A Pair of OWL 2 RL Reasoners

Martin J. O'Connor, Amar Das

Stanford Center for Biomedical Informatics Research
Stanford, CA 94305, U.S.A. martin.oconnor@stanford.edu

OWL 2 RL is an expressive OWL profile designed to be amenable to implementation using conventional rule-based technologies. The profile specification provides a direct pathway to develop OWL reasoners using off-the-shelf rule engines. In this paper, we describe our experiences developing two OWL 2 RL reasoners using the popular Jess and Drools rule systems. A useful feature of these implementations is that they support the SWRL rule language and the SQWRL query language. We describe an initial validation of the system carried out using a large set of biomedical ontologies from the BioPortal ontology repository. Free, open source implementations of these reasoners have been released for the Protégé ontology development environment. The implementations demonstrate that powerful OWL 2 RL-based reasoners can be developed relatively quickly, though significant performance enhancements remain to be carried out in the current implementations.

1 Introduction

The OWL 2 W3C Recommendation includes several language profiles [1]. These profiles are restricted subsets of OWL that trade some expressivity to provide more desirable computational guarantees. Primarily, these profiles restrict the types of OWL constructs that can be used in an ontology or place syntactic restrictions on how those constructs can be used. Three profiles are provided: OWL 2 EL, which is designed for ontologies that contain a large number of classes or properties; OWL 2 QL, which is aimed primarily at query answering; and OWL 2 RL, which is aimed at applications that require quite a bit of the expressivity provided by OWL 2 DL but also require scalable reasoning. Many real world ontologies do not use the full set of features provides by OWL 2 DL and often fall into one or more of these profiles.

A notable feature of OWL 2 RL is that the profile is designed to be implementable using standard rule-based systems. The specification document contains a set of first order implication rules that can be used to implement such an OWL 2 RL reasoner. The rules are described in terms of first order implications and can be applied to sets of RDF triples. In the specification, each rule is given a unique name. For example, a rule called `eq-sym` that describes the symmetric property of the `owl:sameAs` axiom expressed using ternary predicate `T` is written:

```
T(?x, owl:sameAs, ?y) → T(?y, owl:sameAs, ?x)
```

Approximately 80 rules are described in the specification. An attractive feature of the OWL 2 RL profile is that reasoning is polynomial with respect to the size of the ontology.

Since its release, a small number of OWL 2 RL reasoners have been developed. Once of the earliest is DLEJena [2]. This system used the Jena rule system in combination with the Pellet reasoner to separately generate ABox and TBox entailments using a subset of OWL 2 RL rules. A more recent implementation describes the development of a highly scalable OWL 2 RL reasoner that can be applied to RDF-based data inside the Oracle database system [3]. A specification of OWL 2 RL rules using the RIF specification has also been developed [4].

To date, however, implementations exist either in prototype form or are inside proprietary commercial systems. The availability of free, open source OWL 2 RL reasoners that are integrated with existing ontology development environments could be of immense benefit. In particular, this profile can be of great use to ontology authors who need more expressivity than RDF but who do not require the full power of OWL 2 DL. Interoperability with the widely-used Semantic Web Rule Language (SWRL) [5] and querying support is also desirable. In this paper, we describe such a system. The implementation provides a pair of reasoners using the Jess [6] and Drools [7] rule engines, both of which are provided as plugins to the Protégé ontology development tool [8]. Both reasoners interoperate with SWRL and also provide OWL querying via the SQWRL query language [9].

2 Implementation

In the OWL 2 RL specification, rules operate on an RDF triple-based serialization of an OWL ontology. Taken together, the full set of rules provides a partial axiomatization of the OWL 2 RDF-based semantics. In our implementation we decided not to support the full RDF-based semantics but instead to concentrate on a practical subset conforming to the direct semantics. Thus, instead of supporting arbitrary RDF graphs, our rules are effectively applied to a restricted subset of a graph conforming to these direct semantics.

Ontology and OWL 2 RL Rule Representation

In the specification documents, rules are subdivided into eight functional groups, each specified in a numbered table. For example, the group specified in table 7 contains rules relating to the semantics of the schema vocabulary. An example rule from this table describing the transitivity of subclass axioms is expressed as:

```
T(?c1, rdfs:subClassOf, ?c2) ^ T(?c2, rdfs:subClassOf, ?c3)
                → T(?c1, rdfs:subClassOf, ?c3)
```

Each OWL 2 RL rule in the specification is short and relatively comprehensible and is designed to be mapped directly to a target rule language. Since each rule effectively operates on triples, a standard approach would be to develop a

representation of a triple-based data structure in the target rule system to represent the input OWL ontology. The OWL 2 RL rules could then be directly translated to the native rule language to operate against that data structure. Since we decided to use direct semantics, we can raise the abstraction level and instead represent the input OWL ontology using an axiom-based data structure. That is, we take each OWL axiom type and develop a native rule engine representation of that axiom; we then take each OWL 2 RL rule and express it in terms of those axioms.

This approach proved particularly attractive using the Java-based Drools rule system because Drools allows rules to operate directly on Java objects. Thus, for example, an OWL subclass axiom can be represented as a Java class `SCA` with two fields, `sub` and `sup`, representing the classes in the relationship. The Drools rule representing the `scm-sco` OWL 2 RL rule can then be written:

```
rule scm_sco
when SCA($c1:sub, $c2:sup) SCA(sub==$c2, $c3:sup)
then SCA sca=new SCA($c1, $c3); inferrer.infer(sca);
end
```

Here, `inferrer` refers to a Java object that accumulates inferred OWL axioms and asserts them into working memory. As can be seen, the translation from the OWL 2 RL specification rule is relatively direct. In principle, this approach could be applied directly to Java objects representing OWL axioms supplied by existing OWL APIs, though a more targeted, lightweight intermediate representation is usually desirable. The approach is particularly useful for dealing with OWL 2 RL rules that use OWL class expressions, which have verbose RDF serializations. For example the `scvm-svf1` OWL 2 RL rule that uses `owl:someValueFrom` class expressions, represented here using the Java class `OSVFCE`, can be written:

```
rule scm_svf1
when OSVFCE($c1:ceid, $p:p, $y1:v) OSVFCE($c2:ceid, p==$p, $y2:v)
SCA(sub==$y1, sup==$y2)
then SCA sca=new SCA($c1, $c2); inferrer.infer(sca);
end
```

Writing rules at this level has multiple benefits beyond ease of axiom mapping and rule comprehension. In Drools, the types of the axiom elements are type checked at run time. Also, this representation provides opportunities to optimize at axiom level rather than at the triple level.

A similar approach can also be used with the Java-based Jess rule system, though syntactically the Jess rule language is based on Lisp so the resulting rules are less Java like. For example, the above subclass rule is written in Jess as follows:

```
                    (defrule scm-sco
  (sca (sub ?c1) (super ?c2)) (sca (sub ?c2) (super ?c3))
    => (sca (sub ?c1) (super ?c3)) (inferSCA ?c1 ?c3))
```

Here, `inferCAA` is a user-defined function to process inferred subclass axioms.

In the case of both Jess and Drools, the named OWL 2 RL rules in the specification dealing with OWL concepts are directly translatable to rules that operate on OWL axioms.

The profile also specifies unnamed list processing rules that deal with collections of objects. These collections are represented as RDF lists. These rules could be represented using a set of recursive rules that traverse these list structures at run time. However, this approach could be computationally expensive. A common alternative approach is to treat these rules as templates and preprocess the input ontology and generate instantiations of these template rules for the lists in the input ontology [4]. Our implementation adopts the latter approach.

An additional difficulty is dealing with datatype rules. Rules dealing with equality and inequality of data values can use the data value handling mechanisms of the underlying rule engine for simple datatypes. However, the type hierarchies of the supported datatypes must also be considered in the rules. Also, the OWL 2 RL rules specify type checking of datatype values. The implementations currently do not fully support these rules and deal only with direct datatype equality and inequality tests.

Controlling the OWL 2 RL Reasoners in Protégé-OWL

The Jess- and Drools-based OWL 2 RL reasoners are included in the 3.5 alpha release of Protégé-OWL. Currently, these reasoners act as engines for the SWRL rule language and the SQWRL query language, though they can also be used with OWL ontologies that do not contain rules or queries.

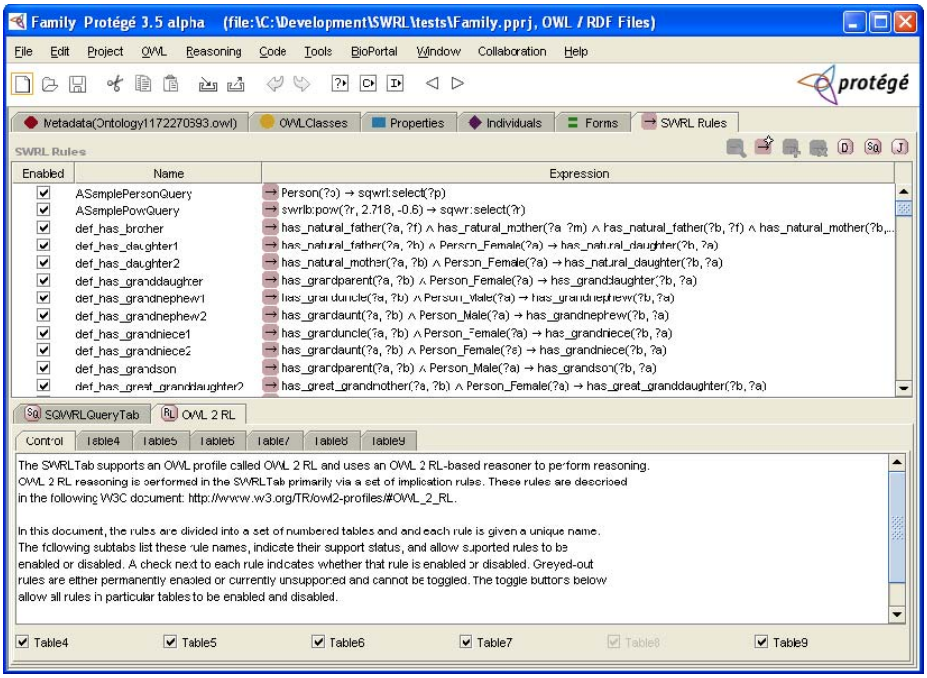

Figure 1. Screen shot of the Protégé-OWL SWRLTab with the OWL 2 RL control panel selected.

We developed a set of graphical interfaces for controlling these reasoners. These interfaces are accessible with the Protégé-OWL SWRLTab [10,11]. Figure 1 shows this interface as displayed in the SWRLTab's SQWRL query plugin. The interface permits control of the OWL 2 RL inference process by allowing the selective enabling and disabling of the rules or tables of rules. This interface also provides a control tab to indicate if rule tables are active or inactive.

The interface also contains sub tabs for controlling individual OWL 2 RL rules (Figure 2). These sub tabs list these rule names, indicate their support status, and allow supported rules to be enabled or disabled. A check next to each rule indicates whether that rule is enabled or disabled. Grayed-out rules are either permanently enabled or currently unsupported and cannot be toggled. Support is also provided for saving the OWL 2 RL rule settings for a particular ontology. These settings are saved as annotation properties in that ontology and are reused if the ontology is later opened. Control of OWL 2 RL rule activation is also provided via a Java API.

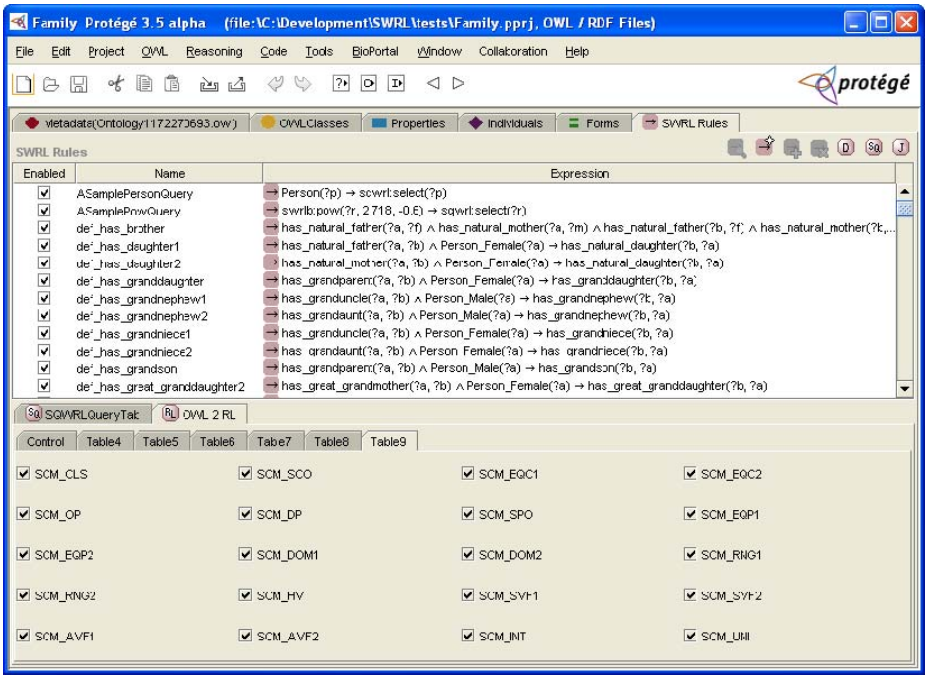

Figure 2. Screen shot of OWL 2 RL control sub tab for controlling individual rules from table 9 of the OWL 2 RL specification.

3 Testing and Evaluation

We performed initial evaluation and testing of our two reasoners using ontologies from the BioPortal ontology repository [12]. Of the 305 ontologies currently in the repository, we found 87 that were within the OWL 2 RL profile. We excluded three of these ontologies from the evaluation because we had difficulties processing them due to ontology import resolution issues, leaving a test suite of 84 ontologies. We processed these ontologies using both the Jess- and Drools-based reasoners and generated materialized ontologies from each of them. We repeated this process with the HermiT reasoner [13]. In each case we recorded the time taken to generate the materialized views. We excluded file load and saving times from this total. We verified that the three reasoners produced identical entailed ontologies.

Both the Jess and Drools implementations performed similarly. Execution times ranging from the hundreds of milliseconds for the smallest ontologies, which contained a few dozen OWL axioms, to 30 seconds for the largest, which contained 50 thousand asserted axioms. The Drools implementation was marginally faster in most cases, but in all cases the reasoning times of the two implementations were within 20% of each other. This result is not surprising given that a similar Rete-based algorithm is used by both rule systems and the almost identical ontology and OWL 2 RL rule modeling strategies used. Not surprisingly, the HermiT reasoner significantly outperformed our pair of reasoners. It was typically 5 to 6 times faster. For some well-known ontologies it was over ten times faster, perhaps reflecting a training effect. Since our implementations adopt a fairly direct translation of the rules in the specification and do not yet attempt some obvious optimizations, it is likely that this gap can be reduced somewhat.

It is important to note that the ontologies used in the test were heavily TBox biased and contained few OWL individuals. Naïve implementations of OWL 2 RL reasoners are known to perform poorly on ontologies with large ABoxes [14]. Hence, significant optimizations remain to be performed. Fortunately, a variety of very effective optimizations have been described in the literature [3, 15], some of which we are currently implementing. We are also in the process of evaluating our system using a set of publicly available OWL 2 RL conformance tests [16].

4 Discussion

We have described the development of a pair of OWL 2 RL reasoners using the Jess and Drools rules engines. The implementations are integrated into the Protégé-OWL's and are open source and free, though the use of Jess does require a license for non U.S. government or non academic use. The Drools rule system is both open source and free. Both OWL 2 RL reasoners are configurable via graphical and API based interfaces. They are available in the current 3.5 alpha release of Protégé-OWL. We are planning an OWLAPI-compliant release for Protégé 4.2 in the next few weeks. The implementations support the SWRL rule language and offer query support via the SQWRL query language. The extensive set of SWRL built-in libraries provided by the Protégé-OWL SWRLTab can also be used with these reasoners.

As described, the current implementations need further compliance testing and significant optimizations. The implementations are also currently limited in that they deal with in-memory ontology only, which restricts the size of ontologies that they can be applied to. However, they have been show to perform reasonably well on a variety of real world biomedical ontologies. Of particular note is the percentage of OWL 2 RL compliant ontologies in the BioPortal repository. Of the 304 ontologies in the current repository, 89 fit within the OWL 2 RL profile. It is not clear than any were specifically designed to fit within this profile, indicating that perhaps more could fit within it with possibly minor effort. We intend to investigate the remaining ontologies to see how many could be amenable to these modifications and thus fit with the profile.

Acknowledgements

We would like to thank Mark Proctor of Red Hat, Inc., the developer of Drools, for his assistance in the development of the Drools implementation of the OWL 2 RL reasoner. We would also like to thank Matthew Horridge for providing the set of OWL 2 RL compliant BioPortal ontologies for analysis and for his help in setting up the tests. This research was supported in part by the N.L.M. under grants LM007885 and LM009607.

References

[1] Motik, B. Cuenca Grau, B., Horrocks, I., Wu, Z., Fokoue, A, Lutz, C. OWL 2 Web Ontology Language Profiles, W3C Recommendation. www.w3.org/TR/owl2-profiles, 2009.
[2] Meditskos, G. and Bassiliades, N. DLEJena: A Practical Forward-Chaining OWL 2 RL Reasoner Combining Jena and Pellet. Journal of Web Semantics, 8(1), 2010.
[3] Kolovski, V, Wu, Z., Eadon, G. Optimizing Enterprise-scale OWL 2 RL Reasoning in a Relational Database System. 9th International Semantic Web Conference, Shanghai, China, 2010.
[4] Reynolds, D. OWL 2 RL in RIF, W3C Working Group Note. www.w3.org/TR/rif-owl-rl/, 2010.
[5] Horrocks, I., Patel-Schneider P.F., Boley, H., Tabet, S., Grosof, B., Dean, M. SWRL: A Semantic Web Rule Language Combining OWL and RuleML. W3C, www.w3.org/Submission/SWRL/, 2004.
[6] Friedman, E. Jess in Action: Rule-based Systems in Java. Manning Publications Co., Greenwich, CT, 2003.
[7] Drools: www.jboss.org/drools. Last Accessed May, 2012.
[8] Knublauch, H., Fergerson, R., Noy, N., Musen, M. The Protégé OWL Plugin: An Open Development Environment for Semantic Web Applications. 3rd International Semantic Web Conference, Horoshima, Japan, 2004.
[9] O'Connor, Das, A. SQWRL: a Query Language for OWL. OWL: Experiences and Directions, 6th International Workshop, Chantilly, VA, 2009.

[10] O'Connor, M.J., Nyulas, C.I., Shankar, R.D., Das, A.K., Musen, M.A. The SWRLAPI: A Development Environment for Working with SWRL Rules. OWL: Experiences and Directions, 5th International Workshop, Karlsruhe, Germany, 2008.
[11] O'Connor, M.J., Knublauch, H., Tu, S., Grosof, B., Dean, M., Grosso, W, and Musen, M. Supporting Rule System Interoperability on the Semantic Web with SWRL. 4th International Semantic Web Conference, Galway, Ireland, 2005.
[12] BioPortal: bioportal.bioontology.org. Last Accessed May, 2012.
[13] Glimm, B., Horrocks, I, Motik, B, Stoilos, G. Optimising Ontology Classification. 9th International Semantic Web Conference, Shanghai, China, 2010.
[14] Hogan, A. and Decker, A. On the Ostensibly Silent 'W' in OWL 2 RL. 3rd International Conference of Web Reasoning and Rule Systems, Chantilly, VA, 2009.
[15] Bishop, B, Bojanov, S. Implementing OWL 2 RL and OWL 2 QL Rule-sets for OWLIM. OWL: Experiences and Directions, 8th International Workshop, San Francisco, CA, 2011.
[16] OWL 2 Web Ontology Language: Conformance and Test Cases. W3C, www.w3.org/TR/owl2-test/, 2008.