W3C Editor's Draft

OWL 2 RL in RIF

W3C Editor's Draft 4 September 2009

This version:
http://www.w3.org/2005/rules/wg/draft/ED-rif-owl-rl-20090904/
Latest editor's draft:
http://www.w3.org/2005/rules/wg/draft/rif-owl-rl/
Editors:
Dave Reynolds, Hewlett Packard Laboratories

This document is also available in these non-normative formats: PDF version.

Abstract

This document shows how OWL 2 RL can be implemented using RIF.

Status of this Document

May Be Superseded

This section describes the status of this document at the time of its publication. Other documents may supersede this document. A list of current W3C publications and the latest revision of this technical report can be found in the W3C technical reports index at http://www.w3.org/TR/.

Set of Documents

This document is being published as one of a set of 10 documents:

1. RIF Core Dialect
2. RIF Basic Logic Dialect
3. RIF Framework for Logic Dialects
4. RIF RDF and OWL Compatibility
5. RIF Datatypes and Built-Ins 1.0

W3C Editor's Draft

6. RIF Production Rule Dialect
7. RIF Use Cases and Requirements
8. RIF Test Cases
9. RIF Combination with XML data
10. OWL 2 RL in RIF (this document)

First Public Working Draft

@@@UPDATE

@@@ Include-in-this-round?

Please Comment By 23 October 2009

The Rule Interchange Format (RIF) Working Group seeks public feedback on this First Public Working Draft. Please send your comments to public-rif-comments@w3.org (public archive). If possible, please offer specific changes to the text that would address your concern. You may also wish to check the Wiki Version of this document and see if the relevant text has already been updated.

No Endorsement

Publication as a Editor's Draft does not imply endorsement by the W3C Membership. This is a draft document and may be updated, replaced or obsoleted by other documents at any time. It is inappropriate to cite this document as other than work in progress.

Patents

This document was produced by a group operating under the 5 February 2004 W3C Patent Policy. The group does not expect this document to become a W3C Recommendation. W3C maintains a public list of any patent disclosures made in connection with the deliverables of the group; that page also includes instructions for disclosing a patent. An individual who has actual knowledge of a patent which the individual believes contains Essential Claim(s) must disclose the information in accordance with section 6 of the W3C Patent Policy.

Table of Contents

W3C Editor's Draft

1 Introduction

OWL 2 RL is a syntactic subset (*profile*) of OWL 2 that is amenable to implementation using rule-based technologies. The OWL 2 Profiles document provides a partial axiomatization of the OWL 2 RDF-Based Semantics in the form of first-order implications that can be used as the basis for such an implementation.

This note describes how that axiomatization can be expressed as a set of RIF rules within the RIF Core dialect.

A complete translation to RIF is not possible due to a mismatch in the supported datatypes between RIF and OWL 2. However, a subset of OWL 2 RL restricted to the RIF datatypes can be implemented via RIF rules.

2 Summary of issues

The rules for the OWL 2 RL semantics can be divided into four (non-disjoint) categories of rules.

Triple pattern rules derive one or more RDF triples from a conjunction of RDF triple patterns. Translation of these to RIF by means of Frame formulae is trivial.

List rules involve processing of RDF lists in the input graph and are expressed in OWL 2 RL using an informal elipsis notation. In implementing OWL 2 RL in RIF 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. We define both options in this note.

Inconsistency rules indicate inconsistencies in the original RDF graph, they are expressed as first order rules which derive a propositional symbol `false`. We suggest a predicate symbol within the RIF namespace be used for this purpose.

Datatype rules these provide type checking and value equality/inequality checking for typed literals across a set of supported datatypes. The set of datatypes supported by RIF and OWL differ significantly. For those OWL 2 datatypes that are supported by RIF translation of the rules to RIF is largely straightforward.

3 Background OWL 2 RL

The OWL 2 RL profile [*OWL 2 Profiles*] is aimed at applications that require scalable reasoning in return for some restriction on expressive power. It defines a syntactic subset of OWL 2 that is amenable to implementation using rule-based technologies together with a partial axiomatization of the OWL 2 RDF-Based Semantics in the form of first-order implications that can be used as the basis for such an implementation.

For ontologies satisfying the syntactic constraints of the OWL 2 RL profile then a suitable rule-based implementation can provide sound and complete entailment checking and conjunctive query answering (so long as the query also meets the OWL 2 RL syntactic restrictions).

The purpose of this note is to provide a translation of the OWL 2 RL rules into RIF notation. The aim is that this translation be faithful to the OWL 2 RL semantics so that the follow theorem should hold:

Theorem 1. Let O_1 and O_2 be OWL 2 RL ontologies satisying the following properties:

- neither O_1 nor O_2 contains a IRI that is used for more than one type of entity (i.e., no IRIs is used both as, say, a class and an individual);
- O_1 does not contain `SubAnnotationPropertyOf`, `AnnotationPropertyDomain`, and `AnnotationPropertyRange` axioms; and
- each axiom in O_2 is an assertion of the form as specified below, for `a`, a_1, ..., a_n named individuals:
  - `ClassAssertion( C a )` where `C` is a class,
  - `ObjectPropertyAssertion( OP` a_1 a_2 `)` where `OP` is an object property,
  - `DataPropertyAssertion( DP a v )` where `DP` is a data property, or
  - `SameIndividual(` a_1 ... a_n `).`

Let `R(`O_1`)` be either the fixed RIF rule set described in Appendix: OWL 2 RL ruleset - presentation syntax or the RIF rule set derived from O_1 by the algorithm described in Appendix: OWL 2 RL to RIF translation.

Furthermore, let *RDF(O_1)* and *RDF(O_2)* be translations of O_1 and O_2, respectively, into RDF graphs as specified in the OWL 2 Mapping to RDF Graphs [*OWL 2 RDF Mapping*].

Then, O_1 entails O_2 under the OWL 2 RDF-Based semantics [*OWL 2 RDF-Based Semantics*] if and only if the RIF-RDF combination of *R(O_1)* and *RDF(O_1)* entails the generalized RDF graph *RDF(O_2)* according to the notion of entailment for RIF-RDF combinations as defined in [*RIF RDF and OWL Compatibility*].

4 Analysis of OWL 2 RL rules

As noted above the OWL 2 RL rules fall into four categories which pose different issues for translation to RIF. We discuss each category in turn.

Note that throughout this section all rules given in RIF Core presentation syntax will assume the following prefix definitions:

```
Document(
  Prefix(rdf  http://www.w3.org/1999/02/22-rdf-syntax-ns#)
  Prefix(rdfs http://www.w3.org/2000/01/rdf-schema#)
  Prefix(owl  http://www.w3.org/2002/07/owl#)
  Prefix(xsd  http://www.w3.org/2001/XMLSchema#)
  Prefix(rif  http://www.w3.org/2007/rif#)
  Prefix(func http://www.w3.org/2007/rif-builtin-function#)
  Prefix(pred http://www.w3.org/2007/rif-builtin-predicate#)
  Prefix(dc   http://purl.org/dc/terms/)
  ...
)
```

4.1 Triple pattern rules

The triple pattern rules take the form:

If	then
$T(s_1, p_1, o_1)$ ... $T(s_n, p_n, o_n)$	$T(sr_1, pr_1, or_1)$ ... $T(sr_m, pr_m, or_m)$

where each argument to the *T* predicate may be a variable, an IRI or literal value.

Such rules can be expressed in the RIF Core presentation syntax as:

```
Group (
  Forall ?v1_1 ... ?v1_o
    ( sr_1[pr_1->or_1] :- And( s_1[p_1->o_1] ...  sn[p_n->o_n] ))
```

```
    ...
    Forall ?vm1 ... ?vmo
      ( srm[prm->orm] :- And( s1[p1->o1] ...  sn[pn->on] ))
  )
```

where ?vi1 ... ?vio are the variables which occur in the ith rule.

4.2 Inconsistency rules

A number of the OWL 2 RL rules detect inconsistencies in the original RDF graph. They express this by deriving a distinguished propositional symbol `false`.

For example:

If	then
T(?p1, owl:propertyDisjointWith, ?p2) T(?x, ?p1, ?y) T(?x, ?p2, ?y)	false

We translate such rules using a distinguished unary predicate `rif:error(msg)`. If this predicate is derived then the original RDF graph is inconsistent under OWL 2 RL semantics.

Note: that the use of a unary rather than nullary predicate is arbitrary. We suggest the unary form to enable rules to report an explanation of the nature of the inconsistency by returning a diagnostic string as the argument to the error predicate.

Thus the above rule would be expressed in RIF Core as:

```
Forall ?p1 ?p2 ?x ?y (
  rif:error("disjoint property violation") :-
    And( ?p1[owl:propertyDisjointWith->?p2] ?x[?p1->?y] ?x[?p2->?y] )
)
```

4.3 List rules

A number of the OWL 2 RL rules involve processing OWL expressions which include RDF lists. Such rules are expressed in OWL 2 RL as templates which, for each possible list length, would correspond to a set of Triple Pattern rules.

There are two ways we can approach this in RIF. We can rewrite such rules as recursive rules that traverse the RDF list links or we can introduce the notion of a preprocessor which takes an OWL2 RL ontology in RDF syntax and translates it to a corresponding RIF rule set in which the templates rules have been instantiated for the lists that actually occur in the source ontology. This is possible because

none of the rules is able to deduce new list entries for such lists within the syntactic constraints of OWL 2 RL.

The approach of using a fixed recursive rule set is appropriate to enable a single RIF OWL 2 RL ruleset to be published. However, the required rules violate the safe-rule checks for RIF Core and are not directly implementable by production rule engines. The translation approach yields rules which are more widely implementable and likely to perform better in practice.

We discuss the direct forms of the List Rules here and then describe in Appendix: OWL 2 RL to RIF translation the translation algorithm for the pre-processor approach.

The are several different patterns of List Rules in the OWL 2 RL ruleset, we discuss each group of rules in turn organized by the pattern involved.

4.3.1 Inconsistent pairs rules

These rules check whether any pair of entries from a list match a certain criterion and if they do an error is signaled (derivation of the `false` propositional symbol).

eq-diff2	T(?y_i, owl:sameAs, ?y_j) T(?x, rdf:type, owl:AllDifferent) LIST[?x, ?y_1, ..., ?y_n]	false	for each 1 ≤ i < j ≤ n
prp-adp	T(?z, rdf:type, owl:AllDisjointProperties) LIST[?z, ?p_1, ..., ?p_n] T(?x, ?p_i, ?y) T(?x, ?p_j, ?y)	false	for each 1 ≤ i < j ≤ n
cax-adc	T(?y, rdf:type, owl:AllDisjointClasses) LIST[?y, ?c_1, ..., ?c_n] T(?x, rdf:type, ?c_i) T(?x, rdf:type, ?c_j)	false	for each 1 ≤ i < j ≤ n

These rules can be directly translated with the assistance of a utility predicate to select any element from an RDF list.

```
(* eq-diff2 *)
Forall ?x ?y ?l (
  rif:error("AllDifferent") :- And (
      ?l[rdf:type -> owl:AllDifferent]
      _member(?l ?x) _member(?l ?y)
      ?x[owl:sameAs->?y] ) )
```

```
(* prp-adp *)
Forall ?x ?y ?p1 ?p2 ?l (
  rif:error("AllDisjointProperties") :- And (
      ?l[rdf:type -> owl:AllDisjointProperties]
      _member(?l ?p1) _member(?l ?p2)
      ?x[?p1->?y ?p2->?y]) )

(* cax-adc *)
Forall ?x ?y ?c1 ?c2 ?l (
  rif:error("AllDisjointClasses") :- And (
      ?l[rdf:type -> owl:AllDisjointClasses]
      _member(?l ?c1) _member(?l ?c2)
      ?x[rdf:type->?c1 rdf:type->?c2]) )

Group (
 Forall ?list ?hd (
   _member(?list ?hd ) :- ?list[rdf:first -> ?hd] )

 Forall ?list ?tl ?x (
   _member( ?list ?x ) :- And( ?list[rdf:rest -> ?tl] _member(?tl ?x) ) )
)
```

4.3.2 Property chain rule

This rule expands the list into a chain of properties that need to be checked in the rule body.

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, $?u_{n+1}$)

Which can be translated by the RIF Core rule set:

```
(* prp-spo2 *)
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] ))
```

When writing logic rules for predicates such as `_checkChain` it would be more common to have the first recursive rule and then a final "terminating" fact:

```
Forall ?start (
  _checkChain(?start rdf:nil ?start) )
```

However, this universal fact is not safe and thus not within RIF Core. We can work around this by the trick of unfolding the final fact into the recursive rule to yield the pair of rules shown above. In this way the rule set is still executable by a forward chaining strategy. Note that such a strategy will generate a large number of irrelevant `_checkChain` assertions since it will apply to any RDF List in the source ontology irrespective of whether that list is part of a `owl:propertyChainAxiom`.

An approach which would give better effective performance for a forward chaining strategy would be to recurse down the property chain "marking" list cells that will need to be checked. Such as:

```
Forall ?p ?pc (
  _needCheckChain(?pc) :- And (
      ?p[owl:propertyChainAxiom->?pc] ))

Forall ?x ?y (
  _needCheckChain(?y) :- And (
      _needCheckChain(?x)
      ?x[rdf:tl->?y] ))

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

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

However, since such optimizations further obscure the rules we will omit them from the rule set described here.

An alternative strategy, if performant OWL 2 RL reasoning is required, would be for a forward chaining engine to provide custom builtins for the list scanning needed here. Detailed description of such an approach is beyond the scope of this note.

We anticipate that an ontology translation approach, such as that described in appendix B, would be used by any practical rule-based OWL 2 RL reasoning system.

4.3.3 Haskey rule

This rule checks that two individuals are the same by virtual of having the same value for each of a list of key properties.

prp-key	T(?c, owl:hasKey, ?u) LIST[?u, ?p_1, ..., ?p_n] T(?x, rdf:type, ?c) T(?x, ?p_1, ?z_1) ... T(?x, ?p_n, ?z_n) T(?y, rdf:type, ?c) T(?y, ?p_1, ?z_1) ... T(?y, ?p_n, ?z_n)	T(?x, owl:sameAs, ?y)

This can be translated by the RIF rule set:

```
(* prp-key *)
Forall ?x ?y ?c ?u ?c (
  ?x[owl:sameAs->?y] :- And (
    ?c[owl:hasKey->?u]  ?x[rdf:type->?c]  ?y[rdf:type->?c]
    _sameKey(?u ?x ?y) ))

Forall ?u ?x ?y (
  _sameKey(?u ?x ?y) :- And (
    ?u[rdf:first->?key rdf:rest->?tl]
    ?x[?key->?v] ?y[?key->?v]
    _sameKey(?tl ?x ?y) ))

Forall ?u ?x ?y (
  _sameKey(?u ?x ?y) :- And (
    ?u[rdf:first->?key rdf:rest->rdf:nil]
    ?x[?key->?v] ?y[?key->?v] ))
```

Similar considerations about safety and performance apply here as in the previous section.

4.3.4 Forward intersectionOf rule

This pattern involves a test for all members of a list.

cls-int1	T(?c, owl:intersectionOf, ?x) LIST[?x, ?c_1, ..., ?c_n] T(?y, rdf:type, ?c_1) T(?y, rdf:type, ?c_2) ... T(?y, rdf:type, ?c_n)	T(?y, rdf:type, ?c)

This can be translated by the RIF rule set:

```
(* cls-int1 *)
Forall ?y ?c ?l (
  ?y[rdf:type->?c] :- And (
    ?c[owl:intersectionOf->?l]
    _allTypes(?l ?y) ))

Forall ?l ?y ?ty ?tl (
  _allTypes(?l ?y) :- And (
    ?l[rdf:first->?ty rdf:rest->?tl]
    ?y[rdf:type->?ty]
    _allTypes(?tl ?y) ))

Forall ?l ?y ?ty (
  _allTypes(?l ?y) :- And (
    ?l[rdf:first->?ty rdf:rest->rdf:nil]
    ?y[rdf:type->?ty] ))
```

Similar considerations on safety and performance apply here as in the previous two sections.

4.3.5 Simple member rules

This pattern involves a test for each member of a list.

cls-uni	T(?c, owl:unionOf, ?x) LIST[?x, ?c_1, ..., ?c_n] T(?y, rdf:type, ?c_i)	T(?y, rdf:type, ?c)	for each 1 ≤ i ≤ n

cls-oo	T(?c, owl:oneOf, ?x) LIST[?x, ?y_1, ..., ?y_n]	T(?y_i, rdf:type, ?c)	for each 1 ≤ i ≤ n
cls-int2	T(?c, owl:intersectionOf, ?x) LIST[?x, ?c_1, ..., ?c_n] T(?y, rdf:type, ?c)	T(?y, rdf:type, ?c_1) T(?y, rdf:type, ?c_2) ... T(?y, rdf:type, ?c_n)	
scm-int	T(?c, owl:intersectionOf, ?x) LIST[?x, ?c_1, ..., ?c_n]	T(?c, rdfs:subClassOf, ?c_1) T(?c, rdfs:subClassOf, ?c_2) ... T(?c, rdfs:subClassOf, ?c_n)	
scm-uni	T(?c, owl:unionOf, ?x) LIST[?x, ?c_1, ..., ?c_n]	T(?c_1, rdfs:subClassOf, ?c) T(?c_2, rdfs:subClassOf, ?c) ... T(?c_n, rdfs:subClassOf, ?c)	

These can be translated by the RIF rule set:

```
(* cls-uni *)
Forall ?y ?c ?l ?ci (
  ?y[rdf:type->?c] :- And (
    ?c[owl:unionOf->?l]
    _member(?l ?ci)
    ?y[rdf:type->?ci] ))

(* cls-oo *)
Forall ?yi ?c ?l (
  ?yi[rdf:type->?c] :- And (
    ?c[owl:oneOf->?l]
    _member(?l ?yi) ))

(* cls-int2 *)
Forall ?y ?c ?ci ?l (
  ?y[rdf:type->?ci] :- And (
    ?c[owl:intersectionOf->?l]
    _member(?l ?ci)
    ?y[rdf:type->?c] ))

(* scm-int *)
Forall ?c ?ci ?l (
  ?c[rdfs:subClassOf->?ci] :- And (
```

```
        ?c[owl:intersectionOf->?l]
        _member(?l ?ci) ))

  (* scm-uni *)
  Forall ?c ?ci ?l (
    ?ci[rdfs:subClassOf->?c] :- And (
      ?c[owl:unionOf->?l]
      _member(?l ?ci) ))
```

where the `_member` predicate was defined earlier.

4.4 Datatype rules

OWL 2 RL specifies five groups of rules to capture the semantics of supported datatypes. These are:

If	then		
dt-type1	`true`	`T(df, rdf:type, rdfs:Datatype)`	for each datatype `dt` supported in OWL 2 RL
dt-type2	`true`	`T(lt, rdf:type, dt)`	for each literal `lt` and each datatype `dt` supported in OWL 2 RL such that the data value of `lt` is contained in the value space of `dt`
dt-eq	`true`	`T(lt1, owl:sameAs, lt2)`	for all literals lt_1 and lt_2 with the same data value
dt-diff	`true`	`T(lt1, owl:differentFrom, lt2)`	for all literals lt_1 and lt_2 with different data values
dt-not-type	`T(lt, rdf:type, dt)`	`false`	for each literal `lt` and each datatype `dt` supported in OWL 2 RL such that the data value of `lt` is not contained in the value space of `dt`

4.4.1 Datatypes supported

The RIF and OWL 2 RL working groups have now aligned the set of datatypes so that all the datatypes required by OWL 2 RL are supported by RIF.

4.4.2 Datatype rules and safety

A naieve translation of the datatype rules would require unsafe RIF rules. For example, one might be tempted to translate `dt-eq` as:

```
Forall ?l1 ?l2 ( ?l1[owl:sameAs->l2] ) :- ?l1 = ?l2 )
```

To ensure safety we must ground the variables in the relevant vocabulary. Referring to Theorem-1 we can see that the entailment checks supported in O_2 are limited and do not include direct checks for inequality, type or non-type of literals. Thus we only need to ground the datatype rules in the literal vocabulary of O_1. While O_2 can include axioms of the form `DataPropertyAssertion( DP a v )` since RIF and OWL 2 have the same notion of identity over those datatypes which both support then such direct equality entailments are automatically satisfied.

Thus a sufficient rendering of `dt-eq` would be:

```
Forall ?l1 ?l2 ?s1 ?s2 ?p1 ?p2
 ( ?l1[owl:sameAs->l2] ) :- And(
          ?s1[?p1->?l1] ?s2[?p2->?l2] ?l1 = ?l2 ))
```

However, we can go further than this. Since the entailment checks are limited to only indirect consequences of these rules then we can fold the datatype rules directly into the rules which include owl:sameAs in their premise. This avoids the generation of a quadratic number of `owl:sameAs` and `owl:differentFrom` assertions for all literals.

4.4.3 Rules for supported datatypes

The RIF representation of the data type rules for supported datatypes is reasonably straightforward.

Type membership

```
(* dt-type2 *)
Forall ?s ?p ?lt ( ?lt[rdf:type->?ty rdf:type->rdfs:Literal]
                    :- And( ?s[?p->?lt] External( pred:isLiteralOfType(?l
```

Type checking

```
(* dt-not-type *)
Forall ?lt ?dt (
```

```
    rif:error('datatype violation') :- And (
      ?lt[rdf:type->dt] External(pred:isLiteralNotOfType( ?lt ?dt )) ))
```

Equality and inequality As noted above we can fold the literal equality and inequality checks into the rules which refer to them, in this case the only affected rule is `eq-diff1`:

```
(* eq-diff1 *)
Forall ?x ?y (
    rif:error("inconsistent") :- And(
        ?x[owl:sameAs->?y]
        ?x[owl:differentFrom->?y]  ))
```

So after folding the grounded literal rules for `dt-eq` and `dt-diff` in we arrive at:

```
(* eq-diff1-literal1 *)
Forall ?x ?y ?s1 ?s2 ?p1 ?p2 (
    rif:error("inconsistent") :- And(
        ?s1[?p1->?x]  ?s2[?p2->?y]
        ?x[owl:sameAs->?y]
        External(pred:literalNotIdentical(?x ?y))  ))

(* eq-diff1-literal2 *)
Forall ?x ?y ?s1 ?s2 ?p1 ?p2 (
    rif:error("inconsistent") :- And(
        ?s1[?p1->?x]  ?s2[?p2->?y]
        ?x = ?y
        ?x[owl:differentFrom->?y]  ))
```

Fixed vocabulary rules These are trivial asserted facts of the form:

```
(* dt-type1-text *) Forall ( rdfs:text[rdf:type -> rdfs:Datatype] ) etc
```

4.4.4 Partial support for subtypes

OWL 2 RL includes support for several subtypes of *xsd:string* and of *xsd:decimal* which are not directly supported by RIF.

A RIF Core implementation is normally required to reject rule sets which use datatypes outside of the set supported by RIF but is permitted to support a *run-with-extensions-mode* to turn this off.

Such an engine could support OWL 2 RL reasoning over the full set of OWL 2 RL datatypes without requiring any additional builtins to support those types, we can use the RIF arithmetic and string matching primitives to perform the sub-range checking required.

As an example of this approach we illustrate the implementation of *xsd:nonNegativeInteger*.

```
(* dt-type2 *)
Forall ?s ?p  ( ?lt[rdf:type->xsd:nonNegativeInteger rdf:type->rdfs:Liter
                    :- And( ?s[?p->?lt] isNonNegativeInteger(?lt)) ))

(* dt-not-type-nonNegativeInteger *)
Forall ?lt ?dt (
  rif:error('datatype violation') - And (
    ?lt[rdf:type->xsd:nonNegativeInteger]
       isNotNonNegativeInteger(?lt) )

(* dt-eq-nonNegativeInteger *)
Forall ?l1 ?l2 ( ?l1[owl:sameAs->l2] ) :- And (
  isNonNegativeInteger( ?l1 )
  isNonNegativeInteger( ?l2 )
  External(pred:numeric-equal(  External(func:xsd:integer(?l1))
                                External(func:xsd:integer(?l2)) )) ))

(* dt-diff-nonNegativeInteger1 *)
Forall ?lt1 ?lt2 ( ?lt1[owl:differentFrom->?lt2] :- And (
  isNonNegativeInteger(?lt1) isNotNonNegativeInteger(?lt2) ))
(* dt-diff-nonNegativeInteger2 *)
Forall ?lt1 ?lt2 ( ?lt1[owl:differentFrom->?lt2] :- And (
  isNotNonNegativeInteger(?lt1) isNonNegativeInteger(?lt2) ))
(* dt-diff-nonNegativeInteger3 *)
Forall ?lt1 ?lt2 ( ?lt1[owl:differentFrom->?lt2] :- And (
  isNonNegativeInteger(?lt1) isNonNegativeInteger(?lt2)
  External(pred:numeric-not-equal(  External(func:xsd:integer(?l1))
                                    External(func:xsd:integer(?l2)) )) ))

(* isNonNegativeInteger *)
Forall ?lt (
  isNonNegativeInteger( ?lt ) :-  And (
    External(func:isInteger( ?lt ))
    External(pred:numeric-greater-than-or-equal(
           External(func:xsd:integer( ?lt )) 0 )) ))

(* isNotNonNegativeInteger1 *)
Forall ?lt (
  isNotNonNegativeInteger( ?lt ) :- And (
    External(func:isInteger( ?lt ))
    External(pred:numeric-less-than(
           External(func:xsd:integer( ?lt )) 0 )) ))

(* isNotNonNegativeInteger2 *)
Forall ?lt (
```

```
    isNotNonNegativeInteger( ?lt ) :-
      External(func:isNotInteger( ?lt )) )
```

5 References

[OWL 2 Profiles]
OWL 2 Web Ontology Language Profiles, Grau et al eds, 2008.
[OWL 2 RDF Mapping]
OWL 2 Web Ontology Language: Mapping to RDF Graphs. Bernardo Cuenca Grau and Boris Motik, eds., 2008.
[OWL 2 Semantics]
OWL 2 Web Ontology Language: Model-Theoretic Semantics. Bernardo Cuenca Grau and Boris Motik, eds., 2008.
[RIF RDF and OWL Compatibility]
RIF RDF and OWL Compatibility Jos de Bruijn, editor. W3C Editor's Draft, 4 September 2009, http://www.w3.org/2005/rules/wg/draft/ED-rif-rdf-owl-20090904/. Latest version available at http://www.w3.org/2005/rules/wg/draft/rif-rdf-owl/.

6 Appendix: OWL 2 RL ruleset - presentation syntax

In this appendix we provide a RIF Core presentation syntax translation of the complete OWL 2 RL ruleset. We divide these rules into three documents - the simple triple rules (mechanically translated from the OWL 2 RL rule format), the List rules and the datatype rules.

Editor's Note: The manual rules (and indeed the manually written mechanical translator) may include syntactic errors, the intent is to verify them once we have a working PS-to-XML translator.

Simple triple rules

```
Document (
 Prefix(rdf  http://www.w3.org/1999/02/22-rdf-syntax-ns#)
 Prefix(rdfs http://www.w3.org/2000/01/rdf-schema#)
 Prefix(owl  http://www.w3.org/2002/07/owl#)
 Prefix(xsd  http://www.w3.org/2001/XMLSchema#)
 Prefix(rif  http://www.w3.org/2007/rif#)
 Prefix(func http://www.w3.org/2007/rif-builtin-function#)
 Prefix(pred http://www.w3.org/2007/rif-builtin-predicate#)
 Prefix(dc   http://purl.org/dc/terms/)
 Group (

(* eq-ref *)
Forall ?p ?o ?s (
   ?s[owl:sameAs->?s] :- ?s[?p->?o])
```

```
(* eq-ref1 *)
Forall ?p ?o ?s (
   ?p[owl:sameAs->?p] :- ?s[?p->?o])

(* eq-ref2 *)
Forall ?p ?o ?s (
   ?o[owl:sameAs->?o] :- ?s[?p->?o])

(* eq-sym *)
Forall ?x ?y (
   ?y[owl:sameAs->?x] :- ?x[owl:sameAs->?y])

(* eq-trans *)
Forall ?x ?z ?y (
   ?x[owl:sameAs->?z] :- And(
       ?x[owl:sameAs->?y]
       ?y[owl:sameAs->?z]  ))

(* eq-rep-s *)
Forall ?p ?o ?s ?s2 (
   ?s2[?p->?o] :- And(
       ?s[owl:sameAs->?s2]
       ?s[?p->?o]  ))

(* eq-rep-p *)
Forall ?p ?o ?s ?p2 (
   ?s[?p2->?o] :- And(
       ?p[owl:sameAs->?p2]
       ?s[?p->?o]  ))

(* eq-rep-o *)
Forall ?p ?o ?s ?o2 (
   ?s[?p->?o2] :- And(
       ?o[owl:sameAs->?o2]
       ?s[?p->?o]  ))

(* eq-diff1 *)
Forall ?x ?y (
   rif:error("inconsistent") :- And(
       ?x[owl:sameAs->?y]
       ?x[owl:differentFrom->?y]  ))

(* prp-ap-label *)
Forall (
   rdfs:label[rdf:type->owl:AnnotationProperty])

(* prp-ap-comment *)
Forall (
   rdfs:comment[rdf:type->owl:AnnotationProperty])
```

```
(* prp-ap-seeAlso *)
Forall (
   rdfs:seeAlso[rdf:type->owl:AnnotationProperty])

(* prp-ap-isDefinedBy *)
Forall (
   rdfs:isDefinedBy[rdf:type->owl:AnnotationProperty])

(* prp-ap-deprecated *)
Forall (
   owl:deprecated[rdf:type->owl:AnnotationProperty])

(* prp-ap-priorVersion *)
Forall (
   owl:priorVersion[rdf:type->owl:AnnotationProperty])

(* prp-ap-backwardCompatibleWith *)
Forall (
   owl:backwardCompatibleWith[rdf:type->owl:AnnotationProperty])

(* prp-ap-incompatibleWith *)
Forall (
   owl:incompatibleWith[rdf:type->owl:AnnotationProperty])

(* prp-dom *)
Forall ?p ?c ?x ?y (
   ?x[rdf:type->?c] :- And(
       ?p[rdfs:domain->?c]
       ?x[?p->?y]  ))

(* prp-rng *)
Forall ?p ?c ?x ?y (
   ?y[rdf:type->?c] :- And(
       ?p[rdfs:range->?c]
       ?x[?p->?y]  ))

(* prp-fp *)
Forall ?p ?y2 ?x ?y1 (
   ?y1[owl:sameAs->?y2] :- And(
       ?p[rdf:type->owl:FunctionalProperty]
       ?x[?p->?y1]
       ?x[?p->?y2]  ))

(* prp-ifp *)
Forall ?p ?x1 ?x2 ?y (
   ?x1[owl:sameAs->?x2] :- And(
       ?p[rdf:type->owl:InverseFunctionalProperty]
       ?x1[?p->?y]
       ?x2[?p->?y]  ))
```

```
(* prp-irp *)
Forall ?p ?x (
   rif:error("inconsistent") :- And(
       ?p[rdf:type->owl:IrreflexiveProperty]
       ?x[?p->?x]  ))

(* prp-symp *)
Forall ?p ?x ?y (
   ?y[?p->?x] :- And(
       ?p[rdf:type->owl:SymmetricProperty]
       ?x[?p->?y]  ))

(* prp-asyp *)
Forall ?p ?x ?y (
   rif:error("inconsistent") :- And(
       ?p[rdf:type->owl:AsymmetricProperty]
       ?x[?p->?y]
       ?y[?p->?x]  ))

(* prp-trp *)
Forall ?p ?x ?z ?y (
   ?x[?p->?z] :- And(
       ?p[rdf:type->owl:TransitiveProperty]
       ?x[?p->?y]
       ?y[?p->?z]  ))

(* prp-spo1 *)
Forall ?x ?y ?p2 ?p1 (
   ?x[?p2->?y] :- And(
       ?p1[rdfs:subPropertyOf->?p2]
       ?x[?p1->?y]  ))

(* prp-eqp1 *)
Forall ?x ?y ?p2 ?p1 (
   ?x[?p2->?y] :- And(
       ?p1[owl:equivalentProperty->?p2]
       ?x[?p1->?y]  ))

(* prp-eqp2 *)
Forall ?x ?y ?p2 ?p1 (
   ?x[?p1->?y] :- And(
       ?p1[owl:equivalentProperty->?p2]
       ?x[?p2->?y]  ))

(* prp-pdw *)
Forall ?x ?y ?p2 ?p1 (
   rif:error("inconsistent") :- And(
       ?p1[owl:propertyDisjointWith->?p2]
```

```
        ?x[?p1->?y]
        ?x[?p2->?y]  ))

(* prp-inv1 *)
Forall ?x ?y ?p2 ?p1 (
   ?y[?p2->?x] :- And(
        ?p1[owl:inverseOf->?p2]
        ?x[?p1->?y]  ))

(* prp-inv2 *)
Forall ?x ?y ?p2 ?p1 (
   ?y[?p1->?x] :- And(
        ?p1[owl:inverseOf->?p2]
        ?x[?p2->?y]  ))

(* cls-thing *)
Forall (
   owl:Thing[rdf:type->owl:Class])

(* cls-nothing1 *)
Forall (
   owl:Nothing[rdf:type->owl:Class])

(* cls-nothing2 *)
Forall ?x (
   rif:error("inconsistent") :- ?x[rdf:type->owl:Nothing])

(* cls-svf1 *)
Forall ?p ?v ?u ?x ?y (
   ?u[rdf:type->?x] :- And(
        ?x[owl:someValuesFrom->?y]
        ?x[owl:onProperty->?p]
        ?u[?p->?v]
        ?v[rdf:type->?y]  ))

(* cls-svf2 *)
Forall ?p ?v ?u ?x (
   ?u[rdf:type->?x] :- And(
        ?x[owl:someValuesFrom->owl:Thing]
        ?x[owl:onProperty->?p]
        ?u[?p->?v]  ))

(* cls-avf *)
Forall ?p ?v ?u ?x ?y (
   ?v[rdf:type->?y] :- And(
        ?x[owl:allValuesFrom->?y]
        ?x[owl:onProperty->?p]
        ?u[rdf:type->?x]
        ?u[?p->?v]  ))
```

```
(* cls-hv1 *)
Forall ?p ?u ?x ?y (
   ?u[?p->?y] :- And(
       ?x[owl:hasValue->?y]
       ?x[owl:onProperty->?p]
       ?u[rdf:type->?x]  ))

(* cls-hv2 *)
Forall ?p ?u ?x ?y (
   ?u[rdf:type->?x] :- And(
       ?x[owl:hasValue->?y]
       ?x[owl:onProperty->?p]
       ?u[?p->?y]  ))

(* cls-maxc1 *)
Forall ?p ?u ?x ?y (
   rif:error("inconsistent") :- And(
       ?x[owl:maxCardinality->0]
       ?x[owl:onProperty->?p]
       ?u[rdf:type->?x]
       ?u[?p->?y]  ))

(* cls-maxc2 *)
Forall ?p ?y2 ?u ?x ?y1 (
   ?y1[owl:sameAs->?y2] :- And(
       ?x[owl:maxCardinality->1]
       ?x[owl:onProperty->?p]
       ?u[rdf:type->?x]
       ?u[?p->?y1]
       ?u[?p->?y2]  ))

(* cls-maxqc1 *)
Forall ?p ?c ?u ?x ?y (
   rif:error("inconsistent") :- And(
       ?x[owl:maxQualifiedCardinality->0]
       ?x[owl:onProperty->?p]
       ?x[owl:onClass->?c]
       ?u[rdf:type->?x]
       ?u[?p->?y]
       ?y[rdf:type->?c]  ))

(* cls-maxqc2 *)
Forall ?p ?u ?x ?y (
   rif:error("inconsistent") :- And(
       ?x[owl:maxQualifiedCardinality->0]
       ?x[owl:onProperty->?p]
       ?x[owl:onClass->owl:Thing]
       ?u[rdf:type->?x]
       ?u[?p->?y]  ))
```

```
(* cls-maxqc3 *)
Forall ?p ?y2 ?c ?u ?x ?y1 (
   ?y1[owl:sameAs->?y2] :- And(
       ?x[owl:maxQualifiedCardinality->1]
       ?x[owl:onProperty->?p]
       ?x[owl:onClass->?c]
       ?u[rdf:type->?x]
       ?u[?p->?y1]
       ?y1[rdf:type->?c]
       ?u[?p->?y2]
       ?y2[rdf:type->?c]  ))

(* cls-maxqc4 *)
Forall ?p ?y2 ?u ?x ?y1 (
   ?y1[owl:sameAs->?y2] :- And(
       ?x[owl:maxQualifiedCardinality->1]
       ?x[owl:onProperty->?p]
       ?x[owl:onClass->owl:Thing]
       ?u[rdf:type->?x]
       ?u[?p->?y1]
       ?u[?p->?y2]  ))

(* cax-sco *)
Forall ?x ?c1 ?c2 (
   ?x[rdf:type->?c2] :- And(
       ?c1[rdfs:subClassOf->?c2]
       ?x[rdf:type->?c1]  ))

(* cax-eqc1 *)
Forall ?x ?c1 ?c2 (
   ?x[rdf:type->?c2] :- And(
       ?c1[owl:equivalentClass->?c2]
       ?x[rdf:type->?c1]  ))

(* cax-eqc2 *)
Forall ?x ?c1 ?c2 (
   ?x[rdf:type->?c1] :- And(
       ?c1[owl:equivalentClass->?c2]
       ?x[rdf:type->?c2]  ))

(* cax-dw *)
Forall ?x ?c1 ?c2 (
   rif:error("inconsistent") :- And(
       ?c1[owl:disjointWith->?c2]
       ?x[rdf:type->?c1]
       ?x[rdf:type->?c2]  ))
```

```
(* scm-cls *)
Forall ?c (
   ?c[rdfs:subClassOf->?c] :- ?c[rdf:type->owl:Class])

(* scm-cls1 *)
Forall ?c (
   ?c[owl:equivalentClass->?c] :- ?c[rdf:type->owl:Class])

(* scm-cls2 *)
Forall ?c (
   ?c[rdfs:subClassOf->owl:Thing] :- ?c[rdf:type->owl:Class])

(* scm-cls3 *)
Forall ?c (
   owl:Nothing[rdfs:subClassOf->?c] :- ?c[rdf:type->owl:Class])

(* scm-sco *)
Forall ?c1 ?c2 ?c3 (
   ?c1[rdfs:subClassOf->?c3] :- And(
       ?c1[rdfs:subClassOf->?c2]
       ?c2[rdfs:subClassOf->?c3]  ))

(* scm-eqc1 *)
Forall ?c1 ?c2 (
   ?c1[rdfs:subClassOf->?c2] :- ?c1[owl:equivalentClass->?c2])

(* scm-eqc11 *)
Forall ?c1 ?c2 (
   ?c2[rdfs:subClassOf->?c1] :- ?c1[owl:equivalentClass->?c2])

(* scm-eqc2 *)
Forall ?c1 ?c2 (
   ?c1[owl:equivalentClass->?c2] :- And(
       ?c1[rdfs:subClassOf->?c2]
       ?c2[rdfs:subClassOf->?c1]  ))

(* scm-op *)
Forall ?p (
   ?p[rdfs:subPropertyOf->?p] :- ?p[rdf:type->owl:ObjectProperty])

(* scm-op1 *)
Forall ?p (
   ?p[owl:equivalentProperty->?p] :- ?p[rdf:type->owl:ObjectProperty])

(* scm-dp *)
Forall ?p (
   ?p[rdfs:subPropertyOf->?p] :- ?p[rdf:type->owl:DatatypeProperty])
```

```
(* scm-dp1 *)
Forall ?p (
   ?p[owl:equivalentProperty->?p] :- ?p[rdf:type->owl:DatatypeProperty])

(* scm-spo *)
Forall ?p3 ?p2 ?p1 (
   ?p1[rdfs:subPropertyOf->?p3] :- And(
       ?p1[rdfs:subPropertyOf->?p2]
       ?p2[rdfs:subPropertyOf->?p3]  ))

(* scm-eqp1 *)
Forall ?p2 ?p1 (
   ?p1[rdfs:subPropertyOf->?p2] :- ?p1[owl:equivalentProperty->?p2])

(* scm-eqp11 *)
Forall ?p2 ?p1 (
   ?p2[rdfs:subPropertyOf->?p1] :- ?p1[owl:equivalentProperty->?p2])

(* scm-eqp2 *)
Forall ?p2 ?p1 (
   ?p1[owl:equivalentProperty->?p2] :- And(
       ?p1[rdfs:subPropertyOf->?p2]
       ?p2[rdfs:subPropertyOf->?p1]  ))

(* scm-dom1 *)
Forall ?p ?c1 ?c2 (
   ?p[rdfs:domain->?c2] :- And(
       ?p[rdfs:domain->?c1]
       ?c1[rdfs:subClassOf->?c2]  ))

(* scm-dom2 *)
Forall ?c ?p2 ?p1 (
   ?p1[rdfs:domain->?c] :- And(
       ?p2[rdfs:domain->?c]
       ?p1[rdfs:subPropertyOf->?p2]  ))

(* scm-rng1 *)
Forall ?p ?c1 ?c2 (
   ?p[rdfs:range->?c2] :- And(
       ?p[rdfs:range->?c1]
       ?c1[rdfs:subClassOf->?c2]  ))

(* scm-rng2 *)
Forall ?c ?p2 ?p1 (
   ?p1[rdfs:range->?c] :- And(
       ?p2[rdfs:range->?c]
       ?p1[rdfs:subPropertyOf->?p2]  ))
```

```
(* scm-hv *)
Forall ?c1 ?c2 ?i ?p2 ?p1 (
   ?c1[rdfs:subClassOf->?c2] :- And(
       ?c1[owl:hasValue->?i]
       ?c1[owl:onProperty->?p1]
       ?c2[owl:hasValue->?i]
       ?c2[owl:onProperty->?p2]
       ?p1[rdfs:subPropertyOf->?p2]  ))

(* scm-svf1 *)
Forall ?p ?y2 ?c1 ?c2 ?y1 (
   ?c1[rdfs:subClassOf->?c2] :- And(
       ?c1[owl:someValuesFrom->?y1]
       ?c1[owl:onProperty->?p]
       ?c2[owl:someValuesFrom->?y2]
       ?c2[owl:onProperty->?p]
       ?y1[rdfs:subClassOf->?y2]  ))

(* scm-svf2 *)
Forall ?c1 ?c2 ?y ?p2 ?p1 (
   ?c1[rdfs:subClassOf->?c2] :- And(
       ?c1[owl:someValuesFrom->?y]
       ?c1[owl:onProperty->?p1]
       ?c2[owl:someValuesFrom->?y]
       ?c2[owl:onProperty->?p2]
       ?p1[rdfs:subPropertyOf->?p2]  ))

(* scm-avf1 *)
Forall ?p ?y2 ?c1 ?c2 ?y1 (
   ?c1[rdfs:subClassOf->?c2] :- And(
       ?c1[owl:allValuesFrom->?y1]
       ?c1[owl:onProperty->?p]
       ?c2[owl:allValuesFrom->?y2]
       ?c2[owl:onProperty->?p]
       ?y1[rdfs:subClassOf->?y2]  ))

(* scm-avf2 *)
Forall ?c1 ?c2 ?y ?p2 ?p1 (
   ?c2[rdfs:subClassOf->?c1] :- And(
       ?c1[owl:allValuesFrom->?y]
       ?c1[owl:onProperty->?p1]
       ?c2[owl:allValuesFrom->?y]
       ?c2[owl:onProperty->?p2]
       ?p1[rdfs:subPropertyOf->?p2]  ))

(* prp-npa1 *)
Forall ?x ?i1 ?p ?i2 (
    rif:error("inconsistent") :- And(
       ?x[owl:sourceIndividual->?i1]
```

```
        ?x[owl:assertionProperty->?p]
        ?x[owl:targetIndividual->?i2]
        ?i1[?p->?i2] ))

(* prp-npa2 *)
Forall ?x ?i ?p ?lt (
    rif:error("inconsistent") :- And(
        ?x[owl:sourceIndividual->?i]
        ?x[owl:assertionProperty->?p]
        ?x[owl:targetValue->?lt]
        ?i[?p->?lt] ))

(* cax-dw *)
Forall ?c1 ?c2 ?x (
    rif:error("inconsistent") :- And(
        ?c1[owl:disjointWith->?c2]
        ?x[rdf:type->?c1]
        ?x[rdf:type->?c2] ))

(* cls-com *)
Forall ?c1 ?c2 ?x (
    rif:error("inconsistent") :- And(
        ?c1[owl:complementOf->?c2]
        ?x[rdf:type->?c1]
        ?x[rdf:type->?c2] ))

))
```

List rules

```
Document (
 Prefix(rdf  http://www.w3.org/1999/02/22-rdf-syntax-ns#)
 Prefix(rdfs http://www.w3.org/2000/01/rdf-schema#)
 Prefix(owl  http://www.w3.org/2002/07/owl#)
 Prefix(xsd  http://www.w3.org/2001/XMLSchema#)
 Prefix(rif  http://www.w3.org/2007/rif#)
 Prefix(func http://www.w3.org/2007/rif-builtin-function#)
 Prefix(pred http://www.w3.org/2007/rif-builtin-predicate#)
 Prefix(dc   http://purl.org/dc/terms/)
 Group (

 (* eq-diff2 *)
 Forall ?x ?y ?l ?a (
   rif:error("AllDifferent") :- And (
       ?a[rdf:type -> owl:AllDifferent]
       ?a[owl:distinctMembers->?l]
       _member(?l ?x) _member(?l ?y)
       ?x[owl:sameAs->?y] ) )
```

```
(* eq-diff3 *)
Forall ?x ?y ?l ?a (
  rif:error("AllDifferent") :- And (
      ?a[rdf:type -> owl:AllDifferent]
      ?a[owl:members->?l]
      _member(?l ?x) _member(?l ?y)
      ?x[owl:sameAs->?y] ) )

(* prp-adp *)
Forall ?x ?y ?p1 ?p2 ?l (
  rif:error("AllDisjointProperties") :- And (
      ?l[rdf:type -> owl:AllDisjointProperties]
      _member(?l ?p1) _member(?l ?p2)
      ?x[?p1->?y ?p2->?y]) )

(* cax-adc *)
Forall ?x ?y ?c1 ?c2 ?l (
  rif:error("AllDisjointClasses") :- And (
      ?l[rdf:type -> owl:AllDisjointClasses]
      _member(?l ?c1) _member(?l ?c2)
      ?x[rdf:type->?c1 rdf:type->?c2]) )

Group (
 Forall ?list ?hd (
   _member(?list ?hd ) :- ?list[rdf:first -> ?hd] )

 Forall ?list ?tl ?x (
   _member( ?list ?x ) :- And( ?list[rdf:rest -> ?tl] _member(?tl ?x) ) )
)

(* prp-spo2 *)
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] ))

  Forall ?x ?y ?c ?u ?c (
  ?x[owl:sameAs->?y] :- And (
```

```
    ?c[owl:hasKey->?u]  ?x[rdf:type->?c]  ?y[rdf:type->?c]
    _sameKey(?u ?x ?y) ))

Forall ?u ?x ?y (
  _sameKey(?u ?x ?y) :- And (
    ?u[rdf:first->?key rdf:rest->?tl]
    ?x[?key->?v] ?y[?key->?v]
    _sameKey(?tl ?x ?y) ))

Forall ?u ?x ?y (
  _sameKey(?u ?x ?y) :- And (
    ?u[rdf:first->?key rdf:rest->rdf:nil]
    ?x[?key->?v] ?y[?key->?v] ))

(* cls-int1 *)
Forall ?y ?c ?l (
  ?y[rdf:type->?c] :- And (
    ?c[owl:intersectionOf->?l]
    _allTypes(?l ?y) ))

Forall ?l ?y ?ty ?tl (
  _allTypes(?l ?y) :- And (
    ?l[rdf:first->?ty rdf:rest->?tl]
    ?y[rdf:type->?ty]
    _allTypes(?tl ?y) ))

Forall ?l ?y ?ty (
  _allTypes(?l ?y) :- And (
    ?l[rdf:first->?ty rdf:rest->rdf:nil]
    ?y[rdf:type->?ty] ))

(* prp-key *)
Forall ?x ?y ?c ?u ?c (
  ?x[owl:sameAs->?y] :- And (
    ?c[owl:hasKey->?u]  ?x[rdf:type->?c]  ?y[rdf:type->?c]
    _sameKey(?u ?x ?y) ))

Forall ?u ?x ?y (
  _sameKey(?u ?x ?y) :- And (
    ?u[rdf:first->?key rdf:rest->?tl]
    ?x[?key->?v] ?y[?key->?v]
    _sameKey(?tl ?x ?y) ))

Forall ?u ?x ?y (
  _sameKey(?u ?x ?y) :- And (
    ?u[rdf:first->?key rdf:rest->rdf:nil]
    ?x[?key->?v] ?y[?key->?v] ))

(* cls-uni *)
```

```
Forall ?y ?c ?l ?ci (
  ?y[rdf:type->?c] :- And (
    ?c[owl:unionOf->?l]
    _member(?l ?ci)
    ?y[rdf:type->?ci] ))

(* cls-oo *)
Forall ?yi ?c ?l (
  ?yi[rdf:type->?c] :- And (
    ?c[owl:oneOf->?l]
    _member(?l ?yi) ))

(* cls-int2 *)
Forall ?y ?c ?ci ?l (
  ?y[rdf:type->?ci] :- And (
    ?c[owl:intersectionOf->?l]
    _member(?l ?ci)
    ?y[rdf:type->?c] ))

(* scm-int *)
Forall ?c ?ci ?l (
  ?c[rdfs:subClassOf->?ci] :- And (
    ?c[owl:intersectionOf->?l]
    _member(?l ?ci) ))

(* scm-uni *)
Forall ?c ?ci ?l (
  ?ci[rdfs:subClassOf->?c] :- And (
    ?c[owl:unionOf->?l]
    _member(?l ?ci) ))

))
```

Datatype rules

```
Document (
 Prefix(rdf  http://www.w3.org/1999/02/22-rdf-syntax-ns#)
 Prefix(rdfs http://www.w3.org/2000/01/rdf-schema#)
 Prefix(owl  http://www.w3.org/2002/07/owl#)
 Prefix(xsd  http://www.w3.org/2001/XMLSchema#)
 Prefix(rif  http://www.w3.org/2007/rif#)
 Prefix(func http://www.w3.org/2007/rif-builtin-function#)
 Prefix(pred http://www.w3.org/2007/rif-builtin-predicate#)
 Prefix(dc   http://purl.org/dc/terms/)
 Group (
```

```
(* dt-type2 *)
Forall ?s ?p ?lt ( ?lt[rdf:type->?ty rdf:type->rdfs:Literal]
                     :- And( ?s[?p->?lt] External( pred:isLiteralOfType(?lt

(* dt-not-type *)
Forall ?lt ?dt (
   rif:error("datatype violation") :- And (
     ?lt[rdf:type->?dt] External(pred:isLiteralNotOfType( ?lt ?dt )) ))

(* eq-diff1-literal1 *)
Forall ?x ?y ?s1 ?s2 ?p1 ?p2 (
   rif:error("inconsistent") :- And(
       ?s1[?p1->?x]  ?s2[?p2->?y]
       ?x[owl:sameAs->?y]
       External(pred:literalNotIdentical(?x ?y))  ))

(* eq-diff1-literal2 *)
Forall ?x ?y ?s1 ?s2 ?p1 ?p2 (
   rif:error("inconsistent") :- And(
       ?s1[?p1->?x]  ?s2[?p2->?y]
       ?x = ?y
       ?x[owl:differentFrom->?y]  ))

(* dt-type1-text *) Forall ( rdfs:text[rdf:type -> rdfs:Datatype] )
(* dt-type1-decimal *) Forall ( xsd:decimal[rdf:type -> rdfs:Datatype] )
(* dt-type1-integer *) Forall ( xsd:integer[rdf:type -> rdfs:Datatype] )
(* dt-type1-double *) Forall ( xsd:double[rdf:type -> rdfs:Datatype] )
(* dt-type1-string *) Forall ( xsd:string[rdf:type -> rdfs:Datatype] )
(* dt-type1-dateTime *) Forall ( xsd:dateTime[rdf:type -> rdfs:Datatype] )
(* dt-type1-XMLLiteral *) Forall ( rdf:XMLLiteral[rdf:type -> rdfs:Datatype
(* dt-type1-Literal *) Forall ( rdfs:Literal[rdf:type -> rdfs:Datatype] )

(* dt-type1-nonNegativeInteger *) Forall ( xsd:nonNegativeInteger[rdf:type
(* dt-type1-nonPositiveInteger *) Forall ( xsd:nonPositiveInteger[rdf:type
(* dt-type1-positiveInteger *) Forall ( xsd:positiveInteger[rdf:type -> rdf
(* dt-type1-negativeInteger *) Forall ( xsd:negativeInteger[rdf:type -> rdf
(* dt-type1-long *) Forall ( xsd:long[rdf:type -> rdfs:Datatype] )
(* dt-type1-int *) Forall ( xsd:int[rdf:type -> rdfs:Datatype] )
(* dt-type1-short *) Forall ( xsd:short[rdf:type -> rdfs:Datatype] )
(* dt-type1-byte *) Forall ( xsd:byte[rdf:type -> rdfs:Datatype] )
(* dt-type1-unsignedLong *) Forall ( xsd:unsignedLong[rdf:type -> rdfs:Data
(* dt-type1-unsignedInt *) Forall ( xsd:unsignedInt[rdf:type -> rdfs:Dataty
(* dt-type1-unsignedShort *) Forall ( xsd:unsignedShort[rdf:type -> rdfs:Da
(* dt-type1-unsignedByte *) Forall ( xsd:unsignedByte[rdf:type -> rdfs:Data
(* dt-type1-normalizedString *) Forall ( xsd:normalizedString[rdf:type -> r
(* dt-type1-token *) Forall ( xsd:token[rdf:type -> rdfs:Datatype] )
(* dt-type1-language *) Forall ( xsd:language[rdf:type -> rdfs:Datatype] )
(* dt-type1-Name *) Forall ( xsd:Name[rdf:type -> rdfs:Datatype] )
```

```
(* dt-type1-NCName *) Forall ( xsd:NCName[rdf:type -> rdfs:Datatype] )
(* dt-type1-NMTOKEN *) Forall ( xsd:NMTOKEN[rdf:type -> rdfs:Datatype] )

(* dt-type1-float *) Forall ( xsd:float[rdf:type -> rdfs:Datatype] )
(* dt-type1-boolean *) Forall ( xsd:boolean[rdf:type -> rdfs:Datatype] )
(* dt-type1-hexBinary *) Forall ( xsd:hexBinary[rdf:type -> rdfs:Datatype]
(* dt-type1-base64Binary *) Forall ( xsd:base64Binary[rdf:type -> rdfs:Data
(* dt-type1-anyURI *) Forall ( xsd:anyURI[rdf:type -> rdfs:Datatype] )
(* dt-type1-dateTimeStamp  *) Forall ( xsd:dateTimeStamp [rdf:type -> rdfs:

))
```

7 Appendix: OWL 2 RL to RIF translation

The static set of rules in the first appendix provides a complete translation of the OWL 2 RL rules into RIF. While that rule set is within the RIF Core dialect it is fairly inefficient, for example in its handling of lists rules.

In practice we would expect many OWL 2 RL implementations to instantiate the ruleset for a particular ontology. The instantiation process only depends upon OWL TBox axioms and the instantiated ruleset can be applied to other ontologies which only differ by virtue of the ABox assertions.

We here define an algorithm for instantiating a RIF Core rule set for a given OWL 2 RL ontology.

Input: An ontology *O* conforming to the OWL 2 RL profile and the corresponding translation of *O* into an RDF Graph *RDF(O)* as specified in the OWL 2 Mapping to RDF Graphs [*OWL 2 RDF Mapping*].

Output: A RIF Core rule set *R(RDF(O))* such that the RIF-RDF combination of *R(RDF(O))* and *RDF(O)* has the same entailments as the combination *R* and *RDF(O)* where *R* is the static OWL 2 RL rule set defined above.

Algorithm: `R(RDF(O)) = FixedRules ∪ templateRules( RDF(O) )`

Where the set of FixedRules and the two algorithms `templateRules` are defined below.

7.1 FixedRules

The *FixedRule* ruleset comprises the following rules:

```
(* eq-ref *)
Forall ?p ?o ?s (
   ?s[owl:sameAs->?s] :- ?s[?p->?o])
```

```
(* eq-ref1 *)
Forall ?p ?o ?s (
   ?p[owl:sameAs->?p] :- ?s[?p->?o])

(* eq-ref2 *)
Forall ?p ?o ?s (
   ?o[owl:sameAs->?o] :- ?s[?p->?o])

(* eq-sym *)
Forall ?x ?y (
   ?y[owl:sameAs->?x] :- ?x[owl:sameAs->?y])

(* eq-trans *)
Forall ?x ?z ?y (
   ?x[owl:sameAs->?z] :- And(
       ?x[owl:sameAs->?y]
       ?y[owl:sameAs->?z]  ))

(* eq-rep-s *)
Forall ?p ?o ?s ?s2 (
   ?s2[?p->?o] :- And(
       ?s[owl:sameAs->?s2]
       ?s[?p->?o]  ))

(* eq-rep-p *)
Forall ?p ?o ?s ?p2 (
   ?s[?p2->?o] :- And(
       ?p[owl:sameAs->?p2]
       ?s[?p->?o]  ))

(* eq-rep-o *)
Forall ?p ?o ?s ?o2 (
   ?s[?p->?o2] :- And(
       ?o[owl:sameAs->?o2]
       ?s[?p->?o]  ))

(* eq-diff1 *)
Forall ?x ?y (
   rif:error("inconsistent") :- And(
       ?x[owl:sameAs->?y]
       ?x[owl:differentFrom->?y]  ))

(* prp-ap-label *)
Forall (
   rdfs:label[rdf:type->owl:AnnotationProperty])

(* prp-ap-comment *)
Forall (
   rdfs:comment[rdf:type->owl:AnnotationProperty])
```

```
(* prp-ap-seeAlso *)
Forall (
   rdfs:seeAlso[rdf:type->owl:AnnotationProperty])

(* prp-ap-isDefinedBy *)
Forall (
   rdfs:isDefinedBy[rdf:type->owl:AnnotationProperty])

(* prp-ap-deprecated *)
Forall (
   owl:deprecated[rdf:type->owl:AnnotationProperty])

(* prp-ap-priorVersion *)
Forall (
   owl:priorVersion[rdf:type->owl:AnnotationProperty])

(* prp-ap-backwardCompatibleWith *)
Forall (
   owl:backwardCompatibleWith[rdf:type->owl:AnnotationProperty])

(* prp-ap-incompatibleWith *)
Forall (
   owl:incompatibleWith[rdf:type->owl:AnnotationProperty])

(* prp-dom *)
Forall ?p ?c ?x ?y (
   ?x[rdf:type->?c] :- And(
       ?p[rdfs:domain->?c]
       ?x[?p->?y]  ))

(* prp-rng *)
Forall ?p ?c ?x ?y (
   ?y[rdf:type->?c] :- And(
       ?p[rdfs:range->?c]
       ?x[?p->?y]  ))

(* cls-thing *)
Forall (
   owl:Thing[rdf:type->owl:Class])

(* cls-nothing1 *)
Forall (
   owl:Nothing[rdf:type->owl:Class])

(* cls-nothing2 *)
Forall ?x (
   rif:error("inconsistent") :- ?x[rdf:type->owl:Nothing])
```

```
(* cax-sco *)
Forall ?x ?c1 ?c2 (
   ?x[rdf:type->?c2] :- And(
       ?c1[rdfs:subClassOf->?c2]
       ?x[rdf:type->?c1]  ))

(* cax-eqc1 *)
Forall ?x ?c1 ?c2 (
   ?x[rdf:type->?c2] :- And(
       ?c1[owl:equivalentClass->?c2]
       ?x[rdf:type->?c1]  ))

(* cax-eqc2 *)
Forall ?x ?c1 ?c2 (
   ?x[rdf:type->?c1] :- And(
       ?c1[owl:equivalentClass->?c2]
       ?x[rdf:type->?c2]  ))

(* cax-dw *)
Forall ?x ?c1 ?c2 (
   rif:error("inconsistent") :- And(
       ?c1[owl:disjointWith->?c2]
       ?x[rdf:type->?c1]
       ?x[rdf:type->?c2]  ))

(* scm-cls *)
Forall ?c (
   ?c[rdfs:subClassOf->?c] :- ?c[rdf:type->owl:Class])

(* scm-cls1 *)
Forall ?c (
   ?c[owl:equivalentClass->?c] :- ?c[rdf:type->owl:Class])

(* scm-cls2 *)
Forall ?c (
   ?c[rdfs:subClassOf->owl:Thing] :- ?c[rdf:type->owl:Class])

(* scm-cls3 *)
Forall ?c (
   owl:Nothing[rdfs:subClassOf->?c] :- ?c[rdf:type->owl:Class])

(* scm-sco *)
Forall ?c1 ?c2 ?c3 (
   ?c1[rdfs:subClassOf->?c3] :- And(
       ?c1[rdfs:subClassOf->?c2]
       ?c2[rdfs:subClassOf->?c3]  ))
```

```
(* scm-eqc1 *)
Forall ?c1 ?c2 (
   ?c1[rdfs:subClassOf->?c2] :- ?c1[owl:equivalentClass->?c2])

(* scm-eqc11 *)
Forall ?c1 ?c2 (
   ?c2[rdfs:subClassOf->?c1] :- ?c1[owl:equivalentClass->?c2])

(* scm-eqc2 *)
Forall ?c1 ?c2 (
   ?c1[owl:equivalentClass->?c2] :- And(
       ?c1[rdfs:subClassOf->?c2]
       ?c2[rdfs:subClassOf->?c1]  ))

(* scm-op *)
Forall ?p (
   ?p[rdfs:subPropertyOf->?p] :- ?p[rdf:type->owl:ObjectProperty])

(* scm-op1 *)
Forall ?p (
   ?p[owl:equivalentProperty->?p] :- ?p[rdf:type->owl:ObjectProperty])

(* scm-dp *)
Forall ?p (
   ?p[rdfs:subPropertyOf->?p] :- ?p[rdf:type->owl:DatatypeProperty])

(* scm-dp1 *)
Forall ?p (
   ?p[owl:equivalentProperty->?p] :- ?p[rdf:type->owl:DatatypeProperty])

(* scm-spo *)
Forall ?p3 ?p2 ?p1 (
   ?p1[rdfs:subPropertyOf->?p3] :- And(
       ?p1[rdfs:subPropertyOf->?p2]
       ?p2[rdfs:subPropertyOf->?p3]  ))

(* scm-eqp1 *)
Forall ?p2 ?p1 (
   ?p1[rdfs:subPropertyOf->?p2] :- ?p1[owl:equivalentProperty->?p2])

(* scm-eqp11 *)
Forall ?p2 ?p1 (
   ?p2[rdfs:subPropertyOf->?p1] :- ?p1[owl:equivalentProperty->?p2])

(* scm-eqp2 *)
Forall ?p2 ?p1 (
   ?p1[owl:equivalentProperty->?p2] :- And(
       ?p1[rdfs:subPropertyOf->?p2]
       ?p2[rdfs:subPropertyOf->?p1]  ))
```

```
(* scm-dom1 *)
Forall ?p ?c1 ?c2 (
   ?p[rdfs:domain->?c2] :- And(
       ?p[rdfs:domain->?c1]
       ?c1[rdfs:subClassOf->?c2]  ))

(* scm-dom2 *)
Forall ?c ?p2 ?p1 (
   ?p1[rdfs:domain->?c] :- And(
       ?p2[rdfs:domain->?c]
       ?p1[rdfs:subPropertyOf->?p2]  ))

(* scm-rng1 *)
Forall ?p ?c1 ?c2 (
   ?p[rdfs:range->?c2] :- And(
       ?p[rdfs:range->?c1]
       ?c1[rdfs:subClassOf->?c2]  ))

(* scm-rng2 *)
Forall ?c ?p2 ?p1 (
   ?p1[rdfs:range->?c] :- And(
       ?p2[rdfs:range->?c]
       ?p1[rdfs:subPropertyOf->?p2]  ))

(* dt-type2 *)
Forall ?s ?p ?lt ( ?lt[rdf:type->?ty rdf:type->rdfs:Literal]
                     :- And( ?s[?p->?lt] External( pred:isLiteralOfType(?lt

(* dt-not-type *)
Forall ?lt ?dt (
   rif:error('datatype violation') :- And (
     ?lt[rdf:type->dt] External(pred:isLiteralNotOfType( ?lt ?dt )) ))

(* eq-diff1-literal1 *)
Forall ?x ?y ?s1 ?s2 ?p1 ?p2 (
   rif:error("inconsistent") :- And(
       ?s1[?p1->?x]  ?s2[?p2->?y]
       ?x[owl:sameAs->?y]
       External(pred:literalNotIdentical(?x ?y))  ))

(* eq-diff1-literal2 *)
Forall ?x ?y ?s1 ?s2 ?p1 ?p2 (
   rif:error("inconsistent") :- And(
       ?s1[?p1->?x]  ?s2[?p2->?y]
       ?x = ?y
       ?x[owl:differentFrom->?y]  ))

(* dt-type1-text *) Forall ( rdfs:text[rdf:type -> rdfs:Datatype] )
(* dt-type1-decimal *) Forall ( xsd:decimal[rdf:type -> rdfs:Datatype] )
```

```
(* dt-type1-integer *) Forall ( xsd:integer[rdf:type -> rdfs:Datatype] )
(* dt-type1-double *) Forall ( xsd:double[rdf:type -> rdfs:Datatype] )
(* dt-type1-string *) Forall ( xsd:string[rdf:type -> rdfs:Datatype] )
(* dt-type1-dateTime *) Forall ( xsd:dateTime[rdf:type -> rdfs:Datatype] )
(* dt-type1-XMLLiteral *) Forall ( rdf:XMLLiteral[rdf:type -> rdfs:Datatype
(* dt-type1-Literal *) Forall ( rdfs:Literal[rdf:type -> rdfs:Datatype] )

(* dt-type1-nonNegativeInteger *) Forall ( xsd:nonNegativeInteger[rdf:type
(* dt-type1-nonPositiveInteger *) Forall ( xsd:nonPositiveInteger[rdf:type
(* dt-type1-positiveInteger *) Forall ( xsd:positiveInteger[rdf:type -> rdf
(* dt-type1-negativeInteger *) Forall ( xsd:negativeInteger[rdf:type -> rdf
(* dt-type1-long *) Forall ( xsd:long[rdf:type -> rdfs:Datatype] )
(* dt-type1-int *) Forall ( xsd:int[rdf:type -> rdfs:Datatype] )
(* dt-type1-short *) Forall ( xsd:short[rdf:type -> rdfs:Datatype] )
(* dt-type1-byte *) Forall ( xsd:byte[rdf:type -> rdfs:Datatype] )
(* dt-type1-unsignedLong *) Forall ( xsd:unsignedLong[rdf:type -> rdfs:Data
(* dt-type1-unsignedInt *) Forall ( xsd:unsignedInt[rdf:type -> rdfs:Dataty
(* dt-type1-unsignedShort *) Forall ( xsd:unsignedShort[rdf:type -> rdfs:Da
(* dt-type1-unsignedByte *) Forall ( xsd:unsignedByte[rdf:type -> rdfs:Data
(* dt-type1-normalizedString *) Forall ( xsd:normalizedString[rdf:type -> r
(* dt-type1-token *) Forall ( xsd:token[rdf:type -> rdfs:Datatype] )
(* dt-type1-language *) Forall ( xsd:language[rdf:type -> rdfs:Datatype] )
(* dt-type1-Name *) Forall ( xsd:Name[rdf:type -> rdfs:Datatype] )
(* dt-type1-NCName *) Forall ( xsd:NCName[rdf:type -> rdfs:Datatype] )
(* dt-type1-NMTOKEN *) Forall ( xsd:NMTOKEN[rdf:type -> rdfs:Datatype] )

(* dt-type1-float *) Forall ( xsd:float[rdf:type -> rdfs:Datatype] )
(* dt-type1-boolean *) Forall ( xsd:boolean[rdf:type -> rdfs:Datatype] )
(* dt-type1-hexBinary *) Forall ( xsd:hexBinary[rdf:type -> rdfs:Datatype]
(* dt-type1-base64Binary *) Forall ( xsd:base64Binary[rdf:type -> rdfs:Data
(* dt-type1-anyURI *) Forall ( xsd:anyURI[rdf:type -> rdfs:Datatype] )
(* dt-type1-dateTimeStamp  *) Forall ( xsd:dateTimeStamp [rdf:type -> rdfs:
```

7.2 templateRules algorithm

We specify the algorithm for instantiating the template rules by means of a translation table.

The first column gives a set of RDF triple patterns `(s p o)` where any of the elements can be a variable (indicated with a '?') prefix.

The second column gives a template to be instantiated. For each match of the triple patterns in *RDF(O)* we generate a binding map which maps each variable in the triple pattern to a corresponding RDF Node in *RDF(O)*. The template should be processed with this binding map, replacing the corresponding variables by their mapped values.

For example the pair:

Pattern	Template
(?p rdf:type owl:SymmetricProperty)	`Forall ?x ?y ( ?y[?p->?x] :- And( ?x[?p->?y] ))`

Applied to an ontology containing the following RDF triples:

```
eg:p rdf:type owl:SymmetricProperty .
eg:q rdf:type owl:SymmetricProperty .
```

Would emit the follow RIF rules:

```
Forall ?x ?y (
   ?y[eg:p->?x] :- And(
       ?x[eg:p->?y] ))

Forall ?x ?y (
   ?y[eg:q->?x] :- And(
       ?x[eg:q->?y] ))
```

In order to specify the list related rules we need to define some addition notation for templates.

Notation	Interpretation
`{$ rule text $}`	Emit the *rule text* substituting any occurrence of variables from the binding map. The outer {$ $} can be omitted in cases where there is no ambiguity.
`for(?elt in ?list) { template }`	?list is a variable in the pattern which is bound to an RDF List. The *for* operator iterates over each element of the *?list* in turn replacing the binding map for the ?elt variable with the next list entry and processes the enclosed *template* in the context of that new binding map.
`$length(?list)$`	Used within a template this will be replaced by the length the RDFList bound to *?list*.
`$i$`	Used within a *for(?elt in ?list){ template }* this will be replaced by the index of the current ?elt.

`$i+1$`	Used within a *for(?elt in ?list){ template }* this will be replaced by the index of the current ?elt, plus 1.

The templateRule algorithm is defined using this notation by the following set of pattern/template pairs.

Pattern	Template

(?p rdf:type owl:FunctionalProperty)	`(* prp-fp *)`
	`Forall ?y2 ?x ?y1 (`
	`  ?y1[owl:sameAs->?y2] :- And(`
	`  ?x[?p->?y1]`
	`  ?x[?p->?y2] ))`

(?p rdf:type owl:InverseFunctionalProperty])	`(* prp-ifp *)`
	`Forall ?x1 ?x2 ?y (`
	`  ?x1[owl:sameAs->?x2] :- And(`
	`  ?x1[?p->?y]`
	`  ?x2[?p->?y] ))`

(?p rdf:type owl:IrreflexiveProperty)	`(* prp-irp *)`
	`Forall ?x (`
	`  rif:error("inconsistent") :- And(`
	`  ?x[?p->?x] ))`

(?p rdf:type owl:SymmetricProperty)	`(* prp-symp *)`
	`Forall ?x ?y (`
	`  ?y[?p->?x] :- And(`
	`  ?x[?p->?y] ))`

(?p rdf:type owl:AsymmetricProperty)	`(* prp-asyp *)`
	`Forall ?x ?y (`
	`  rif:error("inconsistent") :- And(`
	`  ?x[?p->?y]`
	`  ?y[?p->?x] ))`

(?p rdf:type owl:TransitiveProperty)	(* prp-trp *) Forall ?x ?z ?y (?x[?p->?z] :- And(?x[?p->?y] ?y[?p->?z]))
(?p1 rdfs:subPropertyOf ?p2)	(* prp-spo1 *) Forall ?x ?y (?x[?p2->?y] :- And(?x[?p1->?y]))
(?p1 owl:equivalentProperty ?p2)	(* prp-eqp1 *) Forall ?x ?y (?x[?p2->?y] :- And(?x[?p1->?y])) (* prp-eqp2 *) Forall ?x ?y (?x[?p1->?y] :- And(?x[?p2->?y]))
(?p1 owl:propertyDisjointWith ?p2)	(* prp-pdw *) Forall ?x ?y (rif:error("inconsistent") :- And(?x[?p1->?y] ?x[?p2->?y]))
(?p1 owl:inverseOf ?p2)	(* prp-inv1 *) Forall ?x ?y (?y[?p2->?x] :- And(?x[?p1->?y])) (* prp-inv2 *) Forall ?x ?y (?y[?p1->?x] :- And(?x[?p2->?y]))
(?x owl:someValuesFrom ?y) (?x owl:onProperty ?p)	(* cls-svf1 *) Forall ?v ?u (

	?u[rdf:type->?x] :- And(?u[?p->?v] ?v[rdf:type->?y]))
(?x owl:someValuesFrom owl:Thing) (?x owl:onProperty ?p)	(* cls-svf2 *) Forall ?v ?u (?u[rdf:type->?x] :- And(?u[?p->?v]))
(?x owl:allValuesFrom ?y) (?x owl:onProperty ?p)	(* cls-avf *) Forall ?v ?u (?v[rdf:type->?y] :- And(?u[rdf:type->?x] ?u[?p->?v]))
(?x owl:hasValue ?y) (?x owl:onProperty ?p)	(* cls-hv1 *) Forall ?u (?u[?p->?y] :- And(?u[rdf:type->?x])) (* cls-hv2 *) Forall ?u (?u[rdf:type->?x] :- And(?u[?p->?y]))
(?x owl:maxCardinality 0) (?x owl:onProperty ?p)	(* cls-maxc1 *) Forall ?u ?y (rif:error("inconsistent") :- And(?u[?p->?y] ?u[rdf:type->?x]))
(?x owl:maxCardinality 1) (?x owl:onProperty ?p)	(* cls-maxc2 *) Forall ?y2 ?u ?y1 (?y1[owl:sameAs->?y2] :- And(?u[?p->?y1] ?u[?p->?y2] ?u[rdf:type->?x]))

(?x owl:maxQualifiedCardinality 0) (?x owl:onProperty ?p) (?x owl:onClass ?c)	(* cls-maxqc1 *) Forall ?u ?y (rif:error("inconsistent") :- And(?u[rdf:type->?x] ?u[?p->?y] ?y[rdf:type->?c]))
(?x owl:maxQualifiedCardinality 0) (?x owl:onProperty ?p) (?x owl:onClass owl:Thing)	(* cls-maxqc2 *) Forall ?y (rif:error("inconsistent") :- And(?u[rdf:type->?x] ?u[?p->?y]))
(?x owl:maxQualifiedCardinality 1) (?x owl:onProperty ?p) (?x owl:onClass ?c)	(* cls-maxqc3 *) Forall ?y2 ?u ?y1 (?y1[owl:sameAs->?y2] :- And(?u[rdf:type->?x] ?u[?p->?y1] ?y1[rdf:type->?c] ?u[?p->?y2] ?y2[rdf:type->?c]))
(?x owl:maxQualifiedCardinality 1) (?x owl:onProperty ?p) (?x owl:onClass owl:Thing)	(* cls-maxqc4 *) Forall ?y2 ?u ?y1 (?y1[owl:sameAs->?y2] :- And(?u[rdf:type->?x] ?u[?p->?y1] ?u[?p->?y2]))
(?c1 owl:hasValue ?i) (?c1 owl:onProperty ?p1) (?c2 owl:hasValue ?i) (?c2 owl:onProperty ?p2)	(* scm-hv *) Forall (?c1[rdfs:subClassOf->?c2] :- And(?p1[rdfs:subPropertyOf->?p2]))
(?c1 owl:someValuesFrom ?y1) (?c1 owl:onProperty ?p) (?c2 owl:someValuesFrom ?y2) (?c2 owl:onProperty ?p)	(* scm-svf1 *) Forall (?c1[rdfs:subClassOf->?c2] :- And(?y1[rdfs:subClassOf->?y2]))

(?c1 owl:someValuesFrom ?y) (?c1 owl:onProperty ?p1) (?c2 owl:someValuesFrom ?y) (?c2 owl:onProperty ?p2)	(* scm-svf2 *) Forall (?c1[rdfs:subClassOf->?c2] :- And(?p1[rdfs:subPropertyOf->?p2]))
(?c1 owl:allValuesFrom ?y1) (?c1 owl:onProperty ?p) (?c2 owl:allValuesFrom ?y2) (?c2 owl:onProperty ?p)	(* scm-avf1 *) Forall (?c1[rdfs:subClassOf->?c2] :- And(?y1[rdfs:subClassOf->?y2]))
(?c1 owl:allValuesFrom ?y) (?c1 owl:onProperty ?p1) (?c2 owl:allValuesFrom ?y) (?c2 owl:onProperty ?p2)	(* scm-avf2 *) Forall (?c2[rdfs:subClassOf->?c1] :- And(?p1[rdfs:subPropertyOf->?p2]))

(?a rdf:type owl:AllDifferent)
(?a owl:members ?l)
	for(?x in ?l) {
	for(?y in ?l) {
	{$
	Forall (
	rif:error('AllDifferent') :- And (
	?x[owl:sameAs->?y]))
	$}
	}
	}

(?a rdf:type owl:AllDifferent)
(?a owl:distinctMembers ?l)
	for(?x in ?l) {
	for(?y in ?l) {
	{$
	Forall (
	rif:error('AllDifferent') :- And (
	?x[owl:sameAs->?y]))
	$}
	}
	}|-

(?l rdf:type owl:AllDisjointClasses)	for(?x in ?l) {
	for(?y in ?l) {
	{$
	Forall ?c1 ?c2 (

	rif:error('AllDisjointClasses') :- And (?x[rdf:type->?c1 rdf:type->?c2])) $} } }

(?p owl:propertyChainAxiom ?pc)	{$
	Forall ?u0 ?u$length(?pc)$ (
	?start[?p->?last] :- And (
	$}
	for(?next in ?sc) {
	{$
	?ui[?next->?u$i+1$]
	$}
	}
	{$)) $}

(?c owl:hasKey ?u)	{$
	Forall ?x ?y (
	?x[owl:sameAs->?y] :- And (
	?x[rdf:type->?c] ?y[rdf:type->?c]
	$}
	for(?key in ?u) {
	{$
	?x[?key->?v] ?y[?key->?v]
	$}
	}
	{$)) $}

(?c owl:intersectionOf ?l)	{$
	Forall ?y (
	?y[rdf:type->?c] :- And (
	$}
	for(?ty in ?l) {
	{$
	?y[rdf:type->?ty]
	$}
	}
	{$)) $}

(?c owl:unionOf ?l)	`for(?ci in ?l) { {$ Forall ?y ( ?y[rdf:type->?c] :- And ( ?y[rdf:type->?ci] )) $} }`
(?c owl:oneOf ?l)	`for(?yi in ?l) { {$ Forall ( ?yi[rdf:type->?c] ) $} }`
(?c owl:intersectionOf ?l)	`for(?ci in ?l) { {$ Forall ?y ( ?y[rdf:type->?ci] :- And ( ?y[rdf:type->?c] )) Forall( ?c[rdfs:subClassOf->?ci] ) $} }`
(?c owl:unionOf ?l)	`for(?ci in ?l) { {$ Forall ( ?ci[rdfs:subClassOf->?c] ) $} }`
(?x owl:sourceIndividual ?i1) (?x owl:assertionProperty ?p) (?x owl:targetIndividual ?i2)	`(* prp-npa1 *) Forall ( rif:error("inconsistent") :- And( ?i1[?p->?i2] ))`
(?x owl:sourceIndividual ?i1) (?x owl:assertionProperty ?p) (?x owl:targetValue ?i2)	`(* prp-npa2 *) Forall ( rif:error("inconsistent") :- And( ?i1[?p->?i2] ))`

(?c1 owl:disjointWith ?c2)	(* cax-dw *) Forall ?x (rif:error("inconsistent") :- And(?x[rdf:type->?c1] ?x[rdf:type->?c2]))
(?c1 owl:complementOf ?c2)	(* cls-com *) Forall ?x (rif:error("inconsistent") :- And(?x[rdf:type->?c1] ?x[rdf:type->?c2]))