

Computing OWL Ontology Decompositions Using Resolution

Robert Schiaffino³, Achille Fokoue¹, Aditya Kalyanpur¹, Aaron Kershenbaum¹,
Li Ma², Chintan Patel⁴, Edith Schonberg¹, and Kavitha Srinivas¹

¹ IBM Watson Research Center, P.O. Box 704, Yorktown Heights, NY 10598, USA
achille, aaronk, ediths, ksrinivs@us.ibm.com

² IBM China Research Lab, Beijing 100094, China
malli@cn.ibm.com

³ Iona College, New Rochelle, NY 10000
rschiaffino@iona.edu

⁴ Columbia University Medical Center
chintan.patel@dbmi.columbia.edu

Abstract. Reasoning over large ontologies can be done more effectively if they can be decomposed into smaller parts which can be reasoned on independently. This requires identifying parts of the ontology relevant to the problem at hand. We present a novel algorithm, based on resolution calculus, for decomposing an OWL ontology into smaller, more manageable components, such that the union of reasoning over each of these components separately is the same as reasoning over the original ontology. We describe our computational experience using the algorithm, and demonstrate that it is indeed possible to efficiently solve the standard concept subsumption reasoning problem in four large real-world OWL ontologies: SNOMED-CT, NCI, SWEET-JPL and GALEN. We chose SNOMED-CT and NCI because of their size; Galen because it is highly interconnected; SWEET-JPL because it is expressive (containing negation, both existential and universal quantifiers and both intersections and unions).

1 Introduction

Ontologies that model domains in the real world are often very large, pushing the limits of existing reasoners. For example, SNOMED CT⁵, has over 300,000 concepts and approximately the same number of axioms. To reduce the cost of reasoning over large ontologies, we propose a sound and complete technique which allows us to work only with axioms and concepts relevant to the current task. In some cases, this may also involve merging results obtained from parts of a decomposed ontology.

In this paper, we focus on OWL ontologies, specifically the TBox which contains terminological assertions about concepts and the RBox which contains assertions about roles and role hierarchies. Furthermore, we focus on the standard

⁵ <http://www.snomed.org/snomedct/index.html>

reasoning task of computing the concept subsumption hierarchy. Thus, our goal is to decompose the original OWL ontology into smaller components (that are not necessarily disjoint), such that performing subsumption reasoning on each of the components separately and aggregating the results produces the same subsumption hierarchy as reasoning over the entire ontology.

The approach we take is based on *resolution calculus*. Resolution is a standard, sound and complete⁶ inference procedure for Propositional and First Order Logic. This allows our decomposition strategy to be applicable to any fragment of FOL, such as *SHOIN*(\mathcal{D}), which is the description logic that OWL-DL corresponds to. However, in this paper, we restrict our attention to *SHIN*

Before performing reasoning, we use resolution to detect which axioms in the KB are relevant to infer a particular subsumption entailment, and form components based on these axioms. The approach is conservative in that it may produce *non-minimal* components, i.e., the components may include additional axioms that are not relevant to the subsumption entailment under consideration. However, in our experiments, we have observed that we still obtain substantial gains in runtime and memory by doing decomposition. Next we perform the actual reasoning task on each component, using Pellet [1], a sound and complete DL-optimized reasoner.

To summarize, the contributions of this paper are:

1. We provide a theoretical framework for ontology decomposition by defining the notion of an inconsistency-preserving and subclass-preserving sub-KB, i.e., fragments of a KB that preserve key logical entailments.
2. We present a novel algorithm based on logical resolution to compute such fragments for OWL ontologies.
3. We demonstrate the practical significance of this algorithm for decomposing OWL TBoxes and RBoxes for the purpose of subsumption reasoning on four large real-world OWL ontologies, SNOMED-CT, NCI, SWEET-JPL and GALEN, and show that reasoning time and memory are significantly reduced.

2 Approach

We first focus on the general problem that forms the basis of our ontology decomposition technique: given a DL knowledge base which may be unsatisfiable, find a fragment of this KB that contains sufficient axioms to prove its inconsistency. The motivation here is that KB consistency checking is a key entailment problem in OWL-DL since all standard reasoning tasks can be reduced to it. Thus, the goal of decomposing a KB when checking a specific entailment (such as concept subsumption) is to produce relatively smaller fragments of the KB that are sufficient to prove the entailment. We first apply our approach to ontologies containing only Horn clauses and then extend it to non-Horn clauses.

⁶ Resolution is complete only w.r.t. derivation of the empty clause

2.1 Extracting an Inconsistency-Preserving sub-KB

Definition 1. Given a knowledge base \mathcal{K} , an inconsistency-preserving sub-KB \mathcal{K}' is a subset of \mathcal{K} that satisfies the following property $\mathcal{K} \models \perp$ iff $\mathcal{K}' \models \perp$.

Note that the above definition does not make any guarantees about the size or the *minimality* of \mathcal{K}' . However, in practice, we would like our algorithm to produce a \mathcal{K}' that is relatively small compared to the original KB.

Our approach is based on FOL resolution. Since *SHIN*, is a subset of FOL, it is possible to rewrite a *SHIN* KB, say \mathcal{K}_1 , as a set of FOL clauses \mathcal{K}_2 . If \mathcal{K}_1 is unsatisfiable, then \mathcal{K}_2 is obviously unsatisfiable as well, and in particular, an FOL-based resolution procedure when applied to \mathcal{K}_2 must produce the empty clause. Thus, a precise solution to determining an inconsistency-preserving sub-KB is to apply FOL resolution to \mathcal{K}_2 and determine which clauses in \mathcal{K}_2 play a role in generating the empty clause.

However, our approach does not require that we actually do the complete resolution. Instead, we compute a conservative approximation, guaranteeing that we will never fail to recognize when the empty clause can be derived. We may, however, sometimes conclude that it can be derived when in fact it cannot. In practice, this turns out to be a good tradeoff as our approach is much faster than doing a full resolution, and in our experiments, we have seen that the fragments we extract are still small enough to allow us to obtain significant reductions in reasoning time. Having obtained a decomposition using this conservative approximation, we then perform sound and complete DL reasoning.

Our conservative approach is based on *abstracting* the FOL KB into Propositional Logic (PL), i.e., we get rid of all the variables and constants in the clauses, and treat each of the predicates as propositions instead. Below is an example of how a DL axiom is converted into an equivalent FOL clause and then abstracted into PL:

$$(\mathbf{DL}) C \sqsubseteq \forall R.D \rightarrow (\mathbf{FOL}) \forall x, y, \neg C(x) \vee \neg R(x, y) \vee D(y) \rightarrow (\mathbf{PL}) \neg C \vee \neg R \vee D$$

Next, we apply *unit-resolution* to the resulting approximate Horn propositional KB. Unit resolution is a sound and complete resolution procedure for Horn-clause KBs, where each resolution step must include at least one unit clause, i.e., a clause containing a single literal. If applying unit-resolution over the PL KB derives an empty clause, we record the complete set of responsible clauses, and obtain the corresponding DL axioms that they were abstracted from.

However, there is an important caveat here. Since we are conservative in our approximation of the KB, it is possible that unit resolution produces an empty clause when their corresponding DL axioms are not inconsistent. In such a case, the output of our algorithm is not an inconsistency-preserving sub-KB. In order for our solution to be complete, we need to modify unit resolution to find all possible derivations of the empty clause. In practice, the performance of the algorithm is not affected much by the modification, since typically there are a small number of derivations of the empty clause, and unit resolution has poly-

nomial complexity and is fast in practice. An outline of our *Approx-Resolution* algorithm is given in Table 1.

<p>Algorithm: Approx-Resolution Input: <i>Horn – SHLN</i> KB \mathcal{K}_1 Output: <i>Horn – SHLN</i> KB \mathcal{K}_2</p>
<ol style="list-style-type: none"> 1. Convert \mathcal{K}_1 into an equivalent (unsatisfiability preserving) set of FOL clauses \mathcal{K}_1^{FOL} as shown in Table 2 2. Abstract the FOL KB \mathcal{K}_1^{FOL} into a propositional logic KB \mathcal{K}_1^{PL} by removing all variables/constants from the clauses in \mathcal{K}_1^{FOL} Let $\mathcal{K}_1^{PL-Horn}$ be corresponding pure Horn KB formed. 3. (This step is used only in the extension to non-Horn clauses as described in Section 2.4) Replace each non-Horn clause in \mathcal{K}_1^{PL} of the form $(\neg P_1 \vee \dots \vee \neg P_n) \vee (Q_1 \vee \dots \vee Q_m)$, with ‘m’ new clauses of the form $(\neg P_1 \vee \dots \vee \neg P_n) \vee Q_i$ ($1 \leq i \leq m$). 4. Perform unit resolution on $\mathcal{K}_1^{PL-Horn}$ until <i>no more empty clauses</i> can be derived set $\mathcal{K}_2 \leftarrow \emptyset$ for each empty clause derived, let $(\mathcal{K}_1^{PL-Horn})_{\perp} \subseteq \mathcal{K}_1^{PL-Horn}$ denote clauses responsible for deriving \perp for each clause $\alpha \in (\mathcal{K}_1^{PL-Horn})_{\perp}$, $\beta \leftarrow$ DL axiom in \mathcal{K}_1 that α was abstracted from $\mathcal{K}_2 \leftarrow \mathcal{K}_2 \cup \beta$ return \mathcal{K}_2

Table 1. Approx-Resolution Algorithm

$C \sqsubseteq D$	$\forall x, \neg C(x) \vee D(x)$ (Note: $C \equiv D$ is converted into 2 subclass axioms)
$R \sqsubseteq S$	$\forall x, \forall y, \neg R(x, y) \vee S(x, y)$ (Note: $R \equiv S$ is converted into 2 subproperty axioms)
$\exists R.C$	$\forall x, R(x, f(x)) \wedge C(f(x))$ (where $f(x)$ is a skolem fn.)
$\forall R.C$	$\forall x, \forall y : R(x, y) \rightarrow C(y)$
$\geq n.R$	$\forall x, R(x, f_1(x)) \wedge R(x, f_2(x)) \wedge \dots \wedge R(x, f_n(x)) \wedge \bigwedge_{i < j} f_i(x) \neq f_j(x)$ (where $f_1..f_n$ are skolem fns.)
$\leq n.R$	$\forall x, \forall y_1, \dots, y_{n+1}, R(x, y_1) \wedge R(x, y_2) \wedge \dots \wedge R(x, y_{n+1}) \rightarrow \bigcup_{i < j} y_i = y_j$

Table 2. Converting DL axioms/concepts to FOL clauses. Note that C,D are complex concepts

The following theorem shows that this algorithm produces a valid inconsistency-preserving sub-KB.

Theorem 1. *Let \mathcal{K}_1 be a Horn – SHLN KB and $\mathcal{K}_2 = \text{Approx-Resolution}(\mathcal{K}_1)$. Then, \mathcal{K}_2 is a inconsistency-preserving sub-KB of \mathcal{K}_1 , i.e., $\mathcal{K}_2 \subseteq \mathcal{K}_1$ and $\mathcal{K}_1 \models \perp$ iff $\mathcal{K}_2 \models \perp$.*

Proof. The fact that \mathcal{K}_2 is a subset of \mathcal{K}_1 is obvious given that \mathcal{K}_2 is populated in step 3 of the algorithm, wherein it only includes original axioms (β) in \mathcal{K}_1 from which propositional KB clauses have been abstracted. It also follows from the monotonicity of the logic, that if $\mathcal{K}_2 \subseteq \mathcal{K}_1$, then $\mathcal{K}_2 \models \perp \Rightarrow \mathcal{K}_1 \models \perp$. Hence, we only need to prove the reverse direction, i.e., if $\mathcal{K}_1 \models \perp$, then $\mathcal{K}_2 \models \perp$.

We prove this by contradiction. Suppose $\mathcal{K}_1 \models \perp$ and $\mathcal{K}_2 \not\models \perp$.

Let \mathcal{K}_1^{FOL} be the \mathcal{K}_1 equivalent FOL-clause KB. We know that \mathcal{K}_1^{FOL} is a pure Horn KB. Let $\mathcal{K}_1^{PL-Horn}$ be the propositional Horn KB, abstracted from \mathcal{K}_1^{FOL} (by removing all variables/constants in the clauses).

We know the following:

(a) Since $\mathcal{K}_1 \models \perp$, a sound and complete FOL resolution procedure on \mathcal{K}_1^{FOL} must yield the empty clause. Let $(\triangleright^\perp)_{FOL}$ denote the derivation that leads to the empty clause.

(b) If two FOL-clauses resolve, the corresponding propositional-clauses, which ignore variables or constants in the predicates, must resolve. This is because FOL resolution requires unification of variables in each resolution step, and the weaker Propositional-logic resolution does not.

(c) However, the converse of (b) is not true, i.e., it is possible that two FOL clauses do not resolve because of the inability to unify variables, but resolve in the propositional case.

From (a) and (b), it follows that a sound and complete resolution procedure on $\mathcal{K}_1^{PL-Horn}$ must yield the empty clause. Let $(\triangleright^\perp)_{PL}$ be the derivation under propositional resolution corresponding to the derivation $(\triangleright^\perp)_{FOL}$ under FOL resolution, i.e., a derivation that uses the same set of clauses and resolves on the same propositions in PL as predicates in FOL. We denote this correspondence as $(\triangleright^\perp)_{PL} \cong (\triangleright^\perp)_{FOL}$. Let \mathcal{K}'_2 be the set of DL axioms responsible for the derivation $(\triangleright^\perp)_{PL}$. Then, it follows that $\mathcal{K}'_2 \models \perp$.

Now, because of (c), it is possible that unit resolution on $\mathcal{K}_1^{PL-Horn}$ produces an empty clause under derivation $\triangleright \not\cong (\triangleright^\perp)_{FOL}$. The algorithm, however, uses a modified version of unit resolution that finds all possible derivations, say $\triangleright_1, \triangleright_2 \dots \triangleright_n$, and returns the sum union \mathcal{K}_2 of DL axioms used in each of the derivations.

Hence, there must exist at least one $\triangleright_i (1 \leq i \leq n)$ s.t. $\triangleright_i \cong (\triangleright^\perp)_{FOL}$. This implies that $\mathcal{K}'_2 \subseteq \mathcal{K}_2$. Since $\mathcal{K}'_2 \models \perp$, by monotonicity, $\mathcal{K}_2 \models \perp$. This contradicts our assumption.

2.2 Using Approx-Resolution in Ontology Decomposition

We describe how the *Approx-Resolution* algorithm is used to compute ontology decompositions for the purpose of subsumption reasoning. Again, our goal is to decompose the original ontology into a set of smaller components, perform subsumption reasoning independently over each of the components, and merge the results at the end to obtain the complete subsumption hierarchy.

Definition 2. Let \mathcal{K} be a consistent KB containing named concepts C_1, \dots, C_n . A subclass-preserving sub-KB $\mathcal{K}_{sub}(C_i)$ w.r.t. a named concept C_i is a subset of

\mathcal{K} that satisfies the following property: $\mathcal{K} \models (C_i \sqsubseteq C_j) \ 1 \leq j \leq n, j \neq i$, iff $\mathcal{K}_{sub}(C_i) \models (C_i \sqsubseteq C_j)$

Just as in Section 2.1, we would like to produce sub-KBs that are as small as possible. Based on the above definition, we can see that if we have an efficient technique to compute $\mathcal{K}_{sub}(C_i)$, we can obtain and reason over a smaller fragment sufficient to compute all C_i 's subsumers. We can then repeat the process for all named concepts $\{C_1, ..C_n\}$ in the KB, and derive the entire subsumption hierarchy.

We now describe an approach to compute $\mathcal{K}_{sub}(C_i)$. Note that in general, we can prove (by refutation) that a concept C_j subsumes another concept C_i in a consistent *SHIN* KB \mathcal{K} iff the complex concept $(C_i \sqcap \neg C_j)$ is unsatisfiable in \mathcal{K} , i.e., if we construct $\mathcal{K}' \leftarrow \mathcal{K} \cup (C_i \sqcap \neg C_j)(a) \models \perp$, where a is a new individual not present in \mathcal{K} , then \mathcal{K}' is inconsistent. This principle gives us a direct solution to finding $\mathcal{K}_{sub}(C_i)$ using the *Approx-Resolution* algorithm as shown in the description of the Decomposition Algorithm (3):

<p>Algorithm: Decompose Inputs: <i>Horn</i> – <i>SHIN</i> KB \mathcal{K}_1 containing named concepts $C_1..C_n$; specific concept C_i Output: <i>Horn</i> – <i>SHIN</i> KB \mathcal{K}_2</p>
<ol style="list-style-type: none"> 1. Construct $(n - 1)$ DL KBs as follows, $1 \leq j \leq n, j \neq i : \mathcal{K}_j \leftarrow \mathcal{K}_1 \cup \{(C_i \sqcap \neg C_j)(a)\}$ 2. Set $\mathcal{K}_2 \leftarrow \emptyset$ 3. for each $j, 1 \leq j \leq n, j \neq i$, $\mathcal{K}_2 \leftarrow \mathcal{K}_2 \cup \text{Approx-Resolution}(\mathcal{K}_j)$ 4. return \mathcal{K}_2.

Table 3. Decomposition Algorithm

Theorem 2. Let \mathcal{K}_1 be a *Horn*–*SHIN* KB containing named concepts $C_1, ..C_n$ and let $\mathcal{K}_2 \leftarrow \text{Decompose}(\mathcal{K}_1, C_i)$ for some $i, 1 \leq i \leq n$. Then, \mathcal{K}_2 is a subclass-preserving sub-KB of \mathcal{K}_1 w.r.t C_i

Proof. Consider any one KB \mathcal{K}_j constructed in step 1 of the algorithm. Let $\mathcal{K}'_j \leftarrow \text{Approx-Resolution}(\mathcal{K}_j)$. By Theorem 1, \mathcal{K}'_j is an inconsistency-preserving sub-KB of \mathcal{K}_j , and thus $\mathcal{K}_j \models C_i \sqsubseteq C_j$ iff $\mathcal{K}'_j \models \perp$, i.e., \mathcal{K}'_j can be used to check whether C_j is a subsumer of C_i . \mathcal{K}_2 , constructed in step 3, simply represents the sum union of the output of *Approx-Resolution* for each \mathcal{K}_j ($1 \leq j \leq n$), and thus for any $j, \mathcal{K}_j \models C_i \sqsubseteq C_j$ iff $\mathcal{K}_2 \models \perp$. This implies that \mathcal{K}_2 is a subclass-preserving sub-KB of \mathcal{K}_1 w.r.t C_i .

Now, we do not have to actually construct all $(n-1)$ KBs in order to compute $\mathcal{K}_{sub}(C_i)$. A more efficient way to solve this problem is given in Table 4.

<p>Algorithm: Optimized-Decompose Inputs: <i>Horn</i> – <i>SHLN</i> KB \mathcal{K}_1 containing named concepts $C_1..C_n$; specific concept C_i Output: <i>Horn</i> – <i>SHLN</i> KB \mathcal{K}_2</p>
<ol style="list-style-type: none"> 1. Construct a new KB \mathcal{K}' as follows: $\mathcal{K}' \leftarrow \mathcal{K} \cup \{(C_i \sqcap \neg X)(a)\}$, where ‘X’ is a new concept not present in \mathcal{K} 2. $\mathcal{K}_2 \leftarrow \text{Approx-Resolution}(\mathcal{K}')$, with the following modification to step 3 of <i>Approx-Resolution</i>: for each unit positive literal clause A, $A \neq C_i$, derived under unit resolution, assume A resolves with $\neg X$ to produce \perp

Table 4. Optimized Decomposition Algorithm

In the optimized version of the decomposition algorithm the concept ‘X’ plays the role of a ‘variable’ that can take any named concept value, and hence a single run of *Approx-Resolution* is sufficient to find all named concept subsumers of C_i .

2.3 Controlling Number of Components in Ontology Decomposition

The decomposition algorithms presented so far satisfy our key goal of producing a smaller version of the KB on which the subsumption test can be performed. However, to compute the entire subsumption hierarchy, subsumption reasoning must be performed on all the components. For large ontologies, it is not practical to decompose the ontology for each concept. Thus, an important property of our decomposition is to ensure that the number of components produced by our decomposition is not too large. The following Lemma implies that if we consider the asserted ontology class hierarchy as a graph, we only need to build subclass-preserving sub-KBs for the leaf nodes of this graph.

Lemma 1. *Let \mathcal{K} be a consistent KB containing named concepts $C_1, ..C_n$, and suppose $\mathcal{K} \models C_i \sqsubseteq C_j$. Then, $\text{Optimized-Decompose}(\mathcal{K}, C_j) \subseteq \text{Optimized-Decompose}(\mathcal{K}, C_i)$.*

Thus, the entire inferred class hierarchy can be computed bottom-up using only the decompositions for the leaf node concepts. Moreover, if an ontology has numerous leaf nodes, we can merge components for ‘closely related’ leaf-nodes. One possibility is to perform a standard stack-based depth-first search over the entire asserted class hierarchy, record the time at which a concept leaves the stack, and consider concepts with short time-stamp gaps to be closely related. This is the approach we use in our implementation discussed in the next section.

2.4 Extension to Non-Horn Clauses

In this section, we establish that *Approx-Resolution*, which computes an inconsistency-preserving sub-KB and whose correctness has already been proven for *Horn-SHLN* KBs, can be extended to *SHLN* KBs that are not Horn. This result

relies in part on the transformation performed by adding Step 3 to *Approx-Resolution* as shown below. This step approximates a non-Horn clause by a set of Horn clauses. From the correctness of *Approx-Resolution* on any *SHLN* KB, it directly follows that *Decompose* and *Optimized-Decompose* can also be correctly applied to any *SHLN* KB (even non-Horn-*SHLN* KB).

Theorem 3. *Given a TBox T that can be described by a CNF expression E, in general including both Horn and non-Horn clauses, and containing an inconsistency. Let E' the expression obtained from E by applying the third step of the Approx-Resolution Algorithm to non-Horn clauses. E' is comprised entirely of Horn clauses. Furthermore, the union U' of the unit resolutions of the empty clause from E' is such that, for each minimal derivation D of the empty clause in E and for each clause C in D, there is a clause C' used in U' such that either C'= C or C' is one of the Horn-clauses derived from C by applying Step 3 of Approx-Resolution to C.*

Without loss of generality, we assume that if the TBox T contains an inconsistency then there exists a (not necessarily unit) resolution based derivation D of the empty clause from E. We transform E into E', containing only Horn clauses, using Step 3 of the *Approx-Resolution* algorithm. We then show that a corresponding set of unit based resolutions can be done in E'. In the discussion below, we refer to the transformed ontology as the Horn ontology as it contains only Horn clauses. For example the clause ABxy in E would be replaced by the clauses Axy and Bxy in E'. Before proceeding further with the proof, we first prove two lemmas.

Lemma 2. *Let K be a set of clauses, in general including both Horn and non-Horn clauses, from which the empty clause can be derived via resolution in S steps. Let K_i denote the set of clauses after i resolution steps have been done. Thus K_0 is K and K_S is the empty clause. If the i-th resolution step is between two clauses k_{ai} and k_{bi} producing clause k_{ci} , where k_{ai} contains the positive literal being resolved and k_{bi} contains the negative literal being resolved. Thus, $K_{i+1} = (K_i - \{k_{ai}, k_{bi}\}) \cup \{k_{ci}\}$ This lemma asserts that K_i contains a clause comprised entirely of positive literals for all $i = 0, 1, \dots, S-1$.*

Proof. In the last resolution step, $k_{a,S-1}$ is a single positive literal, $k_{b,S-1}$ is a single negative literal and $k_{c,S-1}$ is the empty clause. Thus the lemma is true for K_{S-1} ; $k_{a,S-1}$ is the required clause. If $k_{a,S-1}$ was one of the clauses in K_0 , then it was also in K_i for all i between 1 and S-1, and the lemma is proved. If on the other hand, it was produced as the resolvent in some step j (i.e., it is k_{cj}), then k_{aj} must be a clause comprised entirely of positive literals as k_{aj} contains the same literals as k_{cj} plus the positive literal being resolved. This argument can be repeated back to K_0 . Thus the lemma is proven.

Corollary 1. *When we transform E to E' above, containing only Horn clauses, as in the statement of the Lemma 2, the purely positive clauses contain only a single positive literal and the derivation described there can be transformed into a unit resolution derivation.*

Proof. $k_{c,S-1}$ is a single positive literal. If it was the resolvent of some previous resolution step, say in step j of the derivation, then k_{aj} would have to have two positive literals. This is not possible since all the clauses in the derivation are Horn clauses. Thus, $k_{c,S-1}$ was part of E' itself and was present during the entire derivation. It is therefore possible to use $k_{c,S-1}$ which contains the positive literal X to remove its negated counterpart x from all clauses in E' . This eliminates X and x from the derivation and we are left with a derivation involving one fewer literal. Repeated application of this argument yields a unit resolution derivation.

Lemma 3. *Let D be a minimal derivation of the empty clause starting from some clause C , where C is defined by $C = x_1x_2x_k$, and where the x_i may be either positive or negative literals. Let C' be another clause formed from C by removing a literal, say x_1 . Then a derivation D' of the empty clause can be done starting with C' , by removing from D the resolution step R_1 where x_1 was removed along with resolution steps to remove literals introduced directly or indirectly by R_1 . Furthermore, resolution steps are done in D' in the same order as they are done in D .*

Proof. Both D and D' proceed identically until we reach R_1 . If by the time we reach R_1 , x_1 was reintroduced, then the original removal of x_1 has had no effect; D and D' are identical. So suppose x_1 was not reintroduced. In this case, in D we do R_1 and in D' we do not; in either case, x_1 has been removed. If R_1 did not introduce any new literals, again D and D' have left us in the same state and the two derivations proceed identically to their conclusions. If R_1 did introduce new literals, $y_2 \dots y_n$, we make the same argument we just did for x_1 ; i.e., we proceed until we reach R_2 (corresponding to y_2 , the first of the literals introduced by R_1) and proceed as above. Thus, we remove from D any resolution steps R_k corresponding to literals y_k not present D' . Note that D' still contains all the resolutions needed to remove literals from C' ; i.e., the only resolution steps removed in D' are R_1 and those associated with literals it introduced directly or indirectly.

We now return to the proof of the theorem. Lemma 3 assures us that the removal of literals from an expression does not affect our ability to derive the empty clause through resolution. The Corollary of Lemma 2 assures us that this derivation can be a unit resolution based derivation. The only thing left to show is that the union of such unit derivations covers all clauses in the original derivation.

Before doing this, we give an example of the process. Let

$$E = \{(1)AByz, (2)YDx, (3)Bdx, (4)bz, (5)AZ, (6)a, (7)X\}$$

. E' resulting from our transformation is such that

$$E' = \{(1')Ayz, (1'')Byz, (2')Yx, (2'')Dx, (3')Bdx, (4')bz, (5')A, (5'')Z, (6')a, (7')X\}$$

For a derivation of the empty clause D_0 (see Table 5) in E , the covering derivations in E' are D_1 , D_2 and D_3 presented in Table 5.

D0	D1	D2	D3
$AByz$ (1)	Ayz (1')	Byz (1'')	
YDx (2)	Yx (2')	Yx (2')	Dx (2'')
$ABDxz$	Axz	Bxz	
Bdx (3)			Bdx (3')
$ABxz$			Bx
bx (4)		bx (4')	bx (4')
Axz		xz	xz
AZ (5)	Z (5'')	Z (5'')	Z (5'')
Ax	Ax	x	x
a (6)	a (6')		
x	x		
X (7)	X (7')	X (7')	X (7')
\perp	\perp	\perp	\perp

Table 5. Derivation Example

Note that at least one clause derived from each clause in E appears in the union of the three covering derivations. Thus we can say that the derivations cover all the clauses in E .

We return to the proof of the theorem. Each of the resolution step that reduces E to \perp removes a literal, either a literal present in the first clause in E or a literal introduced by a previous resolution step. Without loss of generality, we assume this literal appears in non-negated form in the first clause (i.e. the first clause is k_{a_0} as described in the proof of Lemma 2). In the example above, the first resolution removes Y from YDx . The transformation to E' produces all possible Horn clauses that can be derived from E . Thus, one of those Horn clauses includes the literal being removed by first resolution; Yx in the example above. Lemma 3 assures us that we can start a derivation of the empty clause with this clause. That derivation covers the first clause. The second clause in this derivation can be any of the Horn clauses produced by the transformation from E to E' . This is true because the second clause contains the negated form of the literal being removed and all the Horn clauses contain all the negated literals in the clause from which they were produced. When we consider the union of all such derivations, we see that we are introducing all the literals introduced by the first resolution step in the original derivation using E . Lemma 3 assures us that all clauses used to remove these literals in the original derivation will also be present in one of the derivations using E' . In the example above, derivation $D1$ covers clauses 1,2,5,6, and 7 corresponding to literals Y , Z , A and X . Similarly, derivation $D2$ covers clauses 1,2,4,5 and 7 corresponding to literals Y , B , Z and X . We see that all clauses and all literals are covered except for D

and clause 3, the clause used in the resolution step removing D in the original derivation. Thus, D is not involved in the first resolution step or any of the other resolutions involving literals involved in this resolution step. It is thus possible to start another derivation (in this case derivation D3) with the first Horn clause containing D; i.e. Dx. This derivation covers clause 3 and the literal D. There are no clauses or literals remaining to be accounted for and we have completed the coverage of all clauses in the original derivation. It will always be the case that a literal is either involved in the first resolution step or that we can start a derivation in E' using a Horn clause containing this literal. Thus we will always be able to cover all clauses in the original derivation and the theorem is proved.

2.5 Example

We demonstrate how our algorithm works with an example. Consider a *SHIN* KB \mathcal{K} containing atomic concepts $A - G$, atomic roles R, S and the 6 axioms given below:

$$\begin{array}{lll} 1 : A \sqsubseteq B \sqcap \forall R.C \sqcap D & 2 : E \sqcup F \sqsubseteq B & 3 : F \sqsubseteq A \sqcup G \\ 4 : D \sqsupseteq 1.S & 5 : \exists R.C \sqcap B \sqsubseteq G & 6 : S \sqsubseteq R \end{array}$$

Our goal is to find all atomic concept subsumers of A .

We first convert \mathcal{K} into an equivalent FOL KB \mathcal{K}^{FOL} (see Table 6), where the DL axioms have been replaced by equivalent FOL clauses (shown in the second column).

DL#	Corresponding clauses in \mathcal{K}^{FOL}	Approx. to \mathcal{K}^{PL}	$\mathcal{K}^{PL-Horn}$
1	$\neg A(x) \vee B(x), \neg A(x) \vee \neg R(x, y) \vee C(y),$ $\neg A(x) \vee D(x)$	$\neg A \vee B, \neg A \vee \neg R \vee C,$ $\neg A \vee D$..
2	$\neg E(x) \vee B(x), \neg F(x) \vee B(x)$	$\neg E \vee B, \neg F \vee B$..
3	$\neg F(x) \vee A(x) \vee G(x)$	$\neg F \vee A \vee G$	$\neg F \vee A, \neg F \vee G$
4	$\neg D(x) \vee (S(x, f1(x)) \wedge S(x, f2(x)) \wedge$ $(f1(x) \neq f2(x)))$	$\neg D \vee S$..
5	$\neg R(x, y) \vee \neg C(y) \vee B(x) \vee G(x)$	$\neg R \vee \neg C \vee B \vee G$	$\neg R \vee \neg C \vee B,$ $\neg R \vee \neg C \vee G$
6	$\neg S(x, y) \vee R(x, y)$	$\neg S \vee R$..

Table 6. Converting *SHIN* DL axioms into FOL and then PL clauses

In the table, each FOL clause has an implied universal quantification over each of the variables in it. For example, consider the part $A \sqsubseteq \dots \forall R.C.$ in the DL axiom 1. Here, the universal restriction in the RHS translates into the implication $\forall x, \forall y : R(x, y) \rightarrow C(y)$, and in combination with A on the LHS of the axiom produces the disjunctive FOL clause $\forall x, \forall y : \neg A(x) \vee \neg R(x, y) \vee C(y)$.

Note that the disjunction in the RHS of axiom 3 and the conjunction in the LHS of axiom 5 produces non-Horn FOL clauses for both these cases. Also, the presence of the min-cardinality restriction on role S in axiom 4 gives rise to two

skolem functions $f1(x), f2(x)$ for creating at least two distinct S -successors of x .

After creating \mathcal{K}^{FOL} , we approximate it into a propositional KB \mathcal{K}^{PL} (shown in the third column of Table 6), by ignoring all variables and constants appearing in the FOL clauses. Finally, the non-Horn PL clauses in \mathcal{K}^{PL} are approximated further into Horn clauses by treating the disjunction of positive literals as a conjunction, and thus splitting the clauses further as shown in the last column of Table 6, producing KB $\mathcal{K}^{PL-Horn}$.

Now, our goal is to find $A \sqsubseteq X$, where X stands for any atomic concept (except A) in \mathcal{K} . Thus, we follow the standard proof-by-refutation procedure of adding $A \wedge \neg X$ to $\mathcal{K}^{PL-Horn}$ (where X acts as a ‘variable’ for other atomic concepts), and finding all inconsistencies due to this addition. That is, we perform unit resolution over the modified Horn KB to find *all* derivations of \perp .

Table 7 shows the sets of clauses that resolve to \perp , and the corresponding literal that resolves with $\neg X$ in each derivation. Note that the superscript on each PL clause depicts the original DL axiom it was derived from.

Set of clauses leading to \perp	$\neg X..$
$A, \neg X, (\neg A \vee B)^1$	B
$A, \neg X, (\neg A \vee D)^1$	D
$A, \neg X, (\neg A \vee D)^1, (\neg D \vee S)^4$	S
$A, \neg X, (\neg A \vee D)^1, (\neg D \vee S)^4, (\neg S \vee R)^6$	R
$A, \neg X, (\neg A \vee D)^1, (\neg D \vee S)^4, (\neg S \vee R)^6, (\neg A \vee \neg R \vee C)^1$	C
$A, \neg X, (\neg A \vee D)^1, (\neg D \vee S)^4, (\neg S \vee R)^6, (\neg A \vee \neg R \vee C)^1, (\neg R \vee \neg C \vee G)^5$	G

Table 7. Unit resolution to produce all derivations of \perp

Thus, the algorithm finds potential concept subsumers of A : B, D, C, G (since R, S are actually atomic roles in \mathcal{K}) and the axioms necessary to derive accurate subsumers of A : $\{1, 4, 5, 6\}$ (Note: axioms 2, 3 are excluded). These four axioms are then fed to a sound and complete tableau reasoner to derive the real subsumers of A : B, D, G .

3 Computational Experience

We performed experiments on an Intel Pentium-M 1.6GHz processor with 2GB RAM running Windows XP on the following OWL-DL ontologies:

SNOMED CT and NCI⁷ are large and popular OWL ontologies in the medical domain. Galen⁸ and NASA’s SWEET-JPL ontology⁹ use almost all of OWL-DL

⁷ <http://www.mindswap.org/2003/CancerOntology/nciOncology.owl>

⁸ <http://www.cs.man.ac.uk/horrocks/OWL/Ontologies/galen.owl>

⁹ <http://sweet.jpl.nasa.gov/ontology/>

Ontology	#concepts (NC)	#axioms (NA)	expressivity
SNOMED CT	379638	379640	<i>Horn - ALC</i>
NCI	59898	86574	<i>Horn - ALC</i>
Galen	2749	4353	<i>Horn - SHF</i>
SWEET-JPL	1639	1807	<i>Non - Horn - SHIOF(D)</i>

Table 8. Ontologies Decomposed

expressivity. Since we are not dealing with ABoxes, we modified SWEET-JPL by converting the 56 hasValue constraints to someValuesFrom constraints and all instances of types other than Class to type Class; this increases the size of the ontology slightly.

We define two metrics α and β , which measure how well we have decomposed the and the overhead incurred by decomposing respectively. Specifically, $\alpha = NL / NA$ and $\beta = ND / NC$, where NA is number of axioms in the ontology, NC is the number of named concepts, NL is the number of axioms in the largest component after decomposition, and ND is the total number of concepts in all the components. Ideally, we would like both α and β to be as small as possible, although we expect a tradeoff: if we increase the granularity of our decomposition so that individual components contain fewer axioms (causing α to decrease), we expect the total number of classes across all components to increase (forcing β to increase), resulting in more subsumption tests on smaller sized KBs to obtain the entire class subsumption hierarchy.

The results of the resolution-based decomposition algorithm *Optimize-Decompose* on SNOMED-CT are summarized in the Table 9. We used our DFT-based algorithm, and in particular, the leaf-width metric, to vary the number of components produced by our decomposition, in order to examine the tradeoff between α and β .

Table 9 (Row 1) shows that by using a leaf-width of 10000, SNOMED CT can be decomposed into 76 components, the largest of which is around 10% the size of the total ontology ($\alpha = 0.1$) without having to slightly more than double the total number of concepts ($\beta = 2.3$). This is a very attractive tradeoff, since the order of complexity of reasoning algorithms grows non-linearly with the size of the problem, we should expect a significant reduction in the overall effort. Also, the total runtime of the algorithm is between 3-4 minutes.

leaf width	# components	NL	ND	ND_{max}	$\alpha (NL/NA)$	$\beta (ND/NC)$
10000	76	43819	873082	23677	0.11	2.29
30000	25	76740	652417	42256	0.202	1.71
50000	16	99064	573604	55203	0.26	1.51

Table 9. SNOMED CT - NC = 379638 concepts; NA = 379640 axioms. **All runtimes took between 198s and 237s**

To determine the quality of the components created by our decomposition, we compared our components with top-level semantic biomedical categories. To obtain top-level categories, we mapped SNOMED CT to UMLS [2], which is a meta-thesaurus for the medical domain and specifies 14 top level biomedical categories. SNOMED-CT most frequently uses the semantic categories: *Disorders*, *Procedure*, *Chemicals and Drugs*, *Living Beings/Organisms*, and *Anatomy*. We analyzed the SNOMED-CT decomposition with 25 components to determine the coverage of these top 5 categories across different components. The graph in Figure 1 shows the distribution of the categories.

The semantic categories are cleanly distributed across different components, for example, *Living Being* appears only in components 1, 2, 3 and 21. Furthermore, a set of categories never appear together in a single component, for example *Disorder* and *Chemicals* are independent of one another. This has significant implications for biomedical applications that only deal with specific subset of concepts in SNOMED-CT, for example Laboratory data dealing with a specific domain such as *Living Beings*, and that only cares about concept subsumption information, which is typically the case, can use the components 1,2,3 and 21 only instead of the entire SNOMED CT. The result is also interesting from perspective that the partitioning of SNOMED CT was performed based on a logic procedure that resulted in domain-based partitions.

We then did experiments to determine the effects of decomposition. First, we wanted to validate the decomposition procedure by checking that the derived class hierarchy remained unchanged when the ontology was decomposed. Also, we wanted to examine the improvements in running time and memory requirements obtainable via decomposition.

Table 10 shows the results of these experiments on NCI, GALEN, and SWEET-JPL; SNOMED-CT was not included because memory limitations prevented us from running Pellet on it on our laptop computer. Also, for the same reason, when considering NCI-THESAURUS, we removed leaf concepts (i.e. concepts only involved in subsumption relationships and with no subsumees except other leaf concepts). Times for T-decomp and T-reason in Table 10 are given in seconds. As can be seen from examining Table 10, in all cases, the time to decompose the ontology was roughly an order of magnitude smaller than the time to reason over it. Also, in all cases, there was a significant reduction in reasoning time. In the case of JPL, the reduction was over an order of magnitude. In the case of NCI, the reasoner was unable to compute the derived class hierarchy at all within a 12 hour time period after which we stopped the experiment.

To compare the sizes of the decomposition components we obtained with those obtained by [3], we formed the closures of each of the concepts in JPL, GALEN and NCI-ONCOLOGY. The distribution of component sizes matched closely with those in [3]. These results Table 11 are not surprising as the rules

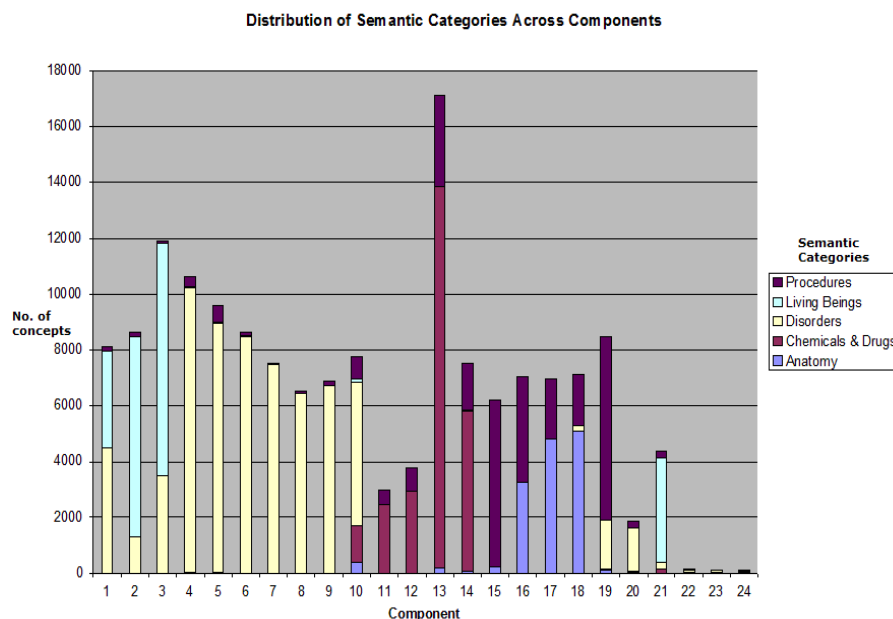


Fig. 1. Domain-based Distribution of SNOMED CT components

used in both approaches are very similar when applied to these ontologies. The average sizes of the components for each of the three ontologies were also very similar, with those obtained by our approach being roughly 10% smaller on average.

4 Related Work

The problem of decomposing OWL ontologies has received a lot of attention in recent years. [4] describes an algorithm for partitioning an ontology by treating the ontology as a graph, and examining dependencies between classes that appear as nodes in the graph based on certain heuristics. On a similar note, [5], [6] provides techniques to extract ontology modules based on structural analysis of the asserted ontology graph (i.e., traversing related classes and properties through term definitions), and more recently, [7] uses structural analysis to get rid of irrelevant axioms fed to an FOL Prover (Vampire) to perform standard DL reasoning tasks such as classification. The main problem with these approaches is that either they do not provide strong logical guarantees about the ‘module’ computed, or in cases where they do, the modules are larger than necessary due

Ontology	# comp.	NL	ND	ND_{max}	$\alpha (NL/NA)$	$\beta (ND/NC)$	T-decomp	T-reason
GALEN	1	4353	2749	2749	1.00	1.00		58
GALEN	3	3059	3329	1613	0.70	1.21	1.6	43
GALEN	11	2553	3453	1460	0.59	1.26	2.6	49
JPL	1	1807	1639	1639	1.00	1.00		122
JPL	5	994	1859	924	0.55	1.13	0.7	53
JPL	10	321	2277	303	0.18	1.36	0.8	3.5
NCI	1	86574	59898	59898	1.00	1.00		> 43200
NCI	20	39363	71009	22018	0.45	1.19	866	10063
NCI	29	25180	80161	16677	0.29	1.34	879	5730
NCI	62	23652	89948	16066	0.27	1.50	927	5530

Table 10. Decomposition of JPL, GALEN and NCI

Ontology	0-49	50-99	100-149	150-199	200-249	250-299
JPL	1624	15				
GALEN	1864	637	205	25	17	1
NCI	25215	1941	566	29	1	

Table 11. Distribution of component sizes

to the conservative nature of structural analysis. For example, in [7], all General Concept Inclusion (GCI) axioms in the ontology are considered relevant for a concept subsumption test, whereas our resolution-based analysis helps prune out irrelevant GCIs as well.

[3] presents a formal definition of module for an atomic concept that is more general than ours, e.g., it entails all atomic subsumers as well as subsumees of the concept. This causes the modules created using their approach to be much larger than what we need. Also, the solution produces a strict partitioning of the ontology, whereas we allow for possibly overlapping components for our decomposition task.

[8] defines modules for concepts that do not preserve all its atomic superclass entailments, and also uses inferences computed by a reasoner in order to build a module. For both these reasons, it does not fit in with our scenario.

More recently, [9] describes a method for extracting modules based on the notion of conservative extension of the ontology. The authors provide a definition of an S-module (where S is a signature) of an ontology w.r.t a language L, show its relation to model and deductive conservative extensions of ontologies, and also describe an elegant technique to find syntactical locality-based S-module in the ontology. Proposition 8 in [9] is directly related to our work, since it states that locality-based modules preserve complete information to find all superclasses of an atomic concept and thus can be used to optimize classification. However, these modules preserve much more information (all entailments involving the concept) than what is needed for classification, and thus may be larger. Also, since we focus only on the subsumption entailment in a single direction, we can make use of Lemma 1 to speed up the decomposition process by building \mathcal{K}_{sub}

for leaf concepts alone, a property that does not hold for locality-based modules defined in [9] which requires computing modules for all atomic concepts in the ontology. Finally, we note that since our definition of \mathcal{K}_{sub} only preserves subclass entailments, it is weaker than the general notion of conservative extension, i.e., \mathcal{K} is not necessarily a conservative extension of $\mathcal{K}_{sub}(C)$ (for any concept C).

5 Conclusions and Future Work

We have presented a novel resolution-based algorithm for OWL ontology decomposition, applied to the problem of determining the implied class hierarchy. For the medical OWL ontology SNOMED CT, the resulting components are approximately an order of magnitude smaller than the original ontology, while the total number of concepts in all components of the decomposition is only a small multiple of the number of concepts in the original ontology. Similar positive results were seen in the decomposition of the more expressive JPL and NCI ontologies.

In the future, we would like to test the effects of decomposition using other reasoners (e.g. RACER [10]) and to explore the additional gains obtainable from incorporating decomposition directly into the reasoner itself.

We would also like to extend our solution to include ABoxes as well, and apply it to ABox reasoning tasks such as query answering. Applying our technique to an ABox is a challenge, since all concept-membership assertions $C(a)$ (resp. role assertions $R(a, b)$) in the ABox are abstracted to the proposition C (resp. R). This has the effect that if the proposition C (resp. R) is considered relevant to generate the empty clause under unit resolution over the propositional KB, then *all* concept-membership assertions involving C (resp. role assertions involving R) will be included in the output KB. Since typically, the ABox is many orders of magnitude larger than the TBox/RBox, this approach is clearly not feasible.

To overcome this challenge, we plan to explore combining the TBox/RBox decomposition algorithm (*Approx-Resolution*) with our earlier ABox summarization technique [11], which dramatically reduces the size of an ABox needed for reasoning. A summary ABox \mathcal{A}' is an abstraction of the original ABox \mathcal{A} , such that if \mathcal{A}' is consistent, then \mathcal{A} must also be consistent, and thus we can often reason on \mathcal{A}' instead of \mathcal{A}^{10} . Our goal is to try to apply *Approx-Resolution* to the KB $\mathcal{K}' \leftarrow \mathcal{T} \cup \mathcal{R} \cup \mathcal{A}'$.

References

1. Sirin, E., Parsia, B.: Pellet: An owl dl reasoner. In: Description Logics. (2004)
2. DA, D.L., Humphreys, B., McCray, A.: The unified medical language system. *Methods Inf Med.* **32(4)** (1993) 281–291

¹⁰ The converse, however, is not true and if \mathcal{A}' is inconsistent, we use a *refinement* process which involves making \mathcal{A}' more precise w.r.t the original \mathcal{A} , in order to confirm whether \mathcal{A} is indeed inconsistent.

3. Grau, B.C., Parsia, B., Sirin, E., Kalyanpur, A.: Automatic partitioning of owl ontologies using e-connections. In: *Description Logics*. (2005)
4. Stuckenschmidt, H., Klein, M.: Structure-based partitioning of large class hierarchies. In: *ISWC*. (2004)
5. Noy, N., Musen, M.: The prompt suite: Interactive tools for ontology merging and mapping. In: *International Journal of Human-Computer Studies*. (2003)
6. Seidenberg, J., Rector, A.: Web ontology segmentation: Analysis, classification and use. In: *WWW*. (2006)
7. Tsarkov, D., Riazanov, A., Bechhofer, S., Horrocks, I.: Using Vampire to reason with OWL. In McIlraith, S.A., Plexousakis, D., van Harmelen, F., eds.: *Proc. of the 3rd International Semantic Web Conference (ISWC 2004)*. Number 3298 in *Lecture Notes in Computer Science*, Springer (2004) 471–485
8. Mathieu d'Aquin, Marta Sabou, E.M.: Modularization: a key for the dynamic selection of relevant knowledge components. In: *First International Workshop on Modular Ontologies, ISWC*. (2006)
9. Bernardo Cuenca Grau, Ian Horrocks, Y.K., Sattler, U.: Extracting modules from ontologies: A logic-based approach. In: *WWW*. (2007)
10. Haarslev, V., Moller, R.: Racer system description. *Conf. on Automated Reasoning (IJCAR)* (2001) 701–705
11. A.Fokoue, A.Kershenbaum, L.Ma, E.Schonberg, K.Srinivas: The summary abox:cutting ontologies down to size. *Proc. of the Int. Semantic Web Conf. (ISWC)* (2006) 136–145