Optimising Ontology Classification

Birte Glimm, Ian Horrocks, Boris Motik, and Giorgos Stoilos

Oxford University Computing Laboratory, UK

Abstract. Ontology classification—the computation of subsumption hierarchies for classes and properties—is one of the most important tasks for OWL reasoners. Based on the algorithm by Shearer and Horrocks [9], we present a new classification procedure that addresses several open issues of the original algorithm, and that uses several novel optimisations in order to achieve superior performance. We also consider the classification of (object and data) properties. We show that algorithms commonly used to implement that task are incomplete even for relatively weak ontology languages. Furthermore, we show how to reduce the property classification problem into a standard (class) classification problem, which allows reasoners to classify properties using our optimised procedure. We have implemented our algorithms in the OWL HermiT reasoner, and we present the results of a performance evaluation.

1 Introduction

Ontology classification—the computation of subsumption hierarchies for classes and properties—is a core reasoning service provided by all OWL reasoners known to us. The resulting class and property hierarchies are used in ontology engineering, where they help users to navigate through the ontology and identify errors, as well as in tasks such as explanation and query answering.

Significant attention has been devoted to the optimisation of individual subsumption tests; however, most OWL reasoners solve the classification problem using an enhanced traversal (ET) classification algorithm similar to the one used in early description logic reasoners [1]. This can be inefficient when classifying large ontologies: even if each subsumption test is very efficient, the extremely large number of tests performed by ET can make classification an expensive operation. Moreover, with the exception of HermiT, all OWL reasoners we are aware of construct property hierarchies simply by computing the reflexive-transitive closure of the subproperty axioms occurring in the ontology—a procedure that is incomplete for each ontology language that supports existential restrictions (someValuesFrom), functional properties, and property hierarchies.

In order to address some of the problems of ET on large ontologies, an alternative classification algorithm, called KP, was proposed recently [9]. Unlike ET, KP does not construct the hierarchy directly; instead, it maintains the sets of known (K) and possible (P) subsumer pairs, and it performs subsumption tests to augment K and reduce P until the two sets coincide. To further reduce the number of tests, KP exploits the transitivity of the subclass relation to propagate (non-)subsumptions and thus speed up the convergence of K and P.

In this paper we address several issues that were left open in the work on KP, we present an optimised version of the resulting algorithm, and we evaluate its implementation in the HermiT reasoner. The new algorithm exhibits a consistent performance improvement over ET, and in some cases it reduces overall classification times by a factor of more than ten.

We then turn our attention to the classification of object and data properties. We show that merely computing the reflexive-transitive closure of the asserted hierarchies produces an incomplete hierarchy, and we discuss why the ET and KP algorithms do not perform well when applied to property classification. We then present a novel encoding of the property classification problem into a class classification problem, which allows us to exploit our new classification algorithm to correctly and efficiently compute property hierarchies. We have implemented our property classification algorithm in HermiT, thus making HermiT the only OWL reasoner we are aware of that correctly classifies object and data properties.

2 Preliminaries

An OWL 2 ontology consists of a set of axioms that describe the domain being modelled. For a full definition of OWL 2, please refer to the OWL 2 Structural Specification and Direct Semantics [7, 6]; here we present only several examples of typical OWL axioms in the OWL 2 Functional Syntax:

$$\text{SubClassOf(Human Animal)} \tag{1}$$

$$\text{DataPropertyAssertion(}\textit{age}\text{ Alex “27”}\hat{}\hat{}\text{xsd:integer)} \tag{2}$$

$$\text{ObjectPropertyRange(}\textit{colour}\text{ ObjectOneOf(red green blue))} \tag{3}$$

Axiom (1) states that the class Human is a subclass of the class Animal (i.e., that all Humans are Animals); axiom (2) states that the individual Alex is related to the integer 27 by the data property *age* (i.e., that the age of Alex is 27); finally, axiom (3) states that the range of the object property *colour* consists of red, green, and blue (i.e., that the colour of an object can only be red, green, or blue). Concrete values such as the *literal* “27”^^xsd:integer in the above example are taken from the OWL 2 *datatype map*, which contains most of the XML Schema datatypes plus certain OWL-specific datatypes.

The interpretation of axioms in an OWL ontology $\mathcal{O}$ is given by means of two-sorted interpretations over the *object domain* and the *data domain*, where the latter contains concrete values such as integers, strings, and so on. An *interpretation* maps classes to subsets of the object domain, object properties to pairs of elements from the object domain, data properties to pairs of elements where the first element is from the object domain and the second one is from the data domain, individuals to elements in the object domain, a datatype to a subset of the data domain, and a literal (a data value) to an element in the data domain. For an interpretation to be a *model* of the ontology, several conditions have to be satisfied [6]. For example, if $\mathcal{O}$ contains SubClassOf(C D), then the interpretation of C must be a subset of the interpretation of D. If the axioms

of $\mathcal{O}$ cannot be satisfied in any interpretation (i.e., if $\mathcal{O}$ has no model), then $\mathcal{O}$ is *inconsistent*; otherwise, $\mathcal{O}$ is *consistent*. If the interpretation of a class C is necessarily a subset of the interpretation of a class D in all models of $\mathcal{O}$, then we say that $\mathcal{O}$ entails $C \sqsubseteq D$ and write $\mathcal{O} \models C \sqsubseteq D$. If the interpretations of C and D necessarily coincide, we write $\mathcal{O} \models C \equiv D$. A class C is *satisfiable* if a model of $\mathcal{O}$ exists in which the interpretation of C is non-empty; otherwise, C is *unsatisfiable*. We use analogous notations for object and data properties. For full details of the OWL 2 Direct Semantics, please refer to the OWL 2 Direct Semantics specification [6]. We use $\mathbf{C}_\mathcal{O}$ to denote the set of classes that occur in $\mathcal{O}$ extended with owl:Thing and owl:Nothing; similarly, we use $\mathbf{OP}_\mathcal{O}$ (resp. $\mathbf{DP}_\mathcal{O}$) to denote the sets of object (resp. data) properties occurring in $\mathcal{O}$ extended with owl:TopObjectProperty and owl:BottomObjectProperty (resp. owl:TopDataProperty and owl:BottomDataProperty).

We next illustrate these definitions by means of an example. Let $\mathcal{O}$ be an ontology containing axioms (4) and (5); then, $\mathcal{O}$ entails $C \sqsubseteq E$ even though this is not stated explicitly. This is because axiom (4) ensures that in every model of $\mathcal{O}$, an instance i of C must be related to an instance of the class D with the property *op*. Since i has an *op*-successor, the property domain axiom (5) ensures that i is also an instance of the class E, and hence that C is contained in E.

$$\text{SubClassOf}(C\ \text{ObjectSomeValuesFrom}(op\ D)) \tag{4}$$

$$\text{ObjectPropertyDomain}(op\ E) \tag{5}$$

2.1 The KP Classification Algorithm

Classification of an ontology $\mathcal{O}$ computes all pairs of classes $\langle C, D\rangle$ such that $\{C, D\} \subseteq \mathbf{C}_\mathcal{O}$ and $\mathcal{O} \models C \sqsubseteq D$; similarly, object (resp. data) property classification of $\mathcal{O}$ computes all pairs of object (resp. data) properties $\langle R, S\rangle$ such that $\{R, S\} \subseteq \mathbf{OP}_\mathcal{O}$ (resp. $\{R, S\} \subseteq \mathbf{DP}_\mathcal{O}$) and $\mathcal{O} \models R \sqsubseteq S$. For example, given an ontology containing (4) and (5), a classification algorithm should compute

$$\{\langle \text{owl:Nothing}, C\rangle, \langle \text{owl:Nothing}, D\rangle, \langle C, E\rangle, \langle E, \text{owl:Thing}\rangle, \langle D, \text{owl:Thing}\rangle\}.$$

The recently proposed KP algorithm [9] extends the standard ET algorithm [1]. The KP algorithm maintains two binary relations K and P over $\mathbf{C}_\mathcal{O}$ such that, at any point during algorithm's execution, $\langle C, D\rangle \in K$ implies that $\mathcal{O} \models C \sqsubseteq D$ is known for certain, and $\langle C, D\rangle \in P$ implies that $\mathcal{O} \models C \sqsubseteq D$ is possible (i.e., no evidence to the contrary has been uncovered thus far). In particular, $\langle C, D\rangle \notin P$ means that $\mathcal{O} \not\models C \sqsubseteq D$ is known, so $P \setminus K$ contains all pairs $\langle C, D\rangle$ such that $C \sqsubseteq D$ is possible but not yet known. The algorithm expands K and reduces P until $K = P$, at which point $\mathcal{O} \models C \sqsubseteq D$ iff $\langle C, D\rangle \in K$. Roughly speaking, the algorithm chooses an unclassified class C (i.e., one where a class D exists such that $\langle C, D\rangle \in P \setminus K$), generates a partial hierarchy $\mathcal{H}_C$ of all unknown possible subsumers of C, and applies the standard ET procedure to insert C into $\mathcal{H}_C$. The newly computed subsumption and non-subsumption relations are then used to extend K and reduce P.

Algorithm 1 Prune Additional Possible Subsumptions

Algorithm: pruneNonPossible(P, K, V, N)
Input: P: a set of possible subsumptions to be pruned, K: a set of known subsumptions, V: a set of new positive subsumptions, N: a set of new non-subsumptions
1 **for each** $\langle C, D\rangle \in N$ **do**
2 **for each** E, F such that $\langle C, E\rangle \in K$ and $\langle F, D\rangle \in K$ remove $\langle E, F\rangle$ from P
3 **for each** $\langle C, D\rangle \in V$ **do**
4 **for each** $\langle D, E\rangle \in P$ **do**
5 **if** $\langle E, F\rangle \in K$ and $\langle C, F\rangle \notin P$ **then** remove $\langle D, E\rangle$ from P
6 **for each** $\langle E, C\rangle \in P$ **do**
7 **if** $\langle F, E\rangle \in K$ and $\langle F, D\rangle \notin P$ **then** remove $\langle E, C\rangle$ from P

The algorithm exploits the transitivity of $\sqsubseteq$ to reduce the number of subsumption tests needed to make K and P converge: whenever K is extended with fresh tuples it is also transitively closed, and a pruning strategy is used to remove tuples from P that correspond to obvious non-subsumptions. For example, if $\{\langle C, D\rangle, \langle E, F\rangle\} \subseteq K$, then $\langle D, E\rangle \in P$ implies $\langle C, F\rangle \in P$ since, by the transitivity of $\sqsubseteq$, adding $\langle D, E\rangle$ to K requires $\langle C, F\rangle$ to be added as well; but then, $\langle C, F\rangle \notin P$ implies $\langle D, E\rangle \notin P$. Analogously, if $\langle C, D\rangle \in P$, $\langle E, F\rangle \in K$ and $\langle C, F\rangle \notin P$, then $\langle C, D\rangle \in K$ implies $\langle D, E\rangle \notin P$. The complete pruning strategy of KP is shown in Algorithm 1. Note that this algorithm consists of several nested loops that iterate over potentially very large relations, which can make the algorithm inefficient in practice.

An important question when using KP is how to initialise K and P. The authors suggested to exploit the information generated by (hyper)tableau reasoners. In particular, when testing the satisfiability of a class A, (hyper)tableau algorithms usually initialise a node s_0 with the label $\mathcal{L}(s_0) = \{A\}$ and then apply expansion rules in order to try to construct a *pre-model*—an abstraction of a model for A; if a pre-model is constructed, then the (possibly expanded) label $\mathcal{L}(s_0)$ may provide information about subsumers and non-subsumers of A (if a pre-model cannot be constructed, then A is unsatisfiable and is equivalent to owl:Nothing). More precisely, if $\mathcal{L}(s_0)$ does not contain a class B, then we can infer the non-subsumption $A \not\sqsubseteq B$. Similarly, if B was deterministically added to $\mathcal{L}(s_0)$ (i.e., if no non-deterministic expansion was involved), then we can infer $A \sqsubseteq B$. Consequently, one can initially perform a satisfiability test for all the classes in $\mathbf{C}_\mathcal{O}$ and use the resulting pre-models to initialise K and P. It is not clear, however, whether it is generally efficient to perform all these tests.

3 Optimised Classification

We now present a new classification algorithm that we have implemented in the HermiT reasoner. Our algorithm is based on KP, but it addresses several open problems and incorporates numerous refinements and optimisations. The latter include, for example, a more efficient strategy for initialising K and P, a

practical approach to pruning P, and several heuristics. We next describe our new algorithm and then contrast it with the relevant parts of KP.

Our approach is shown in Algorithm 2. Like KP, our algorithm maintains a set K of known and a set P of possible subsumption pairs. The algorithm uses an OWL reasoner to check satisfiability of classes (line 6) or subsumption between classes (line 25) using the well-known reduction of class subsumption to class satisfiability. In lines 2, 16, 24, 35 and 37, the algorithm manipulates K and P using operations that are defined next.

Definition 1. *Let U be a set of elements and let $R \subseteq U \times U$ be a binary relation over U. The set* $\mathsf{reachable}(C, R)$ *of elements* reachable *from $C \in U$ in R contains all $D \in U$ for which a* path $\{\langle C, C_1\rangle, \langle C_1, C_2\rangle, \ldots, \langle C_n, D\rangle\} \subseteq R$ *exists.*

Let $\sim$ be a relation over U defined as follows: $C \sim D$ if and only if $D = C$, or $D \in \mathsf{reachable}(C, R)$ and $C \in \mathsf{reachable}(D, R)$. Let $[C] := \{D \in U \mid D \sim C\}$ be the set of elements equivalent to C under $\sim$, and let $U_\sim := \{[C] \mid C \in U\}$. The relation $R_\sim$ induced by $\sim$ on R is defined as $R_\sim := \{\langle [C], [D]\rangle \mid \langle C, D\rangle \in R\}$.

The hierarchy *in R is the triple* $\mathsf{hierarchy}(R) = (V, \mathcal{H}, \rho)$ *where $V \subseteq U$ contains exactly one arbitrarily chosen element $C \in [D]$ for each $[D] \in U_\sim$, ρ maps each $C \in V$ into $\rho(C) = [C]$, and $\mathcal{H}$ is a transitively-reduced strict partial order over V such that $\langle C, D\rangle \in \mathcal{H}$ if and only if $\langle \rho(C), \rho(D)\rangle \in R_\sim$.*

The projection $\mathsf{project}(R, S)$ *of R to a set $S \subseteq U$, and the* range $R[C]$ *of an element $C \in U$ in R are defined as follows:*

$$\mathsf{project}(R, S) = \{\langle C, D\rangle \mid C, D \in S \text{ and } D \in \mathsf{reachable}(C, R)\}$$
$$R[C] = \{D \mid \langle C, D\rangle \in R\}$$

Intuitively, $\mathsf{hierarchy}(K)$ extracts from K sets of classes for which $\mathcal{O} \models C \equiv D$ is known and then chooses one representative from each set to construct a transitively-reduced strict partial older.

Our algorithm can be roughly divided into two parts. Lines 1–15 are responsible for the initialisation of K and P using a novel heuristic, and lines 16–37 are responsible for extending K and reducing P using a mixture of the ET algorithm—as in KP—and a new technique for pruning P.

The Initialisation Phase In KP, relations K and P are initialised by performing a satisfiability test for each atomic class in $\mathcal{O}$. Although modern reasoners can usually perform individual tests quite efficiently, the initialisation time can become large if there are many classes, so it is beneficial to avoid unnecessary tests whenever possible. For example, if $C \sqsubseteq D$ and C is satisfiable, then the pre-model constructed by a (hyper)tableau satisfiability test for C will also be a pre-model for D and for every other class occurring in the pre-model. We can thus avoid performing satisfiability tests for the classes outlined above, and from the pre-model for C we can read off information about the possible subsumers of all classes occurring in the pre-model. In order to maximise the effect of this optimisation, we first check the satisfiability of classes that are likely to be classified near the bottom of the hierarchy: such classes are likely to produce larger

Algorithm 2 New Classification Algorithm

Algorithm: Classify($\mathcal{O}$)
Input: $\mathcal{O}$: an ontology to be classified

K := performStructuralSubsumption($\mathcal{O}$)
$(V, \mathcal{H}, \rho)$:= hierarchy(K)
Initialise a list ToTest := $\{C \mid \langle \text{owl:Nothing}, C\rangle \in \mathcal{H}\}$, Unsat := $\emptyset$, and $P := \emptyset$
while ToTest $\neq \emptyset$ **do**
  Iteratively remove the head C from ToTest until C is found such that $P[C] = \emptyset$
  $\mathcal{A}$:= buildModelFor($C(s_0)$)
  if $\mathcal{A} = \emptyset$ **then** // C is unsatisfiable
    for each $\langle C, D\rangle \in \mathcal{H}$ **do** add D to the front of ToTest
    for each descendant E of C in $\mathcal{H}$ that is not already in Unsat **do**
      Add $\langle E, \text{owl:Nothing}\rangle$ to K, add E to Unsat, and remove E from ToTest
  else
    for each $D \in \mathcal{L}(s_0)$ that was derived deterministically **do** add $\langle C, D\rangle$ to K
    for each s in $\mathcal{A}$ and for each $D \in \mathcal{L}(s)$ **do**
      if $P[D] = \emptyset$ **then** $P[D] := \mathcal{L}(s) \cap \mathbf{C}_{\mathcal{O}}$
      else $P[D] := P[D] \cap \mathcal{L}(s)$
for each $D \in \mathbf{C}_{\mathcal{O}}$ and for each $E \in$ reachable(D, K) **do** set $P[D] := P[D] \setminus \{E\}$
UnClass := $\{C \in \mathbf{C}_{\mathcal{O}} \mid P[C] \neq \emptyset\}$
while UnClass $\neq \emptyset$ **do**
  Choose some $C \in$ UnClass and set $B := P[C]$
  $\mathcal{A}$:= buildModelFor($(C \sqcap \neg F)(s_0)$) with F the conjunction of all concepts in B
  if $\mathcal{A} \neq \emptyset$ **then** // all possible subsumers of C are non-subsumers
    for each s in $\mathcal{A}$ and each $D \in \mathcal{L}(s)$ **do** set $P[D] := P[D] \cap \mathcal{L}(s)$
  else
    $(V, \mathcal{H}, \rho)$:= hierarchy(project($K, B \cup \{\text{owl:Nothing}, \text{owl:Thing}\}$))
    Initialise a queue Q with $Q := \{\text{owl:Thing}\}$
    while $Q \neq \emptyset$ **do**
      Remove the head H from Q
      for each D such that $\langle D, H\rangle \in \mathcal{H}$ and $D \in P[C]$ **do**
        $\mathcal{A}$:= buildModelFor($(C \sqcap \neg D)(s_0)$)
        if $\mathcal{A} \neq \emptyset$ **then** // $C \sqcap \neg D$ was satisfiable—that is, $C \not\sqsubseteq D$
          for each s in $\mathcal{A}$ and each $D \in \mathcal{L}(s)$ **do** set $P[D] := P[D] \cap \mathcal{L}(s)$
        else
          Add $\langle C, D\rangle$ to K, and add D to the end of Q
  $P[C] := \emptyset$
  for each $D \in$ UnClass and $E \in$ reachable(D, K) **do** set $P[D] := P[D] \setminus \{E\}$
  Remove from UnClass each D such that $P[D] := \emptyset$
return hierarchy(K)

pre-models that are richer in (non-)subsumption information and that can be used as pre-models for many other classes.

Our algorithm implements this idea as follows. First, it applies a simple structural subsumption algorithm to identify the obvious subsumptions in $\mathcal{O}$ and thus instantiate K. Then, it extracts a class hierarchy $\mathcal{H}$ from K and collects all classes C such that $\langle \text{owl:Nothing}, C\rangle \in \mathcal{H}$ (i.e., all 'leaves' of $\mathcal{H}$). Then, for each such C, the algorithm performs a satisfiability test; if C is satisfiable, then the constructed pre-model can be used to determine new known and possible subsumers as illustrated in lines 11–15. Note, however, that C is tested for satisfiability only if $P[C] = \emptyset$ (line 5), which avoids the test if a pre-model for C has been generated previously. The pre-model for C is used to update $K[C]$: if D was added to $\mathcal{L}(s_0)$ deterministically (which can easily be checked in reasoners that use dependency-directed backtracking), then D is guaranteed to be a subsumer of C [8], so $\langle C, D\rangle$ is added to K. The pre-model for C is also used to update $P[D]$ for *each* class D occurring in (any part of) the pre-model: if $D(s) \in \mathcal{A}$ and no possible subsumer for D is known yet, then $P[D]$ is initialised to $\mathcal{L}(s)$; otherwise, $P[D]$ is restricted to the elements in $\mathcal{L}(s)$. Note that $P[D]$ cannot become empty as it necessarily contains D.

Consider, for example, an ontology $\mathcal{O}$ containing axioms (4)–(7). Initially, structural subsumption initialises K by setting $K[X] = \{X, \text{owl:Thing}\}$ for each $X \in \mathbf{C}_\mathcal{O}$, and $K[\text{owl:Nothing}] = \mathbf{C}_\mathcal{O}$. At this point, ToTest contains C, D, E, F and G. Let us assume that C is chosen first, and a pre-model for $C(s_0)$ is generated. Due to axiom (4), s_0 must be related to an instance of D, say s_1, by property *op*. Since $D \in \mathcal{L}(s_1)$, the pre-model is also a pre-model for D. Due to axiom (6) and the ObjectUnionOf constructor, the reasoner can non-deterministically add E or F to $\mathcal{L}(s_1)$. Let us assume that the reasoner chooses E and then terminates returning $\mathcal{A}$; this pre-model can be used to infer that $P[C] = \{C\}$ and $P[E] = P[D] = \{D, E\}$. In the next iteration, D is chosen from the list, but $P[D] \neq \emptyset$ (information for D is already known), so no test is performed for D. At some point G is chosen and a model for $G(s_0)$ is constructed. Due to axiom (7), the reasoner relates s_0 with some fresh s_1 by property *op*2 such that $D \in \mathcal{L}(s_1)$. Let us assume, however, that to satisfy axiom (6), the reasoner now adds F to $\mathcal{L}(s_1)$. Since $P[D] \neq \emptyset$, $\mathcal{L}(s_1)$ can be used to prune $P[D]$; more precisely, since $E \notin \mathcal{L}(s_1)$, E is removed from $P[D]$.

$$\text{SubClassOf}(D\ \text{ObjectUnionOf}(E\ F)) \tag{6}$$

$$\text{SubClassOf}(G\ \text{ObjectSomeValuesFrom}(op2\ D)) \tag{7}$$

Note that neither K nor P are updated if C is unsatisfiable, so little information is obtained from a satisfiability test for C. Hence, if $\mathcal{O}$ contains many unsatisfiable classes, initialisation might not provide enough initial information for K and P. Consequently, whenever our algorithm finds an unsatisfiable class C, it traverses $\mathcal{H}$ "upwards" until it finds a satisfiable class; furthermore, the unsatisfiability is propagated to all descendants of C in $\mathcal{H}$ (lines 7-10). Apart from making initialisation more robust, such an approach potentially identifies unsatisfiable classes without performing actual satisfiability tests (e.g., if D is

discovered to be unsatisfiable and $\mathcal{O}$ contains $C \sqsubseteq D$). An example of such an ontology is FMA [2], which can be classified using our algorithm much more efficiently than with ET (see Section 6).

The Classification Phase It is possible that all subsumers of a class D are identified after the initialisation phase, and this can happen even if the satisfiability of D had not been tested explicitly (in line 6). In our running example, all possible subsumers of D are already known (since $P[D] \subseteq K[D]$). For memory as well as for performance reasons, our algorithm next identifies only those classes for which there are unknown possible subsumers (lines 16-17), and operates only on them.

For these classes our algorithm proceeds as follows. It iteratively chooses a class C with $P[C] \neq \emptyset$ and checks $C \sqsubseteq D$ for each $D \in P[C]$. In order to perform these checks as efficiently as possible, the algorithm does not test each subsumption separately. Instead, inspired by the *clustering* optimisation [3], our algorithm tries to build a model for $C \sqcap \neg F$, where F is the conjunction of all possible subsumers of C (line 20). If a model exists, then $C \not\sqsubseteq F$ and so all concepts in $P[C]$ are non-subsumers of C.

If a model for $C \sqcap \neg F$ does not exist, then at least one concept in $P[C]$ is a subsumer of C, so a more detailed check is needed. The algorithm then proceeds as follows. It computes a transitively-reduced strict partial order $\mathcal{H}$ of the subsumers 'induced' by C. The standard ET algorithm is then applied to C over $\mathcal{H}$ in order to identify the (non-)subsumers of C. In contrast to KP, our algorithm introduces the following optimisation: if $C \sqcap \neg D$ is satisfiable for D a possible unknown subsumer of C (i.e., if $\mathcal{O} \not\models C \sqsubseteq D$), then the constructed pre-model can again be used to prune non-subsumers as was done in the initialisation phase. This process is performed in place of Algorithm 1, as it provides a more efficient pruning strategy. Another interesting and useful consequence of interleaving pruning with subsumption checking is that it can lead to the pruning of other possible subsumers of C that might otherwise be tested in a subsequent iteration. Therefore, the algorithm checks whether D is still a possible subsumer of C (line 28) before trying to construct a pre-model for $C \sqcap \neg D$ (line 29).

After the classification phase, all unknown possible subsumers will have been tested, and K contains all subsumption relations, so it is used to construct the final class hierarchy.

3.1 Further Comparisons with the KP Algorithm

We have already illustrated the major differences between Algorithm 2 and KP, such as the initialisation of K and P, and our new technique for pruning relations from P. In the following, we point out some additional differences, and we discuss further the pruning technique.

- **Memory Efficiency:** Our algorithm uses memory much more efficiently than KP. Recall that KP transitively closes K, which is not a good strategy on large ontologies such as FMA or SNOMED that contain thousands of

classes. Furthermore, KP assumes that $P \supseteq K$—that is, all known subsumptions (including those derived by the transitive closure) are contained in P. In contrast, our algorithm uses a graph reachability algorithm to identify whether $\langle C, D \rangle$ belongs to the transitive closure of K, and removes the information about the classified classes from P, both of which can significantly reduce the algorithm's memory footprint.

- **Pruning:** Although our classification algorithm does not directly use Algorithm 1, it indirectly implements parts of Algorithm 1. For example, if $B \in P[A]$, but tests show that $\mathcal{O} \not\models A \sqsubseteq B$, then B can also be inferred to be a non-subsumer of all the subsumers of A as in the first loop of Algorithm 1. The second loop of Algorithm 1 prunes possible subsumptions when new positive subsumptions are inferred. However, our experience has shown that this strategy rarely identifies new non-subsumptions in practice. Consequently, the cost of applying such an expensive algorithm rarely outweighs the cost of performing a couple of additional subsumption tests.
- **Bottom-up Phase:** As in the ET algorithm, KP includes a bottom-up phase where the subsumees of an unclassified class C are identified in order to correctly place C into the class hierarchy. Our algorithm, however, does not include a bottom-up phase, which considerably simplifies the implementation as one does not need doubly-linked data structures for efficient retrieval of both successors and predecessors of C in K and P. Note that our algorithm is still complete since, if C is a possible but not yet known child of D, then $C \in P[D]$ and the relevant subsumption is tested when D is selected.

4 Object Property Classification

Classification of properties has, to the best of our knowledge, not been discussed in the literature. Apart from HermiT, all ontology reasoners that we are aware of construct the property hierarchy simply by computing the reflexive-transitive closure of the asserted property hierarchy. Such an algorithm is cheap to implement and requires no complex reasoning; however, it is incorrect for OWL as well as for considerably weaker ontology languages. Consider, for example, an ontology containing the following axioms:

$$\text{SubClassOf}(\ \text{ObjectSomeValuesFrom}(op_1\ \textit{owl:Thing})\ \text{ObjectSomeValuesFrom}(op_2\ \textit{owl:Thing})\ ) \tag{8}$$

$$\text{SubObjectPropertyOf}(op_1\ op_3) \tag{9}$$

$$\text{SubObjectPropertyOf}(op_2\ op_3) \tag{10}$$

$$\text{FunctionalObjectProperty}(op_3) \tag{11}$$

These axioms entail $op_1 \sqsubseteq op_2$: given $op_1(i_1, i_2)$, axiom (8) requires the existence of an op_2-successor for i_1; since both op_1 and op_2 are subproperties of op_3 and op_3 is functional, then i_2 must also be the op_2-successor for i_1, so we have $op_2(i_1, i_2)$.

Property chains and nominals can also imply implicit property subsumptions. The problems with property chains are demonstrated by the following example.

$$\text{SubClassOf}(\textit{owl:Thing}\ \text{ObjectSomeValuesFrom}(\textit{op}\ \textit{owl:Thing})) \tag{12}$$

$$\text{SubObjectPropertyOf}(\text{ObjectPropertyChain}(\textit{op}_1\ \textit{op}\ \text{ObjectInverseOf}(\textit{op}))\ \textit{op}_2) \tag{13}$$

Whenever i_1 has an op_1-successor i_2, axiom (12) ensures that i_2 has an op-successor i_3; hence, we have $op_1(i_1, i_2)$, $op(i_2, i_3)$ and $\text{ObjectInverseOf}(op)(i_3, i_2)$, and from axiom (13) we can infer $op_2(i_1, i_2)$, so the ontology implies $op_1 \sqsubseteq op_2$. Property classification in HermiT was initially based on the ET algorithm. Similarly to class subsumption testing, we concluded that $\mathcal{O} \models op_1 \sqsubseteq op_2$, for op_1 and op_2 object properties, iff $\mathcal{O} \cup \{op_1(a, b), \neg op_2(a, b)\}$ is not satisfiable, where a, b were individuals not occurring in $\mathcal{O}$. However, this is correct only for simple properties [7], where simple properties are roughly those that do not occur in property chains and transitivity axioms.

The problem with complex properties (i.e., non-simple ones) is that complex property assertions are not necessarily made explicit in the constructed pre-models. To ensure decidability, property chains and transitivity axioms are typically encoded into subclass axioms that propagate classes along paths in the pre-model in a way such that adding all missing property relationships does not violate any ontology axiom. Roughly speaking, given the property axiom $\text{SubObjectPropertyOf}(\text{ObjectPropertyChain}(op\ op)\ op)$ (which states that op is transitive), each axiom containing a universal quantifier over op is rewritten in a particular way; for example, axiom (14) is replaced with axioms (15)–(17)

$$\text{SubClassOf}(C\ \text{ObjectAllValuesFrom}(\textit{op}\ D)) \tag{14}$$

$$\text{SubClassOf}(C\ \text{ObjectAllValuesFrom}(\textit{op}\ D_{op})) \tag{15}$$

$$\text{SubClassOf}(D_{op}\ D) \tag{16}$$

$$\text{SubClassOf}(D_{op}\ \text{ObjectAllValuesFrom}(\textit{op}\ D_{op})) \tag{17}$$

where D_{op} is a fresh class. In order to compute all axioms required to eliminate all property inclusions, a non-deterministic finite automaton is constructed for each complex property, and subclass axioms are then extracted from automaton's transitions [4]. In order for the elimination to work as desired, negative property assertions with complex properties must be rewritten. For example, assertion (18) must be rewritten as (19)

$$\text{NegativeDataPropertyAssertion}(\textit{op}\ a\ b) \tag{18}$$

$$\text{ClassAssertion}(\text{ObjectAllValuesFrom}(\textit{op}\ \text{ObjectComplementOf}(\text{ObjectOneOf}(b)))\ a) \tag{19}$$

where (19) states that a belongs to the class of individuals for which all op-successors are not b. The universal quantifier then triggers the generation of further axioms in the property chain elimination as described above.

Since complex property assertions are not necessarily made explicit in the pre-models, we cannot read off non-subsumptions from pre-models; that is, when $op_1(a, b)$ occurs but $op_2(a, b)$ does not occur in a pre-model, we cannot conclude $op_1 \not\sqsubseteq op_2$ if op_2 is a complex property. This significantly reduces the opportunities for pruning the search space, which makes property classification harder than standard (class) classification. We point out that, in the case described above, the publicly available 1.2.2 version of HermiT incorrectly concludes $op_1 \not\sqsubseteq op_2$. We corrected this error in the version of HermiT used for evaluation (see Section 6), which significantly decreased the performance of property classification.

In order to address these issues, we developed a new property classification technique that reduces property classification to standard (class) classification. Any classification algorithm, such as the one described in Section 3, can then be used to classify the property hierarchy, and it can use all relevant optimisations for pruning the search space. The reduction is defined as follows.

Definition 2. *Let $\mathcal{O}$ be an OWL 2 ontology and let* **OPE** *be the object properties and inverse object properties occurring in $\mathcal{O}$. An* object property to class mapping w.r.t. $\mathcal{O}$ *is a total and injective function τ from* **OPE** *to classes not occurring in $\mathcal{O}$. Let C_f be a class occurring neither in $\mathcal{O}$ nor in the range of τ. The* object property hierarchy induced by τ w.r.t. $\mathcal{O}$*, written $\mathcal{H}^\tau_\mathcal{O}$, is the transitive reduction of the relation $\{\langle op_1, op_2\rangle \mid op_1, op_2 \in \mathbf{OPE}$ and $\mathcal{O}_\tau \models \tau(op_1) \sqsubseteq \tau(op_2)\}$, where $\mathcal{O}_\tau$ is an extension of $\mathcal{O}$ with axioms of the following form for each object property $op \in \mathbf{OPE}$.*

$$\mathit{EquivalentClasses}(\tau(op)\ \mathit{ObjectSomeValuesFrom}(op\ C_f))$$

We write $(\mathcal{H}^\tau_\mathcal{O})^*$ to denote the reflexive-transitive closure of $\mathcal{H}^\tau_\mathcal{O}$.

Intuitively, to test $op_1 \sqsubseteq^? op_2$, we test $C_1 \sqsubseteq^? C_2$, where C_1 and C_2 are the representative classes introduced by τ for op_1 and op_2, respectively. As in standard classification, the reasoner checks this subsumption by trying to construct a pre-model containing $C_1(i)$ and $\neg C_2(i)$ for some individual i. The axioms in $\mathcal{O}_\tau$ then cause the addition of an op_1-successor of i, say i', with $C_f(i')$. If, due to other axioms in $\mathcal{O}$, i' is necessarily an op_2-successor of i as well, then the corresponding axiom in $\mathcal{O}_\tau$ for op_2 causes the addition of $C_2(i)$, which leads to a clash, which confirms the subsumption. Complex properties are handled using the transformation described earlier, so reading off non-subsumptions and pruning the set of possible subsumers works exactly as for classes.

The following theorem shows that this reduction of the object property classification problem to a standard classification problem is indeed correct.[1]

Theorem 1. *Let $\mathcal{O}$ be an OWL 2 ontology with $op_1, op_2 \in \mathbf{OPE}$, let τ be an object property to class mapping w.r.t. $\mathcal{O}$, and let $\mathcal{H}^\tau_\mathcal{O}$ be the object property hierarchy induced by τ w.r.t. $\mathcal{O}$. Then $\mathcal{O} \models op_1 \sqsubseteq op_2$ iff $\langle op_1, op_2\rangle \in (\mathcal{H}^\tau_\mathcal{O})^*$.*

[1] A complete proof is available in the accompanying technical report at `http://www.hermit-reasoner.com/2010/classification/Classification.pdf`

5 Data Property Classification

Problematic constructors such as property chains do not apply to data properties, so one might think that data properties can be classified by just computing the reflexive-transitive closure of the asserted data property subsumptions. This, however, is not the case since we can easily adjust axioms (8)–(11) to work with data properties and *rdfs:Literal* instead of *owl:Thing*.

Another problem is that data property subsumption tests are difficult to implement. Since data properties are always simple, to test $\mathcal{O} \models dp_1 \sqsubseteq dp_2$ with dp_1 and dp_2 data properties, we might try to check whether $\mathcal{O} \cup \{dp_1(i, n), \neg dp_2(i, n)\}$ is unsatisfiable for i a fresh individual and n a fresh data value. We cannot, however, simply choose n to be any data value that does not occur in the input ontology. Assume, for example, that we selected an integer that does not occur in the input ontology $\mathcal{O}$; there are infinitely many integers, so there is always one not occurring in $\mathcal{O}$. This, however, might lead to conclusions that depend on the chosen integer: unlike for a fresh individual that can be interpreted as an arbitrary element of the object domain, the interpretation of a data value is fixed a priori. This problem can be solved by inventing a dummy datatype D that is considered to be non-disjoint with all datatypes in the OWL 2 datatype map (i.e., its value space can be intersected with any other data range without causing a contradiction); the only constraint for D is that a data value cannot belong to D and its complement. In order to check if $\mathcal{O} \models dp_1 \sqsubseteq dp_2$, the reasoner now checks the satisfiability of $\mathcal{O}$ extended with the following axioms, where i is a fresh individual:

$$\text{ClassAssertion}(\text{DataSomeValuesFrom}(dp_1\ \mathsf{D})\ i) \quad (20)$$

$$\text{ClassAssertion}(\text{DataAllValuesFrom}(dp_2\ \text{DataComplementOf}(\mathsf{D}))\ i) \quad (21)$$

There is, however, still a problem with this approach. Datatype reasoning is typically implemented using a procedure such as the one presented by Motik and Horrocks [5]. If an individual i has a data property successor n, then one must check whether there are only finitely many values that n can take; if that is the case, one must find data values for n and the 'relevant' siblings of n that are related to the same individual i as n. A sibling n' is relevant if it can also have only finitely many possible data values and the assignment must be different from the one for n due to an inequality between n and n' (e.g., the inequality can be introduced by an at-least restriction). Thus, to handle D properly, an inequality must be generated between siblings n and n' if one of them must belong to D while the other must belong to the complement of D, which guarantees that the two nodes are not assigned the same data value in the procedure by Motik and Horrocks. Furthermore, note that even if n and n' must be assigned the same values, n and n' are not merged; for example, if an individual is required to have the integer 1 both as a dp_1- and a dp_2-successor, the two successors will be represented as separate objects in a pre-model. This again prevents the reading off of non-subsumptions between data properties. We should point out that this problem was also overlooked in HermiT 1.2.2, and correcting the error again significantly increased data property classification times.

We can, however, reduce data property classification to standard classification similarly as for object properties. This reduction allows us to read off subsumptions and non-subsumptions between data properties, because such (non-)subsumptions are reflected in the classes introduced by the encoding.

Definition 3. *Let $\mathcal{O}$ be an OWL 2 ontology and let* D *be a dummy datatype as discussed above. A* data property to class mapping w.r.t. $\mathcal{O}$ *is a total and injective function σ from* $\mathbf{DP}$ *to classes not occurring in $\mathcal{O}$. The* data property hierarchy induced by σ w.r.t. $\mathcal{O}$*, written $\mathcal{H}^{\sigma}_{\mathcal{O}}$, is the transitive reduction of the relation $\{\langle dp_1, dp_2\rangle \mid dp_1, dp_2 \in \mathbf{DP}$ and $\mathcal{O}_\sigma \models \sigma(dp_1) \sqsubseteq \sigma(dp_2)\}$, where $\mathcal{O}_\sigma$ is an extension of $\mathcal{O}$ with axioms of the following form for each data property $dp \in \mathbf{DP}$.*

$$\mathit{EquivalentClasses}(\sigma(dp)\ \mathit{DataSomeValuesFrom}(dp\ \mathsf{D}))$$

We write $(\mathcal{H}^{\sigma}_{\mathcal{O}})^*$ to denote the reflexive-transitive closure of $\mathcal{H}^{\sigma}_{\mathcal{O}}$.

The following theorem shows that the reduction is indeed correct. The proof is a straightforward adaptation of the proof of Theorem 1.

Theorem 2. *Let $\mathcal{O}$ be an OWL 2 ontology with $dp_1, dp_2 \in \mathbf{DP}_{\mathcal{O}}$, let σ be a data property to class mapping w.r.t. $\mathcal{O}$, and let $\mathcal{H}^{\sigma}_{\mathcal{O}}$ be the data property hierarchy induced by σ w.r.t. $\mathcal{O}$. Then $\mathcal{O} \models dp_1 \sqsubseteq dp_2$ iff $\langle dp_1, dp_2\rangle \in (\mathcal{H}^{\sigma}_{\mathcal{O}})^*$.*

6 Evaluation

We have implemented Algorithm 2 and the property classification encodings in the HermiT 1.3 (hyper)tableau reasoner. To evaluate the effectiveness of our technique, we compared the performance of HermiT 1.3 against HermiT 1.2.2a (which implements the ET strategy, but with bugs related to property classification corrected as described in Sections 4 and 5). In our tests, we used two versions of the GALEN ontology, several ontologies from the Open Biological Ontologies (OBO) Foundry, the Food and Wine ontology from the OWL Guide, the Foundational Model of Anatomy (FMA), and ontologies from the Gardiner ontology suite. All ontologies and both HermiT versions are available online.[2] Table 1 summarises the numbers of classes and properties in each of the test ontologies.

The tests consisted of classifying the classes and properties of our test ontologies. We measured the classification time (in seconds) as well as the number of actual reasoning tests performed (including both satisfiability and subsumption tests). All experiments were performed on a UNIX machine of an Intel x86 64bit Cluster on one node with two quad core 2.8GHz processors and Java 1.5 allowing 2GB of heap memory. The results are summarised in Table 2. The upper part of the table contains all the deterministic ontologies (that is, the ontologies that do not use disjunctive constructors), while the lower part contains all the non-deterministic ontologies. For ontologies without data properties, we write '-' in Table 2 and OoM stands for Out of Memory.

Table 1. Number of classes and properties in the evaluated ontologies

	classes	object prop.	data prop.		classes	object prop.	data prop.
GALEN-d	2 748	413	0	AEO	760	47	16
GALEN-und	2 748	413	0	substance	1 721	112	33
GO	19 528	1	0	ProPreO	482	30	0
GO_XP	27 883	5	0	OBI	2 638	77	6
chebi	20 979	10	0	Food-Wine	139	17	1
NCI	27 652	70	0	FMA 2.0	41 648	148	20

Table 2. Evaluation results for class and property classification (time in seconds)

Ontology	**Classes**				**Object Properties**				**Data Properties**			
	1.2.2a (ET)		1.3 (KP)		1.2.2a (ET)		1.3 (KP)		1.2.2a (ET)		1.3 (KP)	
	Tests	Time	Tests	Time	Tests	Time	Tests	Time	Tests	Time	Tests	Time
GALEN-d	2 744	3.6	3 380	2.9	6 073	439.2	197	< 1	-	-	-	-
GALEN-und	2 744	28.3	4 009	7.2	6 001	459.5	198	< 1	-	-	-	-
GO	19 260	43.0	14 288	3.7	4	< 1	3	< 1	-	-	-	-
GO_XP	27 880	119	20 029	14.4	9	10.4	6	4.8	-	-	-	-
chebi	20 693	69.8	13 484	7.6	26	59.9	12	18.1	-	-	-	-
NCI	27 652	71.1	21 367	10.5	71	< 1	72	< 1	-	-	-	-
AEO	2285	2.1	364	1.7	214	6.0	34	< 1	223	4.6	28	< 1
substance	4 569	15.9	2 730	12.8	962	23.6	107	< 1	957	22.5	40	< 1
ProPreO	1 441	7.3	1 157	6.8	518	3	33	< 1	-	-	-	-
OBI	12 444	254.7	3 047	170.1	2 278	310.5	52	3.4	39	6.0	7	< 1
Food-Wine	382	18.8	243	11.7	65	11.6	13	2.0	4	< 1	3	< 1
FMA 2.0	49 716	7 973.8	10 980	731.8	8 281	16 668.3	128	8.4	283	469.9	29	< 1

As Table 2 shows, the new classification strategy of HermiT 1.3 is in all cases significantly faster than the ET strategy of HermiT 1.2.2a, sometimes by one or even two orders of a magnitude. This is particularly the case for property classification where, as we have explained in the previous section, none of HermiT's standard optimisations can be applied, and one relies completely on the insertion strategy of ET to reduce the number of subsumption tests. In contrast, our property classification encoding can reuse the standard (class) classification optimisations, thus achieving a very good and robust performance. These results show that it is practically feasible to perform correct property classification through reasoning, instead of the cheap but incomplete transitive closure algorithms. The results for standard classification are similar: the new algorithm has significantly reduced the classification time in most cases. The significant performance gain in the classification of FMA is due in part to the heuristic implemented in lines 7–10 of Algorithm 2, which prevents HermiT from repeatedly performing class satisfiability tests for unsatisfiable classes.

The good performance results are also confirmed by the significant reduction in the number of required reasoning tests. The only case where HermiT 1.3 performs more tests is on GALEN, which is due to the fact that, on deterministic ontologies, HermiT 1.2.2a uses satisfiability tests and the pre-model reading technique [8] which identifies *all* subsumers of the tested class. In contrast, our

[2] `http://www.hermit-reasoner.com/2010/classification/Evaluation.zip`

Table 3. Number of tests performed by HermiT 1.3 compared to KP

	GO$^{\sqcup}$	GALEN$^{\sqcup}$	NCI$^{\sqcup}$
KP	32 614	4 657	48 389
HermiT 1.3	27 250	4 983	41 094

method does not test the satisfiability of each class, so after the first phase there are unknown possible subsumers that need to be checked in the second phase. Especially in GALEN, most of them are subsumers, so the pruning step in lines 30–31 is rarely applicable. Nevertheless, such reasoning tests are usually very fast, so the overall system still performs better than HermiT 1.2.2a. On GALEN-und, where satisfiability tests are expensive, the benefits of not performing a satisfiability test for every class are particularly noticeable.

As a final experiment, we compared the performance of our system with the one that implements the KP algorithm [9]. We tested our system on three specially constructed ontologies that were used in [9] to evaluate the KP algorithm, and we compared the number of tests performed by our method with the number of tests published in [9]; Table 3 summarises the results. We can again see that for all ontologies but GALEN, our system performs fewer tests; furthermore, the same observations as above explain this difference. Unfortunately, the original implementation of KP was not available, so we were unable to compare the performance of HermiT with that of KP on the ontologies from Table 2.

7 Conclusions

In this paper, we considered the problem of efficiently classifying OWL ontologies. Unlike in previous approaches, we consider all classification tasks, including class, object and data property classification. To the best of our knowledge, property classification has not previously been discussed in the literature.

We presented a new classification algorithm that is based on KP [9], but that solves several open problems and that incorporates numerous refinements and optimisations. The latter include, for example, a novel heuristic strategy for initialising relations K and P, an efficient pruning strategy, and a novel heuristic for pruning unsatisfiable classes. Additionally, our new algorithm is more memory efficient than KP.

We presented examples that show why traditionally used algorithms based on the reflexive-transitive closure of the asserted property hierarchy are incomplete for property classification in OWL. We then discussed the difficulties in reusing well-known optimisations in the context of property classification, and we presented a novel reduction of the property classification problem to a standard classification problem. This reduction allows us to reuse all the optimisations applicable to the classification of classes.

Finally, we have implemented all our algorithms and reductions in version 1.3 of the HermiT reasoner, and have compared its performance with earlier versions using the standard classification method. Our results are very encouraging,

showing significant improvements in classification times. Moreover, in the case of properties, our experiments show for the first time that complete property classification can be effectively implemented in practice.

We are currently working on extending our algorithm to handle realisation—the task of computing, for each individual i in an ontology, the most specific classes C such that i is an instance of C—and for realising property instances. Our preliminary results suggest that the performance of realisation can be significantly improved by applying the ideas outlined in this paper.

Acknowledgements The presented work is funded by the EPSRC project HermiT: Reasoning with Large Ontologies. The evaluation has been performed on computers of the Oxford Supercomputing Centre.

References

1. Baader, F., Hollunder, B., Nebel, B., Profitlich, H.J., Franconi, E.: An empirical analysis of optimization techniques for terminological representation systems, or making kris get a move on. In: KR. pp. 270–281 (1992)
2. Golbreich, C., Zhang, S., Bodenreider, O.: The foundational model of anatomy in OWL: Experience and perspectives. Web Semantics 4(3), 181–195 (2006)
3. Haarslev, V., Möller, R.: High performance reasoning with very large knowledge bases: A practical case study. In: IJCAI. pp. 161–168 (2001)
4. Horrocks, I., Kutz, O., Sattler, U.: The even more irresistible $\mathcal{SROIQ}$. In: Proc. KR-06. pp. 57–67 (2006)
5. Motik, B., Horrocks, I.: OWL datatypes: Design and implementation. In: Proc. of the Int. Semantic Web Conf. (ISWC-08). LNCS, vol. 5318, pp. 307–322. Springer (2008)
6. Motik, B., Patel-Schneider, P.F., Cuenca Grau, B.: OWL 2 web ontology language direct semantics. URL (2009), w3C Recommendation, `http://www.w3.org/TR/owl2-direct-semantics/`
7. Motik, B., Patel-Schneider, P.F., Parsia, B.: OWL 2 web ontology language structural specification and functional-style syntax. URL (2009), w3C Recommendation, `http://www.w3.org/TR/owl2-syntax/`
8. Motik, B., Shearer, R., Horrocks, I.: Hypertableau Reasoning for Description Logics. Journal of Artificial Intelligence Research 36, 165–228 (2009)
9. Shearer, R., Horrocks, I.: Exploiting partial information in taxonomy construction. In: Proc. of the Int. Semantic Web Conf. (ISWC-09). vol. 5823, pp. 569–584 (2009)