Implementing OWL 2 RL and OWL 2 QL rule-sets for OWLIM

Barry Bishop, Spas Bojanov

Ontotext AD, 135 Tsarigradsko Chaussee, Sofia 1784, Bulgaria
{barry.bishop,spas.bojanov}@ontotext.com

Abstract. OWLIM is a family of semantic repository components that comprise a native RDF store, a reasoner and a query answering engine. The reasoner is based on R-entailment defined by ter Horst, where inference rules are applied directly to RDF triples. Each rule is made up of a number of premises and conclusions, each of which is an RDF triple pattern with variables allowed at any position. This paper describes an implementation of the OWL 2 RL and OWL 2 QL profiles using this scheme, what modifications were necessary to the rule-engine and what features of the profiles could not be implemented or were modified to make implementing them practical.

Keywords: inference, OWL 2, RDF, rules, semantic-web, triple-store

1 OWLIM

The OWLIM [10, 5] family of semantic repository components is implemented in Java [1] and packaged as a Storage and Inference Layer (SAIL) for the Sesame openRDF framework [7]. They are comprised of a native RDF store, a reasoner and a query answering engine that supports the SeRQL [6] and SPARQL [13] languages. The reasoner uses predominantly forward-chaining to apply the selected inference rules directly to RDF statements (triples), although statements are actually stored as quads – triples plus named graphs (also called 'context' in Sesame terminology). The rule-language is based on R-entailment [17] defined by ter Horst.

There are two editions of OWLIM: SwiftOWLIM and BigOWLIM, that share the same rule-language and are identical in terms of reasoning expressivity and integration. Whereas SwiftOWLIM is an entirely in-memory system, BigOWLIM uses a file-based storage layer and has a number of query and reasoning optimisations. Most significantly, BigOWLIM has special support for `owl:sameAs` by maintaining equivalence classes for individuals. During query-answering, equivalence classes are enumerated in a backward-chaining manner. Typically, SwiftOWLIM can manage millions of explicit statements on desktop hardware, whereas BigOWLIM can manage billions of statements and multiple concurrent user sessions.

Several standard rule-sets are built into all editions of OWLIM, namely: RDFS, OWL-Horst (similar to pD*), OWL-Max (RDFS with most of OWL

Lite) and recently the OWL 2 profiles RL and QL. Users are able to build their own custom rule-sets using datalog like rules with inequality constraints. The general format for defining rules is shown in figure 1.

```
Id: <Rule_Id>
  <Premise #1>      [Optional inequality constraints]
      . . .
  <Premise #n>      [Optional inequality constraints]
---------------------
  <Conclusion #1>   [Optional inequality constraints]
      . . .
  <Conclusion #m>   [Optional inequality constraints]
```

Fig. 1. The OWLIM rule format

Premises and conclusions are triple patterns with variables in any position. Every premise may additionally contain inequality constraints stating that the value of one or more variables in the statement is not a blank node or must not be equal to a full URI, a short name or the value of another variable from the same rule. If an inequality constraint does not hold, then the rule does not *fire*. Conclusions may also have inequality constraints. In the event that a conclusion constraint does not hold, the rule will still *fire*, except that the conclusion adjacent to the failing constraint will not be inferred. Free variables in the head of a rule (without a binding in the body) are used to infer new blank nodes.

The example in figure 2 shows an implementation of the OWL 2 RL functional property rule `prp-fp`. The symbols `p`, `x`, `y1` and `y2` are variables and there is a single constraint in the rule body that prevents the rule from firing if `y1` is equal to `y2`.

```
Id: prp_fp
  p <rdf:type> <owl:FunctionalProperty>
  x p y1                    [Constraint y1 != y2]
  x p y2
  ------------------------------
  y1 <owl:sameAs> y2
```

Fig. 2. An example rule using the OWLIM rule language

BigOWLIM also supports consistency checks using a syntax similar to rule definitions, but without the conclusions (rule head).

2 OWL 2 RL

The OWL 2 Profiles specification [11] provides a definition for OWL 2 RL, which is described as follows: "The OWL 2 RL profile is aimed at applications that require scalable reasoning without sacrificing too much expressive power". The profile is designed to be amenable to implementation on rule-engines and to assist with this the specification provides an RDF-Based Semantics in the form of first-order implications that should be applied directly to RDF graphs.

In order to make it feasible to implement this profile on rule engines while providing some "desirable computational guarantees", certain restrictions are made on the use of OWL 2 [12] constructs. In particular, there is no requirement for existential quantification or non-deterministic reasoning. OWL 2 RL is therefore defined in two ways:

- By restrictions placed on OWL 2 Full in the use and position of certain OWL 2 language features;
- As a set of entailment rules to be applied to the RDF serialisation of an OWL ontology, where these rules represent a partial axiomatisation of the complete OWL 2 RDF-Based Semantics [15].

In order to distinguish these two definitions, the term *OWL 2 RL/RDF rules* will be used to identify the latter case.

The first-order implications provided in the W3C specification were used as a starting point for the implementation of OWL 2 RL/RDF rules using OWLIM's rule notation. These rules are grouped in to separate tables for defining the semantics for: equality, property axioms, classes, class axioms, datatypes and schema vocabulary. The rules themselves are offered as first-order implications over a ternary predicate `T` representing the entire graph of RDF statements – variables are allowed in any position. The rules take a variety of forms:

triple pattern rules The rule body and head are made up of atomic formulae representing triples in the RDF graph;

assertional rules The rule body is empty, in which case they can be considered as being always applicable, e.g. rule `cls-thing` that asserts that `owl:Thing rdf:type owl:Class`;

consistency checks The head of these rules contains `false` only, in which case the input RDF graph should be considered inconsistent when the premises of the rule body hold;

list rules These rules make use of a shorthand notation for processing RDF collections [9], e.g. when defining classes as the intersection or union of a closed set of classes;

data-type rules These rules require special processing for data-types, e.g. rule `dt-eq` that asserts that `lt1 owl:sameAs lt2` for all literals `lt1` and `lt2` with the same data value.

Triple pattern rules, assertional rules and consistency checks are straightforward to implement using OWLIM's rule language. However, list rules involve

T(h, rdf:first, e1)	T(h, rdf:rest, z2)
T(z2, rdf:first, e2)	T(z2, rdf:rest, z3)
...	...
T(zn, rdf:first, en)	T(zn, rdf:rest, rdf:nil)

Fig. 3. The expansion of $LIST[h, e_1, ..., e_n]$

processing RDF collections [9] that are expressed in the partial axiomatisation using an informal ellipsis notation: $LIST[h, e_1, \ldots, e_n]$. These atoms are shorthand for an arbitrary length series of RDF statements that describe a closed set of values $[e_1, \ldots, e_n]$ identified by h (often a blank node). This expansion, shown in figure 3, is used in the definition of the following OWL 2 RL rules; `eq-diff2`, `eq-diff3`, `prp-spo2`, `prp-adp`, `prp-key`, `cls-int1`, `cls-int2`, `cls-oo`, `cls-uni`, `cax-adc`, `scm-int`, `scm-uni`. There are two ways of handling such rules. They can either be expressed as a set of recursive rules that traverse the RDF list structures at run-time or they can be regarded as templates that, for a given RDF input document, can be translated to a set of triple pattern rules as part of a preprocessing step. Such a preprocessing step would require an examination of the actual definitions used in the input ontology in order rewrite some of the entailment rules *long-hand*, e.g. if the input ontology contains a definition of a property chain called `:uncle` in terms of the chain `:parent` and `:brother` then the rule `prp-spo2` shown in figure 5 can be used as a template to instantiate an ontology-specific rule shown in figure 4 that hard-codes the property chain. However, preprocessing is not a practical solution, because

prp-spo2/ uncle	$T(?u_1, :parent, ?u_2)$ $T(?u_2, :brother, ?u_3)$	$T(?u_1, :uncle, ?u_3)$

Fig. 4. An example instantiation of prp-spo2 for a specific input ontology

it requires the re-computation of entailment rules for each input ontology and whenever an ontology changes – this is before any actual entailments are computed or re-computed. In the case of OWLIM, this is further complicated by the fact that rule definitions in OWLIM are compiled to Java byte code for faster execution making them difficult to modify at run-time.

Therefore the requirement is to find a rule-set that can be expressed using OWLIM's rule-language that captures the semantics of OWL 2 RL/RDF rules without requiring any other processing. This poses problems, due to the fact that list rules do not have a corresponding first-order construction, e.g. see OWL 2 RL rule `prp-spo2` in figure 5.

The Rule Interchange Format (RIF) W3C Working Group [4] is chartered to specify a format for rules that functions as an inter-lingua so that rules can be shared across diverse systems. The working group have made several W3C recommendations, including (amongst others):

prp-spo2	T(?p, owl:propertyChainAxiom, ?x) LIST[?x, ?p_1, ..., ?p_n] T(?u_1, ?p_1, ?u_2) T(?u_2, ?p_2, ?u_3) ... T(?u_n, ?p_n, ?u_{n+1})	T(?u_1, ?p, ?u_{n+1})

Fig. 5. OWL 2 RL entailment rule with an example of using $LIST[h, e_1, ..., e_n]$

Core A core dialect (subset of BLD and PRD) similar in expressivity to Datalog[18];

BLD The Basic Logic Dialect corresponding to definite Horn rules with equality, plus extensions for XML Schema data-types and F-logic frames and objects;

PRD Production Rule Dialect, which is similar to BLD, but where rules can affect changes, such as modifying data;

FLD A Framework for Logic Dialects for specifying all RIF logic dialects;

DTB Datatypes and Built-Ins, i.e. the supported datatypes, built-in functions and predicates that are supported by RIF dialects;

Further to this, the working group have published a W3C note [14] showing how OWL 2 RL/RDF rules can be implemented using RIF-Core. This note was used as the basis for creating the OWL 2 RL rule-set for OWLIM. In order to under-

```
Forall ?p ?last ?pc ?start (
  ?start[?p->?last] :- And (
      ?p[owl:propertyChainAxiom->?pc]
      _checkChain(?start ?pc ?last) ))

Forall ?start ?pc ?last ?p ?tl (
  _checkChain(?start ?pc  ?last) :- And (
      ?pc[rdf:first->?p rdf:rest->?tl]
      ?start[?p->?next]
      _checkChain(?next ?tl ?last) ))

Forall ?start ?pc ?last ?p (
  _checkChain(?start ?pc  ?last) :- And (
      ?pc[rdf:first->?p rdf:rest->rdf:nil]
      ?start[?p->?last] ))
```

Fig. 6. The three RIF rules for implementing prp-spo-2

stand how RIF-Core is used to model the OWL 2 RL rules that use lists, consider the RIF implementation of `prp-spo2` that consists of three rules that utilize an auxiliary ternary predicate `_checkChain` as shown in figure 6. These three (recursive) rules infer tuples in the ternary predicate `_checkChain` that work

backwards from the end of lists of predicates forming links between individuals and the last individual in a chain that use the predicates in the same order. If the chain is indeed referred to by a property chain (`owl:propertyChainAxiom`) then the final inference is to connect the first and last individual in any chain with the named property chain. Many more inferences are created this way when compared to the preprocessing approach, but this technique works for arbitrary length collections.

However, OWLIM was designed with R-entailment in mind, where rules are applied directly to the entire graph of stored RDF triples. This presents a problem for implementing auxiliary predicates, because triples are the input to the rule engine (any context is simply ignored) and the output of the rule engine (also triples) are added to the native triple store.

One possible to solution is to use a kind of reification of the tuples belonging to auxiliary predicates and store these in the normal statement indices. This is possible using OWLIM, because of the fact that unbound variables in rule heads are used to infer new, unique blank nodes that can be used to identify a new tuple and RDF triples can be used to associate the 'tuple' with its members. For example, the following tuple:

```
_checkChain(start pc last)
```

could be written using three RDF statements (where `b` is a blank node):

```
b onto:_checkChain1 start
b onto:_checkChain2 pc
b onto:_checkChain3 last
```

Although such a technique produces the correct inferences, it has a number of drawbacks, namely:

- Every tuple for an auxiliary ternary predicate requires a new blank node and three RDF triples, which increases storage and computation complexity;
- Due to the fact that OWLIM does not create any truth maintenance information, it is impossible to know which blank nodes were created from which inference rule and premises, therefore such inferences can not be retracted when the supporting premises no longer hold.

A better method to store tuples for ternary predicates was devised that makes use of the fact that OWLIM stores and indexes quads – RDF triples with context. Such an arity-4 collection can be used to store ternary predicate names their three members. Therefore, in order to support the RIF style implementation rules that use auxiliary ternary predicates (essentially all rules that use $LIST[h, e_1, ..., e_n]$), the OWLIM rule language was extended to include the optional context, i.e. quad patterns are now supported in rule premises and conclusions. This ability to specify the context for a statement pattern provides a means to assert tuples for ternary predicates. Continuing with our example, the final version of the OWLIM rules that correspond to the RIF implementation

```
Id: prp_spo2_1
    p <owl:propertyChainAxiom> pc
    start pc last                    [Context <onto:_checkChain>]
    ----------------------------
    start p last

Id: prp_spo2_2
    pc <rdf:first> p
    pc <rdf:rest> t                  [Constraint t != <rdf:nil>]
    start p next
    next t last                      [Context <onto:_checkChain>]
    ----------------------------
    start pc last                    [Context <onto:_checkChain>]

Id: prp_spo2_3
    pc <rdf:first> p
    pc <rdf:rest> <rdf:nil>
    start p last
    ----------------------------
    start pc last                    [Context <onto:_checkChain>]
```

Fig. 7. The three OWLIM rules equivalent to the RIF implementation of prp-spo-2

of prp-spo2 are given in figure 7. The 'name' of the auxiliary predicate, in this case `_checkChain`, is used in the context position of quads.

In other words, the context `<onto:_checkChain>` is used to associate inferred RDF statements with the `_checkChain` auxiliary ternary predicate. This technique allows the existing RDF statement storage and indexing mechanisms to be reused for this and other auxiliary predicates for all rules that use the $LIST[h, e_1, \ldots, e_n]$ construct, given at the start of this section.

The only difficulty now is that intermediate statements (statements with the `onto:_checkChain` context) generated by the `prp-spo2` OWLIM rules are unsound relative to the semantics of OWL 2 RL. Without any other modification, such statements would be used as input to query answering, when in fact they are simply intermediate values generated as part of the reasoning process. In order to avoid 'polluting' the database model, further modifications to the rule engine and storage mechanisms were required. Even though such intermediate tuples are fully fledged RDF statements, they are flagged in the indices as being the result of the reasoning process and these statements are skipped by the query answering engine.

Datatype rules provide type checking and value equality/inequality checking for typed literals across a set of supported data types. OWLIM does not provide the extended support for typed literals, introduced with the $D*$ entailment extension of the RDFS semantics [16]. Although such support is conceptually clear, it does not scale to large dataset sizes in a rule-based, forward-chaining environment. For example, rule `dt-diff` requires that an `owl:differentFrom`

statement is inferred for all non-equal pairs of literals – for a dataset with one million unique literals (a small number by today's standards) this will infer another trillion statements (the Cartesian product).

3 OWL 2 QL

The OWL 2 QL profile [11] is designed so that data stored in a standard relational database system can be queried through an ontology via a simple rewriting mechanism, i.e. by rewriting the query into an SQL query that is then answered by the RDBMS system, without requiring any changes to the data. OWL 2 QL is based on DL-Lite$_R$, a variant of DL-Lite [8] that does not require the unique name assumption.

At first sight, the design constraints of OWL 2 QL would seem to make it unsuitable for implementation in a forward chaining, rule-based environment, where OWLIM computes all inferences at the time data is loaded or modified. However, an initial analysis showed that the bulk of the semantics of OWL 2 QL can be captured in rules, therefore, for the purpose of expanding the range of rule-sets included with OWLIM and to offer users a wider choice in the expressivity versus complexity spectrum, a rule-set for this OWL profile was developed. The most problematic cases for modelling the semantics of OWL 2 QL arise due to existential quantification as demonstrated in the OWL 2 QL ontology in figure 8.

```
Prefix ( : = <http://example.org/> )
Ontology (
  SubClassOf (:GrandPa ObjectSomeValuesFrom (:fatherOf owl:Thing))
  ClassAssertion ( :GrandPa :Tom ) )
```

Fig. 8. An example OWL 2 QL ontology using existential quantification

Using OWL 2 QL semantics, the conjunctive query `q(x) :- fatherOf(x,y)` should return `:Tom` as a result. The above ontology does not define an object that `:Tom` is the father of, it only asserts that there must be one, because `:Tom` is a `:GrandPa` and so must have a `:fatherOf` relationship with something.

OWLIM's ability to infer new blank nodes during rule evaluation can be used to handle existential quantification. The rule shown in figure 9 is used to model ontologies of the form shown in figure 8. In this example, the variable `b` in the rule head is not bound, so OWLIM will infer a statement containing a new blank node in this position. In combination with the example ontology shown above, this rule will assert that each individual of type `:GrandPa` is a father of some blank node. Note the constraint `x != blank`, which stops the rule firing if `x` is a blank node, preventing a possible infinite, recursive execution, e.g. if the property `p` has a range of `a`, then rule `prp-rng` would infer that the new blank node is of type `a` and rule `exst1` would fire again for this blank node and so on.

```
Id: exst1
  y <owl:onProperty> p
  y <owl:someValuesFrom> <owl:Thing>
  a <rdfs:subClassOf> y
  x <rdf:type> a            [Constraint x != blank]
 ---------------------------
  x p b
```

Fig. 9. An example OWL 2 QL ontology using existential quantification

The other OWL 2 QL constructions were straightforward to model and the implementation passed all W3C tests [2] except for five. Two of these deal with concluding `owl:distinctMembers` for a list of individuals, where for efficiency reasons `owl:differentFrom` pairs are inferred for all combinations of individuals in the list. In two other tests, triples of the form `:a owl:differentFrom :b` are inferred, where in OWLIM `owl:differentFrom` is modelled with a consistency check to prevent an explosion in inferred statements. This can occur whenever two classes are declared `owl:disjointWith` each other and have many members, in which case an `:a owl:differentFrom :b` statement is inferred for each unique pair of members `a` and `b` of the two classes (a Cartesian product).

The last test [3] demonstrates the effect of the comprehension principles in OWL. It states that for every OWL class `C` there is a class constructed from an anonymous union containing only `C`. This is impractical in a forward-chaining environment and its usefulness to the end user is questionable.

4 Conclusions

Rule-sets were created for OWLIM to provide the semantics of the OWL 2 profiles RL and QL. Due to the fact that OWL 2 QL was designed for query rewriting over relational databases, it was discovered that an efficient, scalable implementation using forward-chaining was problematic. While the authors are confident of covering a large part of the semantics, no claim for completeness is made for this profile.

On the other hand, because OWL 2 RL was designed specifically for rule-based systems, the rule-set for this profile was more straightforward to implement, even though it required extensions to the rule-engine to support auxiliary predicates. Using this rule-set, OWLIM is both sound and complete with respect to the semantics of OWL 2 RL/RDF rules, except for the missing support for datatype reasoning. This claim is made based on the 1:1 correspondence between required entailment rules and the implementation in the OWLIM rule-set, and verified using the OWL working group's conformance tests.

References

1. Java programming language. http://www.oracle.com/technetwork/java/index.html.
2. OWL 2 QL Test Cases. http://owl.semanticweb.org/page/Test:QL.
3. OWL 2 Test Case WebOnt–I5.5–005. http://owl.semanticweb.org/page/TestCase:WebOnt-I5.5-005.
4. RIF Working Group. http://www.w3.org/2005/rules/wiki/RIF_Working_Group/.
5. Barry Bishop, Atanas Kiryakov, Damyan Ognyanoff, Ivan Peikov, Zdravko Tashev, and Ruslan Velkov. OWLIM: A family of scalable semantic repositories. *Semantic Web Journal*, 2(1):33–42, 2011.
6. Jeen Broekstra and Arjohn Kampman. SeRQL: A second generation RDF query language. In *SWAD-Europe Workshop on Semantic Web Storage and Retrieval*, page 15, Vrije Universiteit, Amsterdam, Netherlands, November 2003.
7. Jeen Broekstra, Arjohn Kampman, and Frank van Harmelen. Sesame: A generic architecture for storing and querying RDF and RDF Schema. In *International Semantic Web Conference*, Sardinia, Italy, 2002.
8. D. Calvanese, G. de Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Tractable reasoning and efficient query answering in description logics: The DL-Lite family. *Journal of Automated Reasoning*, 39(3):385–429, 2007.
9. Manola F. and E. Miller. RDF Primer, W3C Recommendation. http://www.w3.org/TR/rdf-syntax/, February 2004.
10. A. Kiryakov, D. Ognyanov, and D. Manov. OWLIM – a Pragmatic Semantic Repository for OWL. In *International Workshop on Scalable Semantic Web Knowledge Base Systems*, pages 182–192, New York, USA, 2005.
11. B. Motik, B. Cuenca Grau, I. Horrocks, Z. Wu, A. Fokoue, and C. Lutz. OWL 2 Web Ontology Language Profiles. W3C Recommendation. http://www.w3.org/TR/owl2-profiles/, October 2009.
12. B. Motik, P. Patel-Schneider, and B. Parsia. OWL 2 Web Ontology Language, W3C Recommendation. http://www.w3.org/TR/owl2-syntax/, October 2009.
13. Eric Prud'Hommeaux and Andy Seaborne. SPARQL Query Language for RDF, W3C Recommendation. http://www.w3.org/TR/rdf-sparql-query/, January 2008.
14. Dave Reynolds. OWL 2 RL in RIF, W3C Working Group Note. http://www.w3.org/TR/rif-owl-rl/, June 2010.
15. Michael Schneider. OWL 2 Web Ontology Language: RDF-Based Semantics. http://www.w3.org/TR/owl2-rdf-based-semantics/, October 2099.
16. H. J. ter Horst. Extending the RDFS Entailment Lemma. In *International Semantic Web Conference*, pages 77–91, Hiroshima, Japan, 2004. Springer.
17. H. J. ter Horst. Combining RDF and Part of OWL with Rules: Semantics, Decidability, Complexity. In *International Semantic Web Conference*, pages 668–684, Galway, Ireland, November 2005.
18. J. Ullman. *Principles of Database Systems*, volume 1 of *Principles of Computer Science*. WH Freeman Co., New York, NY, USA, 1983.