Hydrowl: A Hybrid Query Answering System for OWL 2 DL Ontologies

Giorgos Stoilos

School of Electrical and Computer Engineering
National Technical University of Athens, Greece

Abstract. This system description paper introduces the OWL 2 query answering system Hydrowl. Hydrowl is based on novel *hybrid* techniques which in order to compute the query answers combine at run-time a reasoner ans_1 supporting a (tractable) fragment of OWL 2 (e.g., OWL 2 QL and OWL 2 RL) with a fully-fledged OWL 2 DL reasoner ans_2. The motivation is that if most of the (query answering) work is delegated to the (usually) very scalable system ans_1 while the interaction with ans_2 is kept to a bare minimum, then we can possibly provide with scalable query answering even over expressive fragments of OWL 2 DL. We discuss the system's architecture and we present an overview of the techniques used. Finally, we present some first encouraging experimental results.

1 Introduction

Conjunctive query (CQ) answering over ontological knowledge expressed in the OWL 2 DL language has attracted the interest of many researchers as well as application developers the last decade [1, 2]. Unfortunately, query answering over OWL 2 DL ontologies is of very high computational complexity [3, 4] and even after modern optimisations and intense implementation efforts [5] OWL 2 DL systems are still not able to cope with datasets containing billions of data.

The need for efficient query answering has motivated the development of several fragments of OWL 2 DL [6], like OWL 2 EL, OWL 2 QL, and OWL 2 RL for which query answering can be implemented (at-most) in polynomial time with respect to the size of the data. Consequently, for many of these languages there already exist highly scalable systems which have been applied successfully to industrial-strength applications, like OWLim [1] and Oracle's RDF Semantic Graph [7]. The attractive properties of these systems have led application developers to use them even in cases where the input ontology is expressed in the far more expressive OWL 2 DL language. Clearly, in such cases these systems would most likely be *incomplete*—that is, for some user query and dataset they will fail to compute all certain answers. However, techniques that attempt to deliver complete query answering even when using scalable systems that are not complete for OWL 2 DL have also been proposed [8, 9].

In this system description paper we present the architecture and main characteristics of the Hydrowl[1] query answering system. Hydrowl supports expressive

[1] `http://www.image.ece.ntua.gr/~gstoil/hydrowl/`

ontology languages and like the latter mentioned approaches it is based on novel techniques which attempt to use scalable but possibly incomplete systems as much as possible in order to achieve favourable performance. All inferences involving "problematic" constructors of OWL 2 DL that these systems do not support (e.g., existential restrictions and disjunctions) are either explicated ("materialised") at a pre-processing step as additional axioms or are restricted as much as possible during on-line query evaluation. Our first experimental evaluation using Hydrowl has provided with many encouraging results several of which we report in Section 4.

We have so far tried to keep the design of Hydrowl as modular as possible. Hence, we feel that one of its interesting features is that any application developer can plug his/her system into Hydrowl with minimum implementation effort and, moreover, many different combinations of systems are possible. This is also supported by the fact that we have already provided implementations in Hydrowl that combine the OWL 2 RL reasoner OWLim [1] with HermiT [10], with HermiT-BGP [5], and with Rapid [11].

2 Techniques Used in Hydrowl

In the current section we briefly outline the techniques used in Hydrowl.

As mentioned above, for a given OWL 2 DL ontology $\mathcal{T}$, dataset $\mathcal{A}$, and query $\mathcal{Q}$, Hydrowl still tries to use as much as possible an incomplete but scalable system ans to compute the certain answers of $\mathcal{Q}$ over $\mathcal{T} \cup \mathcal{A}$. Clearly, in that case, there can be entailments of $\mathcal{T} \cup \mathcal{A}$ related to $\mathcal{Q}$ that ans will miss. To recover such missing inferences Hydrowl follows two different but not necessarily incompatible approaches.

In the first approach, inferences involving unsupported constructors are explicated ("materialised") in a form that ans can eventually "recognise". For example, let ans be an OWL 2 RL system and let $\mathcal{T} = \{A \sqsubseteq \exists R, \exists R \sqsubseteq B\}$. Since ans is an OWL 2 RL system it cannot handle axioms with existential restrictions in the right hand side. Hence, Hydrowl will compute for ans a new set of axioms $\mathcal{R}$ that will contain the axiom $A \sqsubseteq B$, i.e., it will materialise the entailment $\mathcal{T} \models A \sqsubseteq B$. It can be verified that when ans is applied over $\mathcal{T} \cup \mathcal{R}$ it is able to return all answers to *every* ground query and *every* datasets over $\mathcal{T}$—that is, for every ground CQ $\mathcal{Q}$ and $\mathcal{A}$ we have $\mathsf{cert}(\mathcal{Q}, \mathcal{T} \cup \mathcal{A}) \subseteq \mathsf{ans}(\mathcal{Q}, \mathcal{T} \cup \mathcal{R} \cup \mathcal{A})$.[2] Such a set of axioms $\mathcal{R}$ is called the *repair* of $\mathcal{T}$ for ans [8, 12] and the process of computing it *repairing*.

However, repairing captures only ground entailments hence even after repairing ans is still incomplete for queries containing existential variables (which in the following we call non-SPARQL queries). To also support such queries we need to further materialise the non-ground inferences that are related to the existential variables of the query. Hydrowl accomplishes this by combining ans with a second system ans′ which can explicate such type of information. One such

[2] Note that this allows ans to be unsound, i.e., return wrong answers. However, to the best of our knowledge, the vast majority of OWL 2 RL systems are sound.

family of systems are query rewriting systems which take as input a (possibly non-SPARQL) query $\mathcal{Q}$ and a TBox $\mathcal{T}$ and compute a so-called query rewriting Rew [13–15]. Roughly speaking, a rewriting Rew for $\mathcal{Q}, \mathcal{T}$ consists of two parts, a set of datalog rules Rew_D which captures ground entailments of $\mathcal{T}$ and a union of conjunctive queries Rew_Q which captures all inferences related to non-ground entailments. Consequently, for $\mathsf{Rew}_Q \uplus \mathsf{Rew}_D$ a rewriting for a non-SPARQL CQ $\mathcal{Q}$ over a TBox $\mathcal{T}$ we have that $\mathsf{cert}(\mathcal{Q}, \mathcal{T} \cup \mathcal{A}) \subseteq \mathsf{ans}(\mathsf{Rew}_Q, \mathcal{T} \cup \mathcal{R} \cup \mathcal{A})$ for every dataset $\mathcal{A}$. Summarising, this approach of Hydrowl to query answering follows the following three steps:

1. Compute a repair $\mathcal{R}$ of $\mathcal{T}$ for ans.
2. Load the dataset $\mathcal{A}$, the input TBox $\mathcal{T}$, and the repair $\mathcal{R}$ to ans.
3. For a CQ $\mathcal{Q}$, if $\mathcal{Q}$ is SPARQL then directly evaluate it over ans; otherwise compute a rewriting $\mathsf{Rew}_D \uplus \mathsf{Rew}_Q$ for $\mathcal{Q}, \mathcal{T}$ and evaluate Rew_Q over ans.

Note that steps 1 and 2 are usually required to be performed only once as a pre-processing (changes in $\mathcal{A}$ can be handled incrementally). Moreover, note that the important component both in computing $\mathcal{R}$ as well as in computing Rew_Q is a query rewriting system [8, 12]. Hence, we call this approach rewriting-based query answering.

Interestingly, by recent theoretical results [16–18], it follows that for systems complete for OWL 2 RL repairs always exists for ontologies expressed in Horn-$\mathcal{SHIQ}$ (a fairly expressive fragment of OWL 2) and they might also exist even for arbitrary OWL 2 DL ontologies. Unfortunately, there can be ontologies where a repair does not exist. The second approach followed by Hydrowl can work even if no repair has been pre-computed at all (although some best effort "partial" repair can be assumed). Instead, for an ontology $\mathcal{T}$ and a system ans, Hydrowl first constructs the set $\mathcal{U}$ of atomic (concept and role) queries over $\mathcal{T}$ for which ans is complete, called *query base* of ans for $\mathcal{T}$. Then, for an arbitrary query $\mathcal{Q}$, the query base can be used to efficiently determine if ans is complete for $\mathcal{Q}$ (and any dataset $\mathcal{A}$) or not [19]; in the former case, Hydrowl uses ans to evaluate $\mathcal{Q}$, while in the latter it resorts to a fully-fledged OWL 2 DL reasoner ans'. There are three benefits here. First, it is trivial to see that in theory query bases always exist.[3] Second, ans is expected to be mostly complete ($\mathcal{T}$ usually includes few "problematic" constructors) leaving few queries that need to be evaluated using the fully-fledged OWL 2 DL system. Third, it has been shown [19] that even in cases that Hydrowl needs to use ans', the scalable system can still be exploited in order to speed up the evaluation by ans'. We call this approach of Hydrowl hybrid query answering and is summarised by the following three steps:

1. Load the dataset $\mathcal{A}$ and the input TBox $\mathcal{T}$ to ans.
2. Compute a query base $\mathcal{U}$ of ans for $\mathcal{T}$.
3. For a (ground) CQ $\mathcal{Q}$, if it can be determined using $\mathcal{U}$ that ans is complete for $\mathcal{Q}$ then directly evaluate $\mathcal{Q}$ using ans; otherwise evaluate $\mathcal{Q}$ using a fully-fledged OWL 2 DL reasoner ans' together with ans.

[3] Note, however, that there are limitation in automatically extracting them.

Note that indeed the two previous approaches are not incompatible. For example, before computing a query base one can pre-compute some partial repair $\mathcal{R}$ and also load it to ans. Then clearly, the more the partial $\mathcal{R}$ approximates *the* (full) repair the smaller the query base of ans for $\mathcal{T}$ is. Moreover, a query rewriting system can also be used in Step 3 of the hybrid query answering approach in order to capture some non-ground entailments and hence provide some support for non-SPARQL CQs, however, this has not been forulated and implemented yet.

3 Architecture of Hydrowl

Hydrowl is implemented in JAVA and is available under the AGPL license. Figure 1 presents its main components, where JAVA classes are marked as rectangles, procedures by ovals and data-flow (TBox) with light-blue arrows. The two approaches outlined in the previous section are implemented by the two main classes HybridEvaluator and RewritingBasedEvaluator. As illustrated, to implement these approaches the methods use internally an incomplete and a complete reasoner and this communication is performed through interfaces, namely IncompleteReasoner and CompleteReasoner in the case of HybridEvaluator and IncompleteReasoner and QueryRewritingSystem in the case of RewritingBasedEvaluator. HybridEvaluator is using an additional componenent called (Q,T)-CompletenessChecker with which it decides whether the user query can be evaluated using the incomplete reasoner or the complete one needs to be employed.

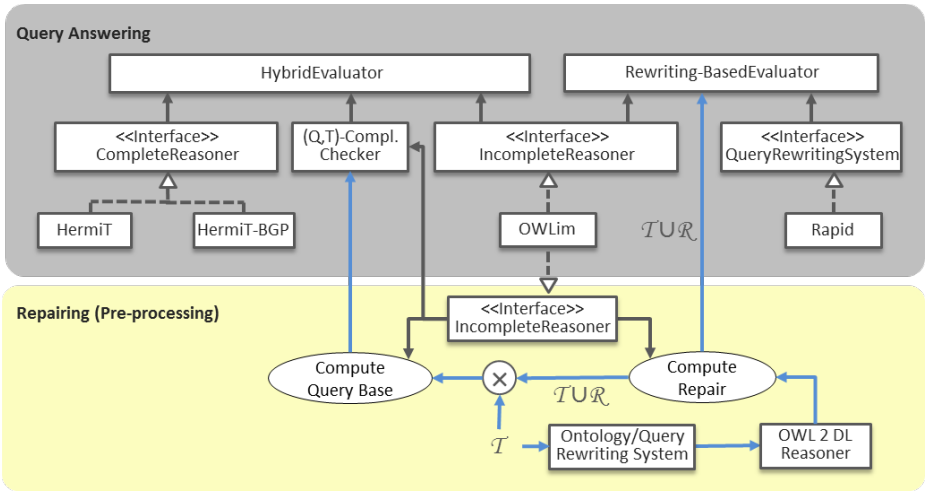

Fig. 1. Main components of Hydrowl

The use of interfaces makes the architecture highly modular as by implementing them one can readily use the query answering techniques of Hydrowl with their reasoner of choice. So far we have provided implementations of CompleteReasoner using the standard HermiT reasoner [10] and HermiT-BGP [5], an implementation of QueryRewritingSystem using Rapid [11], and an implementation of IncompleteReasoner using OWLim, however, we envision that OWL 2 DL systems such as Pellet, query rewriting systems such as Ontop and Clipper, as well as OWL 2 RL systems and triple stores such as Apache Jena, RDFox, and Stardog can be easily integrated.

For both approaches to work, a repair $\mathcal{R}$ or a query base $\mathcal{U}$ for the incomplete system ans used in query answering should have been computed previously at a pre-processing step. This can be done using the repairing package of Hydrowl. Internally, this package uses a rewriting system, an OWL 2 DL reasoner and the system ans with which it again communicates through an interface. The rewriting system is used to produce an initial repair (i.e., some first materialisation of the ground inferences of $\mathcal{T}$) while both the OWL 2 DL reasoner and ans are used as minimisation steps to produce the final repair $\mathcal{R}$. The existing implementation uses again Rapid and HermiT but the choice of these systems compared to the ones used as implementations of CompleteReasoner and QueryRewritingSystem is irrelevant (i.e., one can use completely different systems during query answering). However, clearly, the incomplete reasoner used in this step to produce $\mathcal{R}$ or $\mathcal{U}$ must be the same (or at least equivalent in expressivity) as the one used during query answering.

Subsequently, the query base is used by (Q,T)-CompletenessChecker to determine if the incomplete reasoner is (in)complete for the given user query while the repair needs to be loaded to QueryRewritingSystem in order for the incomplete system to be complete for all ground entailments over the input ontology and data. As mentioned in the previous section, in the hybrid query answering approach one can also compute and load some partial (or even full) repair to the incomplete reasoner. In that case, the "more complete" the partial repair the smaller the query base would be compared to the one computed using $\mathcal{T}$ (e.g., if the partial repair captures all ground entailments for a concept A then the query base won't contain the atomic query $:\text{-}A(x)$).

Finally, due to the systems integrated so far in Hydrowl we note that, the rewriting-based query answering approach supports repairing and query answering (of arbitrary CQs) over ontologies expressed in the $\mathcal{ELHI}$ fragment of OWL 2 DL (a limitation stemming from Rapid which currently supports $\mathcal{ELHI}$), while the hybrid query evaluation approach supports query answering of SPARQL queries over OWL 2 DL ontologies.

4 Evaluation

We report on some experimental evaluations using the HybridEvaluator and RewritingBasedEvaluator classes of Hydrowl to answer queries. We used OWLim as an implementation of IncompleteReasoner, the standard HermiT reasoner as

Table 1. Query Answering Times

	LUBM				UOBM					
Query	1	3	8	9	3	4	9	11	12	14
HermiT-BGP	2.5	1.4	1.4	105	204	5.8	21.6	1.7	1.2	48.7
Hydrowl	.07	.07	.24	.13	.02	.01	.01	.07	.04	35.3

an implementation of CompleteReasoner and Rapid as an implementation of QueryRewritingSystem.

4.1 Hybrid Query Answering

First, using the repairing package of Hydrowl we computed query bases for OWLim for ontologies LUBM and UOBM. For LUBM we required 14.5 seconds while for UOBM we required 48.7 seconds. Some additional manual editing was required for UOBM in order to remove the atomic queries :-Woman(x) and :-PeopleWithManyHobbies(x) from the query base as OWLim is incomplete for them but this is not recognised automatically by the Compute QueryBase procedure of Hydrowl. This is because this procedure is based on a computation of a repair using Rapid which currenlty only supports $\mathcal{ELHI}$ while these queries require reasoning over disjunctions and functional number restrictions. Second, we used HybridEvaluator in order to evaluate all the test queries of LUBM (we used 5 universities) and of UOBM (we used 1 department) and we compared against the HermiT-BGP system [5]. Table 1 presents the results (in seconds) for all the interesting queries (for the rest both systems have similar response times). In grey colour we have marked those queries where Hydrowl uses both OWLim and HermiT. As can be seen, in all queries the hybrid query answering approach of Hydrowl is faster than HermiT-BGP. In some cases the difference is quite significant and this is even in the cases where Hydrowl uses both HermiT and OWLim. It is worth noting query 3 over UOBM which requires non-deterministic reasoning. HermiT-BGP non-deterministicaly checks whether many individuals are instances of the class Student while Hydrowl using both HermiT and OWLim manages to restrict this search space to only a few individuals. Similarly, evaluating query 14 requires non-deterministic reasoning over the class Woman.

4.2 Repairing-Based Query Answering

First, we wanted to evaluate whether repairs for large and complex ontologies can be computed efficiently in practice. Using the repairing package of Hydrowl we managed to compute repairs for 151 out of the 152 ontologies of our dataset. In the vast majority of cases a repair could be computed in less than a few minutes (usually within seconds) and only for the very large ones we required several minutes; Table 2 presents results for the latter. Despite their size and complexity we see that we can compute repairs for them in less than 1 hour which, given that this usually occurs once, we feel is a reasonable amount of

Table 2. $|\mathcal{T}|$ ($|\mathcal{R}|$): number of axioms of the input TBox (repair), t: time in seconds.

$\mathcal{T}$	$\lvert\mathcal{T}\rvert$	$\lvert\mathcal{R}\rvert$	t	$\mathcal{T}$	$\lvert\mathcal{T}\rvert$	$\lvert\mathcal{R}\rvert$	t
Not-Galen	5471	3015(4153)	298(42)	Galen-doc	4229	6051(6176)	1152(28)
Fly	19845	10361(12368)	2884(178)	Galen	4229	3012(3062)	257(24)

Table 3. Loading times for Fly and UOBM for the various ABoxes.

	Universities				
	1	2	5	10	20
UOBM	4.1	6.8	16.2	31.9	73.2
UOBM$\cup\mathcal{R}$	4.4	8.3	24.3	44.9	108.1

(a) UOBM

	$\mathcal{A}$	$2\times\mathcal{A}$	$3\times\mathcal{A}$	$4\times\mathcal{A}$	$5\times\mathcal{A}$
Fly	14.0	21.9	22.7	27.9	31.5
Fly$\cup\mathcal{R}$	31.9	55.1	68.5	93.0	119.3
Fly$\cup\mathcal{R}^-$	33.2	62.1	70.1	100.6	118.2

(b) Fly

time. Actually, if we discard a very expensive minimisation step of repairing, then we can compute some (non-minimal) repair very efficiently while its size is not considerably larger than the minimal one (see Table 2 numbers in brackets).

Next, we loaded the repairs we computed for UOBM and Fly into OWLim; Table 3 presents loading times of the original ontology with (i) data of various sizes (for UOBM we used 1 to 20 universities and for Fly we multiplied the original ABox up to 5 times) and (ii) with and without the computed repairs. As can be seen, the overhead introduced by additionally loading the repair ($\mathcal{R}$) is significant only in the Fly ontology, mostly due to its size, however, note that loading is also usually performed only once. In Fly we have also loaded the non-minimal repair ($\mathcal{R}^-$) and as it turns out there is no significant difference compared to the minimal one (recall that computing it is much more efficient).

Table 4. Results for Answering the Fly Queries

$\mathcal{Q}_1$		$\mathcal{Q}_2$		$\mathcal{Q}_4$		$\mathcal{Q}_5$	
t_{Rapid}	t_{OWLim}	t_{Rapid}	t_{OWLim}	t_{Rapid}	t_{OWLim}	t_{Rapid}	t_{OWLim}
0.31	0.31	0.90	1.28	0.07	0.04	0.05	0.02

Finally, we have used RewritingBasedEvaluator to answer all 4 non-SPARQL queries of Fly (using the original ABox); Table 4 presents the results where t_{Rapid} is the time required by Rapid and t_{OWLim} the time required by OWLim (total time is their sum). As we can see in most cases we were able to compute and evaluate a rewriting almost instantaneously. The good behaviour of Hydrowl can be attributed to the fact that most hard work is pushed to a pre-processing step that is materialising all ground entailments into the repair and explicating them by loading the ontology, the repair, and the data into OWLim.

Acknowledgements. The work was funded by a Marie Curie Career Reintegration Grant within European Union's 7th Framework Programme (FP7/2007-2013) under REA grant agreement 303914.

References

1. Kiryakov, A., Bishoa, B., Ognyanoff, D., Peikov, I., Tashev, Z., Velkov, R.: The Features of BigOWLIM that Enabled the BBCs World Cup Website. In: Workshop on Semantic Data Management (SemData). (2010)
2. Motik, B., Horrocks, I., Kim, S.M.: Delta-Reasoner: A Semantic Web Reasoner for an Intelligent Mobile Platform. In: Proceedings of the 21st International World Wide Web Conference (WWW 2012). (2012) 63–72
3. Ortiz, M., Calvanese, D., Eiter, T.: Data complexity of query answering in expressive description logics via tableaux. Journal of Automated Reasoning **41**(1) (2008) 61–98
4. Glimm, B., Lutz, C., Horrocks, I., Sattler, U.: Conjunctive query answering for the description logic $\mathcal{SHIQ}$. Journal of Artificial Intelligence Research (JAIR) **31** (2008) 157–204
5. Kollia, I., Glimm, B.: Optimizing sparql query answering over owl ontologies. J. Artif. Intell. Res. (JAIR) **48** (2013) 253–303
6. Motik, B., Cuenca Grau, B., Horrocks, I., Wu, Z., Fokoue, A., (Editors), C.L.: OWL 2 Web Ontology Language Profiles. W3C Recommendation (2009)
7. Wu, Z., Eadon, G., Das, S., Chong, E.I., Kolovski, V., Annamalai, M., Srinivasan, J.: Implementing an inference engine for RDFS/OWL constructs and user-defined rules in oracle. In: Proc. of ICDE, IEEE (2008) 1239–1248
8. Stoilos, G., Cuenca Grau, B., Motik, B., Horrocks, I.: Repairing ontologies for incomplete reasoners. In: Proceedings of the 10th International Semantic Web Conference (ISWC-11), Bonn, Germany. (2011) 681–696
9. Zhou, Y., Nenov, Y., Grau, B.C., Horrocks, I.: Complete query answering over horn ontologies using a triple store. In: Proc. of the 12th International Semantic Web Conference (ISWC), Springer LNCS (2013)
10. Glimm, B., Horrocks, I., Motik, B., Stoilos, G., Wang, Z.: Hermit: An owl 2 reasoner. Journal of Automated Reasoning (JAR), In Press (2014)
11. Trivela, D., Stoilos, G., Chortaras, A., Stamou, G.: Optimising resolution-based rewriting algorithms for dl ontologies. In: Proceedings of the 26th Workshop on Description Logics (DL 2013), Ulm, Germany. (2013)
12. Stoilos, G.: Ontology-based data access using rewriting, OWL 2 RL systems and repairing. In: Proceedings of the 11th European Semantic Web Conference (ESWC 2014). (2014)
13. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Rosati, R.: Tractable reasoning and efficient query answering in description logics: The DL-Lite family. Journal of Automated Reasoning **39**(3) (2007) 385–429
14. Artale, A., Calvanese, D., Kontchakov, R., Zakharyaschev, M.: The dl-lite family and relations. Journal of Artificial Intelligence Research (JAIR) **36** (2009) 1–69
15. Pérez-Urbina, H., Motik, B., Horrocks, I.: Tractable query answering and rewriting under description logic constraints. Journal of Applied Logic **8**(2) (2010) 186–209
16. Eiter, T., Ortiz, M., Simkus, M., Tran, T.K., Xiao, G.: Query rewriting for Horn-$\mathcal{SHIQ}$ plus rules. In: Proc. of AAAI. (2012)

17. Cuenca Grau, B., Motik, B., Stoilos, G., Horrocks, I.: Computing datalog rewritings beyond horn ontologies. In: Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence (IJCAI 2013). (2013)
18. Kaminski, M., Nenov, Y., Grau, B.C.: Datalog rewritability of disjunctive datalog programs and its applications to ontology reasoning. In: Proceedings of AAAI 2014. (2014)
19. Stoilos, G., Stamou, G.: Hybrid query answering for owl ontologies. Proceedings of the 21st European Conference on Artificial Intelligence (ECAI 14) (2014)