cup (\mathcal{O} \setminus (\mathcal{O}_m \cup \mathcal{O}_d)) \setminus \{\alpha\} \models \alpha$. As $\alpha \in \mathcal{O}_d$, and because $\mathcal{O} \setminus (\mathcal{O}_m \cup \mathcal{O}_d)$ contains only axioms indispensable in $\mathcal{O}$, we have that $\alpha \in \hat{\mathcal{O}}'$. Moreover, since $\alpha \in \mathcal{O}_d$, we get $\mathcal{O} \setminus \{\alpha\} = (\mathcal{O} \setminus \mathcal{O}_d) \cup (\mathcal{O}_d \setminus \{\alpha\}) \models \alpha$. Since $\mathcal{O}_m$ is a module for $\sigma(\mathcal{O}_d)$ in $\mathcal{O} \setminus \mathcal{O}_d$, the entailment $\mathcal{O}_m \cup (\mathcal{O}_d \setminus \{\alpha\}) \models \alpha$ follows from Definition 2, as $\sigma(\mathcal{O}_d \setminus \{\alpha\}) \cap \sigma(\mathcal{O} \setminus \mathcal{O}_d) \subseteq \sigma(\mathcal{O}_d)$. It implies that $\langle \mathcal{O}_m \cup (\mathcal{O}_d \setminus \{\alpha\}) \rangle = \langle \mathcal{O}_m \cup \mathcal{O}_d \rangle = \langle \hat{\mathcal{O}}' \rangle$, and thus, we get $\hat{\mathcal{O}}' \setminus \{\alpha\} \models \alpha$, since $\alpha \in \langle \mathcal{O}_m \cup (\mathcal{O}_d \setminus \alpha) \rangle$. This contradicts the assumption and $\hat{\mathcal{O}}' \cup (\mathcal{O} \setminus (\mathcal{O}_m \cup \mathcal{O}_d))$ is therefore irredundant. □

According to Proposition 2, computation of reductions of an ontology can be restricted to its dispensable axioms combined with a module in its indispensable axioms computed for the signature of the dispensable axioms. Due to Hypothesis 1 this computation of the module can be expected to potentially filter out large parts of an ontology irrelevant for elimination of redundancy in a pre-computation step when pre-identifying dispensable axioms.

To provide an effective and efficient method for computing all reductions of an ontology, we finally introduce the notion of a *reduction tree* based on principles of the hitting set tree algorithm presented in [12] and applied to computing justifications in [9].

Definition 6 (reduction tree). *For an ontology $\mathcal{O}$ with indispensable axioms $\mathcal{O}_i$, a* reduction tree *is a tree structure T with nodes labelled by sets of axioms and arcs labelled by axioms. Let $\mathcal{O}^* := \mathcal{O}_i \cup \{\alpha \in \mathcal{O} \mid \mathcal{O}_i \not\models \alpha\}$, $P(n)$ be the set of axioms along the path of arcs from the root node of T to node n, $\mathcal{O}_d(n)$ be the axioms dispensable in $\mathcal{O}^* \setminus P(n)$, $\mathcal{O}_d(n, \alpha)$ be the axioms of $\mathcal{O}_d(n)$ dispensable in $\mathcal{O}^* \setminus (P(n) \cup \{\alpha\})$, $\mathcal{O}_m(n)$ be a module for $\sigma(\mathcal{O}_d(n))$ in $\mathcal{O}^* \setminus P(n)$ and $\mathcal{O}_m(n, \alpha)$ be a module for $\sigma(\mathcal{O}_d(n, \alpha))$ in $\mathcal{O}_m(n)$. The tree T is defined recursively:*

- *T has at least one node, the root node n_0, which is labelled by $\hat{\mathcal{O}}' \cup (\mathcal{O}_i \setminus \mathcal{O}_m(n_0))$ for some reduction $\hat{\mathcal{O}}'$ of $\mathcal{O}_m(n_0) \cup \mathcal{O}_d(n_0)$;*
- *if n is a node of T with node label $\hat{\mathcal{O}}_n$ then n has a successor node n_α for each axiom α in $\hat{\mathcal{O}}_n \cap \mathcal{O}_d(n)$ with the following properties:*
  - *n_α is connected to n by an edge labelled by α;*
  - *the label of n_α is $\hat{\mathcal{O}}' \cup (\mathcal{O}_i \setminus \mathcal{O}_m(n, \alpha))$ for some reduction $\hat{\mathcal{O}}'$ of $\mathcal{O}_m(n, \alpha) \cup \mathcal{O}_d(n, \alpha)$.*

We show that a reduction tree can be used as a means to compute all reductions of an ontology in the following theorem.

Theorem 2. *The set of all node labels of a reduction tree for an ontology $\mathcal{O}$ is the set of all reductions of $\mathcal{O}$.*

Proof. We will prove the following two claims: a) the label of any node in T is a reduction of $\mathcal{O}$; b) for any reduction $\hat{\mathcal{O}}$ of $\mathcal{O}$ there is a node in T with label $\hat{\mathcal{O}}$.

a) The proof is by induction over the tree structure of T.

The label $\hat{\mathcal{O}}' \cup (\mathcal{O}_i \setminus \mathcal{O}_m(n_0))$ of n_0 is equivalent to $\hat{\mathcal{O}}' \cup (\mathcal{O}^* \setminus (\mathcal{O}_m(n_0) \cup \mathcal{O}_d(n_0)))$, since $\mathcal{O}^* \setminus \mathcal{O}_d = \mathcal{O}_i$. Due to Proposition 2 $\hat{\mathcal{O}}' \cup (\mathcal{O}^* \setminus (\mathcal{O}_m(n_0) \cup \mathcal{O}_d(n_0)))$ is a reduction of $\mathcal{O}^*$, and due to Lemma 1 it is also a reduction of the semantically equivalent $\mathcal{O}$.

Now, let n be any node in T with label $\hat{\mathcal{O}}_n$ a reduction of $\mathcal{O}$ and n_α be any successor node of n connected to n by an edge labelled α. Then, the node label $\hat{\mathcal{O}} := \hat{\mathcal{O}}' \cup (\mathcal{O}_i \setminus \mathcal{O}_m(n, \alpha))$ of n_α is equivalent to $\hat{\mathcal{O}}' \cup ((\mathcal{O}^* \setminus (P(n) \cup \{\alpha\})) \setminus (\mathcal{O}_m(n, \alpha) \cup \mathcal{O}_d(n, \alpha)))$, since $\mathcal{O}^* \setminus (P(n) \cup \{\alpha\} \cup \mathcal{O}_d(n, \alpha)) = \mathcal{O}_i$. Hence, $\hat{\mathcal{O}}$ is a reduction of $\mathcal{O}^* \setminus (P(n) \cup \{\alpha\})$ due to Proposition 2, since $\mathcal{O}_d(n, \alpha)$ are just the dispensable axioms in $\hat{\mathcal{O}}$. Due to Lemma 2 $\hat{\mathcal{O}}$ is also a reduction of $\mathcal{O}^* \setminus P(n)$, because of $\alpha \in \mathcal{O}_d(n)$. Since also $\hat{\mathcal{O}}_n$ is a reduction of $\mathcal{O}^* \setminus P(n)$ (due to Proposition 2), we have that $\langle \hat{\mathcal{O}}_n \rangle = \langle \hat{\mathcal{O}} \rangle$. Hence, the irredundant $\hat{\mathcal{O}}$ is also a reduction of $\mathcal{O}$, as is $\hat{\mathcal{O}}_n$.

b) Let $\hat{\mathcal{O}}$ be a reduction of $\mathcal{O}$. Again by induction, we show that $\hat{\mathcal{O}}$ is a node label along some branch in T.

For the root node n_0 we have that $\hat{\mathcal{O}} \subseteq \mathcal{O}^* = \mathcal{O}^* \setminus P(n_0)$. Now, let n be any node in T with $\hat{\mathcal{O}} \subseteq \mathcal{O}^* \setminus P(n)$. If the label of n is not $\hat{\mathcal{O}}$ then it contains some axiom α with $\alpha \notin \hat{\mathcal{O}}$, since different reductions of $\mathcal{O}$ deviate by conditionally dispensable axioms due to Proposition 1. In this case, there is a successor node n_α, connected to n by an edge labelled α, such that $\hat{\mathcal{O}} \subseteq \mathcal{O}^* \setminus (P(n_\alpha))$.

Due to the strict decrease of the set $\mathcal{O}^* \setminus P(n)$ down the paths the tree is finite and at a certain point $\hat{\mathcal{O}}$ must be the label of some node. □

With Algorithm 5 and Algorithm 4 we present procedures to compute all reductions of an ontology in an optimized way. A call to the procedure computeReductions initiates the computation with a call to the recursive procedure computeAllReductions, which traverses the reduction tree and makes use of Lemma 2 when passing subsets of dispensable axioms down the tree structure. The procedure computeModule computes a module of an ontology based on a signature with the properties according to Definition 2 using techniques from [4], while the procedure determineUncondDispAxioms in Algorithm 6 identifies part of the unconditionally dispensable axioms according to Lemma 1. In addition to pre-identification of dispensable axioms and module extraction, we can also make use of optimizations for node label reuse and tree pruning that have been devised for hitting set tree algorithms. They are described in [12] and are implemented in the conditions of the if-statements in Algorithm 4, similar to [9].

5 Oberservations on Reduction Computation

In order to evaluate the practicability of removing redundancy from ontologies, the above algorithms were implemented using the latest version of the OWL API[3]. We used the Pellet reasoner [14] (v2.1.1) to check entailments, compute regular justifications and to extract modules. The tests have been performed

[3] `http://owlapi.sourceforge.net`

Algorithm 4. computeAllReductions $(\mathcal{O},\ \mathcal{O}_d,\ \mathcal{O}_m,\ P;\ R,\ P_c,\ P_o)$ – Collect all reductions of an ontology recursively

Require: an ontology $\mathcal{O}$, the set $\mathcal{O}_d$ of axioms dispensable in $\mathcal{O}$, a module $\mathcal{O}_m$ for $\sigma(\mathcal{O}_d)$ in $\mathcal{O} \setminus \mathcal{O}_d$, a set P of axioms with $P \cap \mathcal{O} = \emptyset$
Ensure: R contains all reductions of $\mathcal{O}$, $P_c \ldots$, $P_o \ldots$

```
if R ≠ ∅ and P ∩ O* = ∅ for any O* ∈ R then
  Ô := O*
else
  Ô := ∅
  computeSingleReduction (O_m ∪ O_d, O_d; Ô)
  Ô := Ô ∪ (O \ O_m)
  R := R ∪ {Ô}
endif
O'_m := O'_d := ∅
for all α ∈ Ô ∩ O_d do
  if not (P ∪ {α} ⊇ P* for any P* ∈ P_c or P ∪ {α} = P* for any P* ∈ P_o) then
    determineDispensableAxioms (O \ {α}, O_d \ {α}; O'_d)
    computeModule (O_m, σ(O'_d); O'_m)
    computeAllReductions (O \ {α}, O'_d,O'_m, P ∪ {α};R, P_c, P_o)
  endif
  P_o := P_o ∪ {P ∪ {α}}
end for
P_c := P_c ∪ {P}
```

Algorithm 5. computeReductions $(\mathcal{O}; R)$ – Calculate all reductions of an ontology

Require: an ontology $\mathcal{O}$
Ensure: R contains all reductions of $\mathcal{O}$

```
O_d := ∅
determineDispensableAxioms (O, O; O_d)
O_m := ∅
computeModule (O \ O_d, σ(O_d); O_m)
O_u := ∅
determineUncondDispAxioms (O_m ∪ O_d, O_d; O_u)
R := P_c := P_o := ∅
computeAllReductions (O \ O_u, O_d \ O_u, O_m, ∅; R, P_c, P_o)
```

Algorithm 6. determineUncondDispAxioms $(\mathcal{O}, \mathcal{O}_d; \mathcal{O}_u)$ – Determine unconditionally dispensable axioms

Require: an ontology $\mathcal{O}$, the set $\mathcal{O}_d$ of axioms dispensable in $\mathcal{O}$
Ensure: $\mathcal{O}_u$ contains only axioms that are unconditionally dispensable in $\mathcal{O}$

```
O_u := ∅
for all α ∈ O_d do
  if O \ O_d ⊨ α then O_u := O_u ∪ {α}
end for
```

Table 2. Results of redundancy elimination with all optimizations enabled.

				reduced to		dispensability				
Ontology	DL	$\#\mathcal{O}$	$\#\{\hat{\mathcal{O}}_i\}$	max.	min.	$\#\mathcal{O}_i$	$\#\mathcal{O}_c$	$\#\mathcal{O}_u$	t_{first}	t_{mean}
1 HumanRel	$\mathcal{ALCIQ}$	14	2	50%	57%	6	3	5	0.19	0.10
2 Generations	$\mathcal{ALCOIF}$	38	5	87%	87%	25	2	11	0.10	0.04
3 Nautilus	$\mathcal{ALCHF}(\mathcal{D})$	38	1	84%	84%	32	0	6	0.06	0.06
4 PeriodicTable	$\mathcal{ALU}$	58	1	97%	97%	56	0	2	0.29	0.29
5 People	$\mathcal{ALCHOIN}$	108	1	93%	93%	88	0	20	0.52	0.52
6 DOLCE Lite	$\mathcal{SHIF}$	351	58	54%	56%	134	157	60	7.73	4.61
7 Pizza	$\mathcal{SHOIN}$	712	12	58%	59%	404	26	282	20.38	2.21
8 Transportation	$\mathcal{ALCH}(\mathcal{D})$	1157	3	90%	90%	1011	6	140	32.10	10.87
9 Economy	$\mathcal{ALCH}(\mathcal{D})$	1625	3	43%	44%	705	4	916	43.46	21.74
10 Process	$\mathcal{ALCHOF}(\mathcal{D})$	2578	15	94%	94%	2210	29	339	172.66	14.56
11 Wine	$\mathcal{SHOIN}(\mathcal{D})$	657	3	40%	40%	262	5	390	338.99	57.30
12 FlyAnatomy	$\mathcal{EL}^{++}$	10471	5	98%	98%	10289	14	168	2845.76	576.78

on a laptop with 2.4 GHz dual core processor, with Java 1.6, assigning 1 GB memory to Java. A selection of publicly available ontologies (as shown in Table 2) varying in size and expressivity have been used in the experiments.[4]

Before we conducted our main experiments with the optimized hitting set tree approach, we tested the elimination of redundancy via justifications as described in Algorithm 1 on the same set of ontologies. Small and inexpressive ontologies could be reduced properly in acceptable time. Especially, reductions for the human relationship example (1) could be computed in 2,24s, for the Nautilus ontology (3) in 4,98s and for the PeriodicTable (4) in 9,7s. No redundancy was found in the Generations ontology (2) and the People ontology (5) as their redundancies depend on uninternalized RBox axioms. Unfortunatly, computations for all other ontologies in our test set either terminated with heap space exceptions or did not terminate at all within a timeframe of several hours. These results lead us to the belief that a more scalable method both in terms of performance and with no restrictions regarding axiom types is preferable.

As an alternative, we evaluated redundancy elimination with the optimized hitting set tree approach as described in Algorithm 4. As Table 2 shows, the achieved reduction rates ranged from 40% to 98%. In most cases the number of dispensable axioms is considerably smaller than the number of indispensable ones, which empirically supports Hypothesis 1. Exceptions are ontologies designed as example showcases, such as the human relationship or pizza ontology. Further note that Hypothesis 1 was formulated with focus on reducing TBoxes. If we, for example, consider ABoxes with large amounts of materialized knowledge, such as transitive role assertions, the situation might be different and the optimizations would need to be configured accordingly.

The ontologies in Table 2 are sorted according to the time it took for the first reduction to be computed. The quickest work in real-time (1–5), some take up

[4] The wine ontology may be retrieved from `http://www.w3.org/TR/2003/PR-owl-guide-20031209/wine`. All other ontologies used may be found in the TONES ontology repository at `http://owl.cs.manchester.ac.uk/repository`.

to seconds, others take up to minutes, and the last one reveals the computational limits of the approach. The combination of both size and expressivity determine the efficiency of redundancy elimination although the results indicate that the impact of reasoning complexity is considerably higher. Notably, the wine ontology (11) needed longer time to process than ontologies (8–10), which have more axioms but are less expressive; e.g. it took more than seven times longer to compute the first wine (11) reduction than the first economy (9) reduction despite the significantly smaller size. Interestingly the Pizza ontology does not need as long although it is also expressed in $\mathcal{SHOIN}$. It is however known that certain configurations of concept descriptions are especially hard for automated reasoning as it is the case in the wine ontology as discussed by [13].

We investigated the effect of different configurations of optimizations. As expected, the pre-identification of dispensable axioms brings a large performance benefit. The computation time for the first reduction is in general significantly larger than the computation times for successive reductions.

In our current experiments we considered two configurations for module extraction: firstly only an initial module extraction with no further extraction during the hitting set exploration, and secondly the repeated extraction of module throughout the algorithm, and compared these with the computation times when just using hitting-set optimizations. We observed that for small ontologies (1–3) the cost of modularization was higher than the gain. Here the modularization slowered the computation up to factor of two to three. At ontologies (4–9) a break-even seems to be reached for initial modularization while inner modularization is still costly. For the larger and more expressive ontologies we find a gain in modularization, though no clear gain of repeated modularization is visible in the current setting. An introspection of the results showed however that the size of the extracted modules decreased within the first two or three computations and then remained the same or just change very little. For example, in ontology 12 the first three modules have the size 10289, 10133, 9931. As the following extraction steps return modules of constant size 9931 the computation is counterproductive. However, this also indicates a positive effect of repeated modularization for large ontologies but also that a more fine-tuned approach to when to start or stop modularization is desirable.

6 Conclusion

We have introduced notions around redundancy in OWL ontologies and have identified typical cases where redundancy is introduced in the course of ontology evolution in an unanticipated way. We have provided two methods for eliminating redundancy: one utilising readily available techniques of computing justifications and internalization, and another one based on a hitting-set-tree algorithm further optimized by module extraction. We have shown our optimized methods to be effective and efficient on typical ontologies used in the Semantic Web context, based on a prototypical implementation.

For future work, we plan to investigate the elimination of redundancy in parts of axioms to yield a more fine-grained notion of redundancy, similar to work

on laconic justifications in [7]. Moreover, we want to target the comparison and evaluation of different reductions and to provide measures that help a knowledge engineer to decide for one.

References

1. Alani, H., Harris, S., O'Neil, B.: Winnowing Ontologies Based on Application Use. In: Sure, Y., Domingue, J. (eds.) ESWC 2006. LNCS, vol. 4011, pp. 185–199. Springer, Heidelberg (2006)
2. Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Patel-Schneider, P. (eds.): The Description Logic Handbook: Theory, Implementation and Applications. Cambridge University Press, Cambridge (2007)
3. Bienvenu, M.: Prime Implicate Normal Form for $\mathcal{ALC}$ Concepts. In: AAAI 2008: Proc. of the 23rd Conference on Artif. Intelligence, pp. 412–417. AAAI Press, Menlo Park (2008)
4. Cuenca Grau, B., Horrocks, I., Kazakov, Y., Sattler, U.: Extracting Modules from Ontologies: A Logic-based Approach. In: Stuckenschmidt, H., Parent, C., Spaccapietra, S. (eds.) Modular Ontologies. LNCS, vol. 5445, pp. 159–186. Springer, Heidelberg (2009)
5. Furbach, U., Obermaier, C.: Knowledge Compilation for Description Logics. In: Dershowitz, N., Voronkov, A. (eds.) LPAR 2007. LNCS (LNAI), vol. 4790. Springer, Heidelberg (2007)
6. Gutierrez, C., Hurtado, C., Mendelzon, A.O.: Foundations of Semantic Web Databases. In: PODS 2004: Proceedings of the Twenty-third ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, pp. 95–106. ACM, New York (2004)
7. Horridge, M., Parsia, B., Sattler, U.: Laconic and Precise Justifications in OWL. In: Sheth, A.P., Staab, S., Dean, M., Paolucci, M., Maynard, D., Finin, T., Thirunarayan, K. (eds.) ISWC 2008. LNCS, vol. 5318, pp. 323–338. Springer, Heidelberg (2008)
8. Horrocks, I., Kutz, O., Sattler, U.: The Even More Irresistible $\mathcal{SROIQ}$. In: Proc. of the 10th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR 2006), pp. 57–67. AAAI Press, Menlo Park (2006)
9. Kalyanpur, A., Parsia, B., Horridge, M., Sirin, E.: Finding All Justifications of OWL DL Entailments. In: Aberer, K., Choi, K.-S., Noy, N., Allemang, D., Lee, K.-I., Nixon, L.J.B., Golbeck, J., Mika, P., Maynard, D., Mizoguchi, R., Schreiber, G., Cudré-Mauroux, P. (eds.) ASWC 2007 and ISWC 2007. LNCS, vol. 4825, pp. 267–280. Springer, Heidelberg (2007)
10. Liberatore, P.: Redundancy in Logic I: CNF Propositional Formulae. Artif. Intell. 163(2), 203–232 (2005)
11. W3C, O.W.L.: Working Group. OWL 2 Web Ontology Language: Document Overview. W3C Recommendation, October 27 (2009), `http://www.w3.org/TR/owl2-overview/`
12. Reiter, R.: A Theory of Diagnosis from first Principles. Artif. Intell. 32(1), 57–95 (1987)
13. Sirin, E., Cuenca Grau, B., Parsia, B.: From Wine to Water: Optimizing Description Logic Reasoning for Nominals. In: KR, pp. 90–99. AAAI Press, Menlo Park (2006)
14. Sirin, E., Parsia, B., Cuenca Grau, B., Kalyanpur, A., Katz, Y.: Pellet: A Practical OWL-DL Reasoner. Journal of Web Semantics 5(2), 51–53 (2007)