

The Instance Store: DL Reasoning with Large Numbers of Individuals*

Ian Horrocks, Lei Li, Daniele Turi and Sean Bechhofer
University of Manchester, UK
<lastname>@cs.man.ac.uk

1 Introduction

The Semantic Web [4] aims at making Web resources more accessible to automated processes by adding “semantic annotations”—metadata (data about data) that describes their content. It is envisaged that the semantics in semantic annotations will be given by ontologies, which will provide a source of precisely defined terms (vocabularies) that are amenable to automated reasoning.

A standard for expressing ontologies in the Semantic Web has already emerged: the ontology language OWL [6], which recently became a W3C recommendation. One of the main features of OWL is that there is a direct correspondence between (two of the three “species” of) OWL and Description Logics (*DLs*) [12]. This means that DL reasoners can be used to reason about OWL ontologies and about annotations that are instances of concept descriptions formed using terms from an ontology.

Unfortunately, while existing techniques for *TBox* reasoning (i.e., reasoning about concepts) seem able to cope with real world ontologies [11], it is not clear if existing techniques for *ABox* reasoning (i.e., reasoning about individuals) will be able to cope with realistic sets of instance data. This difficulty arises not so much from the computational complexity of *ABox* reasoning, but from the fact that the number of individuals (e.g., annotations) might be extremely large.

In this paper we describe an approach to *ABox* reasoning that combines a DL reasoner with a database (*DB*). The result, which we call an *Instance Store*, is a system that can deal with very large *ABoxes*, and is able to provide sound and complete answers to instance retrieval queries over such *ABoxes*.

While the Instance Store can be highly effective, it does have limitations when compared to a fully fledged DL *ABox*. Particularly, the Instance Store can only deal with a *role-free* *ABox*, i.e., an *ABox* that does not contain any axioms asserting role

*An extended version of this paper is currently under review in IJCAR'04.

relationships between pairs of individuals. Although this seems a rather severe restriction, the functionality provided by the Instance Store is precisely what is required by many applications, and in particular by applications where ontology based terms are used to describe/annotate and retrieve large numbers of objects. Examples include the use of ontology based vocabulary to describe documents in “publish and subscribe” applications [16], to annotate data in bioinformatics applications [8] and to annotate web resources (web pages) [7] or web service applications [13].

Using a DB in order to support ABox reasoning is certainly not new (see Section 1.1 for the related work), but to the best of our knowledge the Instance Store is the first such system that is general purpose, provides sound and complete reasoning, and places no a-priori restriction on the size of the ABox. In order to evaluate the Instance Store design, and in particular its ability to provide scalable performance for instance retrieval queries, we have performed a number of experiments using the implemented Instance Store with a large (50,000 concept) gene ontology [8] and a very large number (up to 650,000) of individuals that are instances of concept descriptions formed using terms from the ontology. We have also compared the performance of the Instance Store with that of RACER [9] (the only publicly available DL system that supports full ABox reasoning for an expressive DL) and of FaCT [11] (using TBox reasoning to simulate reasoning with a role-free ABox).

1.1 Background

Description Logics are a family of knowledge representation formalisms evolved from early *frame systems* and *semantic networks*. We assume the reader to be familiar with DLs—see [3] for a detailed discussion of DLs.

As already mentioned, the idea of supporting DL style reasoning using DB is not new. One example is [5], which can handle DL inference problems by converting them into a collection of SQL queries. This approach is not limited to role-free ABoxes, but the DL language supported is much less expressive, and the DB schema must be customised according to the given TBox. Another example is the Parka system [2]. Parka is not limited to role-free ABoxes and can deal with very large ABoxes. However, Parka also supports a much less expressive language, and is not based on standard DL semantics, so it is not really comparable to the Instance Store. Finally, [14] describes a “semantic indexing” technique that is very similar to the approach used in the Instance Store except that files and hash tables are used instead of DB tables, and optimisations such as the use of equivalence sets were not considered.

2 Instance Store

An ABox \mathcal{A} is role-free if it contains only axioms of the form $x : C$. We can assume without loss of generality that there is exactly one such axiom for each individual as

$x : C \sqcup \neg C$ holds in all interpretations, and two axioms $x : C$ and $x : D$ are equivalent to a single axiom $x : (C \sqcap D)$. It is well known that for a role-free ABox, instantiation can be reduced to TBox subsumption [10, 15]; i.e., if $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$, and \mathcal{A} is role-free, then $\mathcal{K} \models x : D$ iff $x : C \in \mathcal{A}$ and $\mathcal{T} \models C \sqsubseteq D$. Similarly, if $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ and \mathcal{A} is a role-free ABox, then the instances of a concept D could be retrieved simply by testing for each individual x in \mathcal{A} if $\mathcal{K} \models x : D$.

An alternative approach is to add a new axiom $C_x \sqsubseteq D$ to \mathcal{T} for each axiom $x : D$ in \mathcal{A} , where C_x is a new atomic concept; we will call such concepts *pseudo-individuals*. Classifying the resulting TBox is equivalent to performing a complete realisation of the ABox: the most specific atomic concepts that an individual x is an instance of are the most specific atomic concepts that subsume C_x and that are not themselves pseudo-individuals. Moreover, the instances of a concept D can be retrieved by computing the set of pseudo-individuals that are subsumed by D .

The problem with this latter approach is that the number of pseudo-individuals added to the TBox is equal to the number of individuals in the ABox, and if this number is very large, then TBox reasoning may become inefficient or even break down completely (e.g., due to resource limits). The idea behind the Instance Store is to overcome this problem by using a DL reasoner to classify the TBox and a DB to store the ABox, with the DB also being used store a complete realisation of the ABox, i.e., for each individual x , the concepts that x realises (the most specific atomic concepts that x instantiates). The realisation of each individual is computed using the DL reasoner when an axiom of the form $x : C$ is added to the Instance Store ABox.

A retrieval query to the Instance Store (i.e., computing the set of individuals that instantiate a query concept) can be answered using a combination of DB queries and TBox reasoning. Given an Instance Store containing a KB $\langle \mathcal{T}, \mathcal{A} \rangle$ and a query concept Q , the instances of Q can be computed using the following steps:

1. use the reasoner to compute \mathbf{C} , the set of most specific atomic concepts in \mathcal{T} that subsume Q , and \mathbf{D} , the set of all atomic concepts in \mathcal{T} that are subsumed by Q ;
2. use the DB to compute A_Q , the set of individuals in \mathcal{A} that realise *some* concept in \mathbf{D} , and A_C , the set of individuals in \mathcal{A} that realise *every* concept in \mathbf{C} ;
3. use the DL reasoner to compute A'_Q , the set of individuals $x \in A_C$ such that $x : B$ is an axiom in \mathcal{A} and B is subsumed by Q ;
4. return the answer $A_Q \cup A'_Q$.

Note that if Q is equivalent to an atomic concept X , then $\{X\} \subseteq \mathbf{C} \subseteq \mathbf{D}$, and the answer A_Q can be returned without computing A'_Q .

2.1 An Optimised Instance Store

In practice, several refinements to the above procedure are used to improve the performance of the Instance Store. In the first place, as it is potentially costly, we should try to minimise the DL reasoning required in order to compute realisations and to check if individuals in A_C are instances of the query concept. One way to (possibly)

reduce the need for DL reasoning is to avoid repeating computations for “equivalent” individuals by checking for syntactic equality using a DB lookup.

Similarly, we can avoid repeated computations of sub and super-concepts for the same concept by caching the results of such computations in the DB.

Finally, the number and complexity of DB queries also has a significant impact on the performance of the Instance Store. One way to reduce DB queries is to store not only the most specific concepts instantiated by each individual, but to store *every* concept instantiated by each individual. It greatly simplifies the computation of A_Q : it is only necessary to compute the (normally) much smaller set D' of most general concepts subsumed by Q , and to query the DB for individuals that instantiate some member of D' . On the other hand, the computation of A_C is slightly more complicated as A_Q must be subtracted from the set of individuals that instantiate every concept in C . Empirically, however, the saving when computing A_Q seems to far outweigh the extra cost of computing A_C .

2.2 Implementation

We have implemented the Instance Store using a component based architecture that is able to exploit existing DL reasoners and DBs. For detailed information please visit [1].

In the current implementation, we make the simplifying assumption that the TBox itself does not change. Extending the implementation to deal with monotonic extensions of the TBox would be relatively straightforward, but deleting information from the TBox might require (in the worst case) all realisations to be recomputed.

3 Empirical Evaluation

To test the scalability and performance of the Instance Store we have performed a number of tests using a large TBox and a very large ABox. For comparison, we carried out the same tests using the RACER system, and also tested the pseudo-individual approach (discussed in Section 2) using the FaCT system. In our tests we use a TBox derived from an (enriched) version of the Gene Ontology (GO), more information regarding the tests can be found on the Instance Store website.

The retrieval performance tests use two sets of queries. The first set was formulated with the help of domain experts and consists of five realistic queries (Q1-Q5) that might reasonably be posed by a biologist. The second set consists of six artificial queries (Q6-Q11) designed to test the effect on query answering performance of factors such as the number of individuals in the answer; whether the query concept is equivalent to an atomic (named) concept (if so, then the answer can be returned without computing A'_Q); and the number of candidate concepts in A_C for which DL reasoning is required in order to determine if they form part of the answer.

3.1 Loading and Querying Tests

In these tests, we compared the performance of the Instance Store with that of RACER using the GO TBox and different sized subsets of the GO ABox. The Instance Store was first initialised with the GO TBox, then for each ABox, we measured the time (in CPU seconds) taken to load the ABox into the Instance Store and to answer each of the queries.

For RACER, we carried out the same tests in two different ways. In both cases we first initialised RACER with the GO TBox, then loaded the ABox. In the first test, we used the *realize-abox* function to force RACER to compute a complete realisation of the ABox before answering any queries, if the realisation was successfully completed, we then timed how long it took to answer each of the queries. In the second test, we simply timed how long it took RACER to answer each of the queries without first forcing it to realise the ABox.

Our test results show that the time take by the Instance Store to load the ABoxes increases more slowly than their size: the time taken per distinct description increases from about 1s per description for the size 200 ABox (which contains 155 distinct descriptions) to approximately 3s per description for the size 653,762 ABox (which contains 48,581 distinct descriptions).

The time taken by RACER to realise the smallest ABox is roughly the same as that taken by the Instance Store. As the ABox size grows, however, the time taken increases rapidly, and at ABox size 1,000 it is already taking approximately 22s per axiom. For larger ABoxes, RACER broke down due to a resource allocation error in the underlying Lisp system.

The results for the Instance Store when answering each of the five realistic queries and six artificial queries are plotted against the size of the ABox in Figure 1; note the logarithmic scales on both axes.

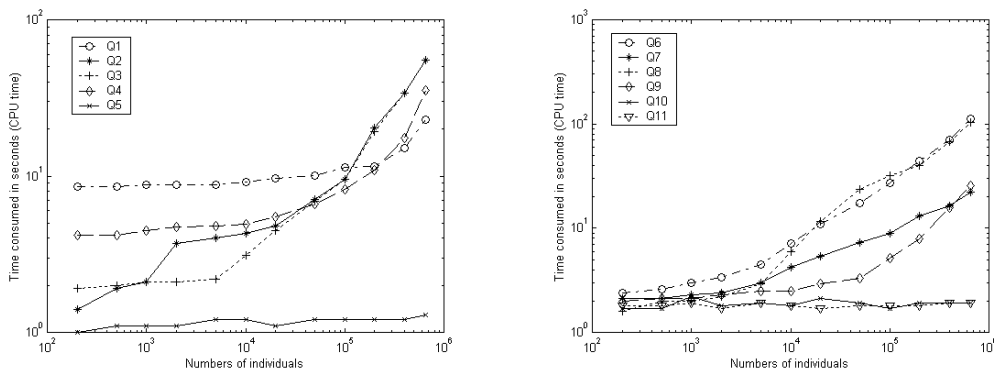


Figure 1: Instance Store realistic (left) and artificial (right) query times -v- ABox size

As can be seen, the time taken to answer queries like Q6 and Q8 becomes quite

large. In these cases, since the number of individuals in A_C is large the time taken to check if these individuals (roughly 0.2s per individual) dominates other factors. The number of “distinct” individuals in the answer also has a significant impact on performance: when there are many such individuals, the DB query required in order to compute the complete answer set can be quite time consuming.

The results for RACER when answering the same sets of queries are also taken, both in the case where the ABox has been realised and where it has not. Timings are only approximate, as precise measurements were not possible when using RACER under Windows. When the ABox had been realised, queries were answered almost *instantly*, but results are only available for the relatively small ABoxes that RACER was able to realise (up to 1,000 individuals). When the ABox was not realised, answers were again returned almost *instantly* for smaller ABoxes, but when the ABox size exceeded 1,000 individuals the answer times increased dramatically, and for ABoxes larger than 10,000 individuals (larger than 5,000 in the case of Q9) RACER again broke down due to a resource allocation error in the underlying Lisp system.

It should be mentioned that the results for the Instance Store include significant communication overheads (both with the DB and DL reasoner), which was not the case for RACER since queries were posed directly via its command line interface.

3.2 Pseudo-individual Tests

As discussed in Section 2, one way to deal with role-free ABoxes is to treat individuals as atomic concepts in the TBox (pseudo-individuals). To test the feasibility of this approach, we again used the GO TBox and ABox, and the set of queries described above. To make the comparison fair, only the distinct instantiated concept expressions are used. The FaCT system was used in these tests as RACER broke down when trying to classify the GO TBox augmented with the pseudo-individuals, again due to a resource allocation error in the underlying Lisp system.

In order to investigate how the pseudo-individual approach would scale with increasing ABox (and hence TBox) size, we tried computing the concepts subsumed by each query with the GO TBox alone (which contains 47,012 concept names) and with the TBox augmented with the pseudo-individuals derived from the GO ABox (a total of 95,593 concept names). The results of these tests are given in Table 1. It is important to note that they do not include the time required to expand answers to include sets of equivalent individuals—as discussed above, this can be quite time consuming for some queries (e.g., 19.5s in the case of Q9 with the largest ABox).

As one can see, the time taken to compute the answers to the queries is heavily dependent on the size of the answers, and in the case of Q4 with the pseudo-individual augmented TBox, the time was over 600s. This is in contrast to the Instance Store, where the size of answer had comparatively little effect on the time taken to answer queries. For queries with relatively small answers, however, the pseudo-individual approach was highly effective, even for queries that were time consuming to answer

Table 1: Pseudo-individual query time (CPU seconds) and answer size

Query	GO TBox		GO TBox + ABox	
	Time	Answer Size	Time	Answer Size
Q1	8.1	220	233.3	2,861
Q2	1.3	1	1.2	1
Q3	0.2	1	1.4	4
Q4	26.0	881	631.8	8,609
Q5	0.5	2	5.2	27
Q6	4.3	86	176.6	2,450
Q7	1.4	1	10.0	147
Q8	1.3	1	1.5	7
Q9	1.4	1	3.5	22
Q10	4.2	109	114.4	1,407
Q11	0.5	1	2.0	2

using the Instance Store.

4 Discussion and Future Work

Our experiments show that the Instance Store provides stable and effective reasoning for role-free ABoxes, even those containing very large numbers of individuals. In contrast, full ABox reasoning using the RACER system exhibited accelerating performance degradation with increasing ABox size, and at least the current RACER release was not able to deal with the larger ABoxes used in our evaluation. The pseudo-individual approach to role-free ABox reasoning was more promising, and may be worth further investigation.

The acceptability of the Instance Store’s performance would obviously depend on the nature of the application and the characteristics of the KB and of typical queries. It is likely that the performance of the Instance Store can be substantially improved simply by dealing with constant factors such as communication overheads.

Future work includes the investigation of additional optimisations and enhancements. For instance, providing a more sophisticated query interface. We are also investigating ways to extend the Instance Store approach to ABoxes that are not completely role-free. This may be possible in restricted cases by applying some form of *precompletion* [10] to the ABox.

Acknowledgements.

Thanks to Phil Lord for help with the implementation and to Chris Wroe for help with the GO ontology and the formulation of realistic queries.

References

- [1] Instance Store website. <http://instancestore.man.ac.uk/>.
- [2] W. A. Andersen et al. Parka: Support for extremely large knowledge bases. In G. Ellis et al. editors, *Knowledge Retrieval, Use and Storage for Efficiency: Proc. of the 1st Int. KRUSE Symposium*, 1995.
- [3] F. Baader et al. editors. *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press, 2003.
- [4] T. Berners-Lee. *Weaving the Web*. Harpur, San Francisco, 1999.
- [5] A. Borgida and R. J. Brachman. Loading data into description reasoners. In *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, 1993.
- [6] M. Dean et al. OWL web ontology language 1.0 reference, July 2002. Available at <http://www.w3.org/TR/owl-ref/>.
- [7] S. Dill et al. Semtag and seeker: Bootstrapping the semantic web via automated semantic annotation. In *Proc. of WWW 2003*, 2003.
- [8] GO project. European Bioinformatics Institute. <http://www.ebi.ac.uk/go>.
- [9] V. Haarslev and R. Möller. RACER system description. In *Proc. of IJCAR 2001*, volume 2083 of *Lecture Notes in Artificial Intelligence*, Springer, 2001.
- [10] B. Hollunder. Consistency checking reduced to satisfiability of concepts in terminological systems. *Ann. of Mathematics and AI*, 18(2–4):133–157, 1996.
- [11] I. Horrocks. Using an expressive description logic: FaCT or fiction? In *Proc. of KR'98*, pages 636–647, 1998.
- [12] I. Horrocks and P. F. Patel-Schneider. Reducing OWL entailment to description logic satisfiability. In *Proc. of ISWC 2003*, 2003.
- [13] L. Li and I. Horrocks. A software framework for matchmaking based on semantic web technology. In *Proc. of WWW 2003*, pages 331–339. ACM, 2003.
- [14] A. Schmiedel. Semantic indexing based on description logics. In F. Baader et al. editors, *Proceedings of the KI'94 Workshop KRDB'94*, Sept. 1994.
- [15] S. Tessaris. *Questions and Answers: Reasoning and Querying in Description Logic*. PhD thesis, Univ. of Manchester, Dept. of Computer Science, Apr. 2001.
- [16] M. Uschold et al. A semantic infosphere. In D. Fensel et al. editors, *Proc. of ISWC 2003*, number 2870 in *Lecture Notes in Computer Science*, pages 882–896. Springer, 2003.