W3C Editor's Draft

RIF RDF and OWL Compatibility

W3C Editor's Draft 28 October 2012

This version:
http://www.w3.org/2005/rules/wg/draft/ED-rif-rdf-owl-20121028/
Latest editor's draft:
http://www.w3.org/2005/rules/wg/draft/rif-rdf-owl/
Previous version:
http://www.w3.org/2005/rules/wg/draft/REC-rif-rdf-owl-20100622/
Editors:
Jos de Bruijn, Vienna University of Technology
Chris Welty, IBM Research

A color-coded version of this document showing changes made since the previous version is also available.

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

Abstract

Rules interchanged using the Rule Interchange Format RIF may depend on or be used in combination with RDF data and RDF Schema or OWL ontologies. This document, developed by the Rule Interchange Format (RIF) Working Group, specifies the interoperation between RIF and the data and ontology languages RDF, RDF Schema, and OWL.

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May Be Superseded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 12 documents:

1. RIF Overview
2. RIF Core Dialect
3. RIF Basic Logic Dialect
4. RIF Production Rule Dialect
5. RIF Framework for Logic Dialects
6. RIF Datatypes and Built-Ins 1.0
7. RIF RDF and OWL Compatibility (this document)
8. OWL 2 RL in RIF
9. RIF Combination with XML data
10. RIF In RDF
11. RIF Test Cases
12. RIF Primer

Summary of Changes

There have been no substantive changes since the previous version. For details on the minor changes see the change log and color-coded diff.

W3C Members Please Review By 25 November 2012

The W3C Director seeks review and feedback from W3C Advisory Committee representatives, via their review form by 25 November 2012. This will allow the Director to assess consensus and determine whether to issue this document as a W3C Edited Recommendation.

Others are encouraged by the Rule Interchange Format (RIF) Working Group to continue to send reports of implementation experience, and other feedback, to public-rif-comments@w3.org (public archive). Reports of any success or difficulty with the test cases are encouraged. Open discussion among developers is welcome at public-rif-dev@w3.org (public archive).

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. 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 Overview of RDF and OWL Compatibility

The Rule Interchange Format (RIF) is a format for interchanging rules over the Web. Rules that are exchanged using RIF may refer to external data sources and may be based on data models that are represented using a language different from RIF. The Resource Description Framework RDF [RDF-Concepts] is a Web-based language for the representation and exchange of data; RDF Schema (RDFS) [RDF-Schema] and the OWL Web Ontology Language [OWL2-Syntax] are Web-based languages for representing and exchanging ontologies. This document specifies how combinations of RIF documents and RDF data and RDFS and OWL ontologies are interpreted; i.e., it specifies how RIF interoperates with RDF, RDFS, and OWL. We consider here OWL 2 [OWL2-Syntax], which is an extension of OWL 1 [OWL-Reference]. Therefore, the notions defined in this document also apply to combinations of RIF documents with OWL 1 ontologies.

We consider here the RIF Basic Logic Dialect (BLD) [RIF-BLD] and RIF Core [RIF-Core], a subset of RIF BLD. The RIF Production Rule Dialect (PRD) [RIF-PRD] is an extension of RIF Core. Interoperability between RIF and RDF/OWL is only defined for the Core subset of PRD. In the remainder, when speaking about RIF documents and rules, we refer to RIF Core and BLD.

RDF data and RDFS and OWL ontologies can be represented using *RDF graphs*. There exist several alternative syntaxes for OWL ontologies; however, for exchange purposes it is assumed they are represented using RDF graphs.

Several syntaxes have been proposed for the exchange of RDF graphs, the normative syntax being RDF/XML [RDF-Syntax]. RIF does not provide a format for exchanging RDF graphs; it is assumed that RDF graphs are exchanged using RDF/XML, or any other syntax that can be used for representing or exchanging RDF graphs.

A typical scenario for the use of RIF with RDF/OWL is the exchange of rules that use RDF data and/or RDFS or OWL ontologies: an interchange partner *A* has a rules language that is RDF/OWL-aware, i.e., it supports the use of RDF data, it uses an RDFS or OWL ontology, or it extends RDF(S)/OWL. *A* sends its rules using RIF, possibly with references to the appropriate RDF graph(s), to partner *B*. *B* receives the rules and retrieves the referenced RDF graph(s). The rules are translated to the internal rules language of *B* and are processed, together with the RDF graphs, using the RDF/OWL-aware rule engine of *B*. The use case Vocabulary Mapping for Data Integration [RIF-UCR] is an example of the interchange of RIF rules that use RDF data and RDFS ontologies.

A specialization of this scenario is the publication of RIF rules that refer to RDF graphs; publication is a special kind of interchange: one to many, rather than one-to-one. When a rule publisher *A* publishes its rules on the Web, there may be several consumers that retrieve the RIF rules and RDF graphs from the Web, translate the RIF rules to their respective rules languages, and process them together with the RDF graphs in their own rules engines. The use case Publishing Rules for Interlinked Metadata [RIF-UCR] illustrates the publication scenario.

Another specialization of the exchange scenario is the Interchange of Rule Extensions to OWL [RIF-UCR]. The intention of the rule publisher in this scenario is to extend an OWL ontology with rules: interchange partner *A* has a rules language that extends OWL. *A* splits its ontology+rules description into a separate OWL ontology and a RIF document, publishes the OWL ontology, and sends (or publishes) the RIF document, which includes a reference to the OWL ontology. A consumer of the rules retrieves the OWL ontology and translates the ontology and document into a combined ontology+rules description in its own rule extension of OWL.

A RIF document that refers to (*imports*) RDF graphs and/or RDFS/OWL ontologies, or any use of a RIF document with RDF graphs, is viewed as a combination of a document and a number of graphs and ontologies. This document specifies how, in such a combination, the document and the graphs and ontologies interoperate in a technical sense, i.e., the conditions under which the combination is satisfiable (i.e., consistent), as well as the entailments (i.e., logical consequences) of the combination. The interaction between RIF and RDF/OWL is realized by connecting the model theory of RIF [RIF-BLD] with the model theories of RDF [RDF-Semantics] and OWL [OWL2-Semantics], respectively.

The notation of certain symbols in RIF, particularly IRIs and plain literals, is slightly different from the notation in RDF/OWL. These differences are illustrated in the Section Symbols in RIF Versus RDF/OWL.

The RDF Semantics specification [RDF-Semantics] defines four normative notions of entailment for RDF graphs: Simple, RDF, RDFS, and Datatype entailment. OWL 2 specifies two different semantics, with corresponding notions of entailment: the Direct Semantics [OWL2-Semantics] and the RDF-Based Semantics [OWL2-RDF-Based-Semantics]. This document specifies the interaction between RIF and RDF/OWL for all six notions. The Section RDF Compatibility is concerned with the combination of RIF and RDF/RDFS. The combination of RIF and OWL is addressed in the Section OWL Compatibility. The semantics of the interaction between RIF and the OWL 2 Direct Semantics is close in spirit to [SWRL].

RIF provides a mechanism for referring to (importing) RDF graphs and a means for specifying the *profile* of this import, which corresponds to the intended entailment regime. The Section Importing RDF and OWL in RIF specifies how such import statements are used for representing RIF-RDF and RIF-OWL combinations.

The Appendix: Embeddings (Informative) describes how reasoning with combinations of RIF rules with RDF and OWL 2 RL (a subset of OWL 2 DL) can be reduced to reasoning with RIF documents. This reduction can be seen as an implementation hint for interchange partners who do not have RDF/OWL-aware rule systems, but want to process RIF rules that import RDF graphs and OWL ontologies. In terms of the aforementioned scenario: if the interchange partner *B* does not have an RDF/OWL-aware rule system, but *B* can process RIF rules, then the appendix explains how the rule system of *B* could be used for processing RIF-RDF/OWL combinations.

Throughout this document the following conventions are used when writing RIF and RDF statements in examples and definitions.

- All RIF statements are written using the RIF presentation syntax [RIF-BLD]. Where possible, this document uses the shortcut syntax for IRIs and strings as defined in [RIF-DTB].
- RDF triples are written using the Turtle syntax [Turtle]: triples are written as s p o, where s, p, and o are blank nodes _:*x*, IRIs delimited with '<' and '>', compact IRIs *prefix*:*localname*, plain literals without language tags "*literal*", plain literals with language tags "*literal*"@*lang*, or typed literals "*literal*"^^*datatype-IRI*.
- The following namespace prefixes are used throughout this document: ex refers to http://example.org/example#, xs refers to http://www.w3.org/2001/XMLSchema#, rdf refers to http://www.w3.org/1999/02/22-rdf-syntax-ns#, rdfs refers to http://www.w3.org/2000/01/rdf-schema#, owl refers to http://www.w3.org/2002/07/owl#, and rif refers to http://www.w3.org/2007/rif#.

2 Symbols in RIF versus RDF/OWL (Informative)

Where RDF/OWL has four kinds of constants: URI references (i.e., IRIs), plain literals without language tags, plain literals with language tags and typed literals (i.e., Unicode sequences with datatype IRIs) [RDF-Concepts], RIF has one kind of constants: Unicode sequences with symbol space IRIs [RIF-DTB].

Symbol spaces can be seen as groups of constants. Every datatype is a symbol space, but there are symbol spaces that are not datatypes. For example, the symbol space rif:iri groups all IRIs. The correspondence between constant symbols in RDF graphs and RIF documents is explained in Table 1.

Table 1. Correspondence between RDF and RIF symbols.

RDF Symbol	Example	RIF Symbol	Example
IRI	<http://www.w3.org/2007/rif>	Constant in the rif:iri symbol space	"http://www.w3.org/2007/rif"^^rif:iri
Plain literal without language tag	"literal string"	Constant in the rdf:PlainLiteral symbol space	"literal string@"^^rdf:PlainLiteral
Plain literal with language tag	"literal string"@en	Constant in the rdf:PlainLiteral symbol space	"literal string@en"^^rdf:PlainLiteral
Typed literal	"1"^^xs:integer	Constant with symbol space	"1"^^xs:integer

The shortcut syntax for IRIs and strings [RIF-DTB], used throughout this document, corresponds to the syntax for IRIs and plain literals in Turtle [Turtle], a commonly used syntax for RDF.

- IRIs, i.e., constants of the form "*IRI*"^^rif:iri, may be written as <*IRI*> or as compact IRIs [CURIE], i.e., as *prefix*:*localname*, where *prefix* is understood to refer to an IRI *namespace-IRI*, and *prefix*:*localname* stands for the IRI (*IRI*) obtained by concatenating *namespace-IRI* and *localname*.
- Plain literals without language tags, i.e., constants of the form "*my string*@"^^rdf:PlainLiteral may be written as "*my string*".

RIF does not have a notion corresponding exactly to RDF blank nodes. RIF local symbols, written _*symbolname*, have some commonality with blank nodes; like the blank node label, the name of a local symbol is not exposed outside of the document. However, in contrast to blank nodes, which are essentially existentially quantified variables, RIF local symbols are *constant* symbols. In many applications and deployment scenarios, this difference may be inconsequential. However the results will differ when such symbols are used in a non-assertional context, such as in a query pattern or rule body.

Finally, variables in the bodies of RIF rules or in query patterns may be existentially quantified, and are thus similar to blank nodes; however, RIF BLD does not allow existentially quantified variables to occur in rule heads.

3 RDF Compatibility

This section specifies how a RIF document interacts with a set of RDF graphs in a RIF-RDF combination. In other words, how rules can "access" data in the RDF graphs.

There is a correspondence between statements in RDF graphs and certain kinds of formulas in RIF. Namely, there is a correspondence between RDF triples of the form s p o and RIF frame formulas of the form s'[p' -> o'], where s', p', and o' are RIF symbols corresponding to the RDF symbols s, p, and o, respectively. This means that whenever a triple s p o is satisfied, the corresponding RIF frame formula s'[p' -> o'] is satisfied, and vice versa.

Consider, for example, a combination of an RDF graph that contains the triples

```
ex:john ex:brotherOf ex:jack .
ex:jack ex:parentOf ex:mary .
```

saying that ex:john is a brother of ex:jack and ex:jack is a parent of ex:mary, and a RIF document that contains the rule

```
Forall ?x ?y ?z (?x[ex:uncleOf -> ?z] :-
      And(?x[ex:brotherOf -> ?y] ?y[ex:parentOf -> ?z]))
```

which says that whenever some x is a brother of some y and y is a parent of some z, then x is an uncle of z. From this combination the RIF frame formula :john[:uncleOf -> :mary], as well as the RDF triple :john :uncleOf :mary, are consequences of this combination.

Note that blank nodes cannot be referenced directly from RIF rules, since blank nodes are local to a specific RDF graph. Variables in RIF rules do, however, range over objects denoted by blank nodes. So, it is possible to "access" an object denoted by a blank node from a RIF rule using a variable in a rule.

The following example illustrates the interaction between RDF and RIF in the face of blank nodes.

Consider a combination of an RDF graph that contains the triple

```
_:x ex:hasName "John" .
```

saying that there is something, denoted here by a blank node, which has the name "John", and a RIF document that contains the rules

```
Forall ?x ?y ( ?x[rdf:type -> ex:named] :- ?x[ex:hasName -> ?y] )
Forall ?x ?y ( <http://a>[<http://p> -> ?y] :- ?x[ex:hasName -> ?y] )
```

which says that whenever there is some x that has some name y, then x is of type ex:named and http://a has a property http://p with value y.

From this combination the following RIF condition formulas can be derived:

```
Exists ?z (?z[rdf:type -> ex:named])
<http://a>[<http://p> -> "John"]
```

as can the following RDF triples:

```
_:y rdf:type ex:named .
<http://a> <http://p> "John" .
```

However, there is no RIF constant symbol t such that t[rdf:type -> ex:named] can be derived, because there is no constant that represents the named individual.

Note that, even when considering Simple entailment, not every combination is satisfiable. In fact, not every RIF document has a model. For example, the RIF BLD document consisting of the fact

```
"a"="b"
```

does not have a model, since the symbols "a" and "b" are mapped to the (distinct) character strings "a" and "b", respectively, in every semantic structure.

One consequence of the difference of the alphabets of RDF and RIF is that IRIs of the form `http://iri` and typed literals of the form `"http://iri"^^rif:iri` that occur in an RDF graph are treated the same in RIF-RDF combinations, even if the RIF document is empty. However, documents importing RDF graphs containing typed literals of the form `"http://iri"^^rif:iri` must be rejected.

Plain literals without language tags of the form `"mystring"` and typed literals of the form `"mystring"^^xs:string` also correspond. For example, consider the combination of an empty document and an RDF graph that contains the triple

```
<http://a> <http://p> "abc" .
```

This combination entails, among other things, the following frame formula:

```
<http://a>[<http://p> -> "abc"^^xs:string]
```

as well as the following triple:

```
<http://a> <http://p> "abc"^^xs:string .
```

These entailments are sanctioned by the semantics of plain literals and `xs:strings`.

Lists in RDF (also called collections) have a natural correspondence to RIF lists. For example, the RDF list `_:l1 rdf:first ex:b . _:l1 rdf:rest rdf:nil .` corresponds to the RIF list `List(ex:b)`. And so, the combination of the empty RIF document with the RDF graph

```
ex:a ex:p _:l1 .
_:l1 rdf:first ex:b .
_:l1 rdf:rest rdf:nil .
```

entails the formula

```
ex:a[ex:p -> List(ex:b)].
```

Likewise, the combination of the empty RDF graph with the RIF fact

```
ex:p(List(ex:a))
```

entails the triples

```
_:l1 rdf:first ex:a .
_:l1 rdf:rest rdf:nil .
```

as well as the formula

```
Exists ?x (And(ex:p(?x) ?x[rdf:first -> ex:a] ?x[rdf:rest -> rdf:nil])).
```

The remainder of this section formally defines combinations of RIF rules with RDF graphs and the semantics of such combinations. A combination consists of a RIF document and a set of RDF graphs. The semantics of combinations is defined in terms of combined models, which are pairs of RIF and RDF interpretations. The interaction between the two interpretations is defined through a number of conditions. Entailment is defined as model inclusion, as usual.

3.1 Syntax of RIF-RDF Combinations

This section first reviews the definitions of RDF Vocabularies and RDF graphs, after which RIF-RDF combinations are formally defined. The section concludes with a review of definitions related to datatypes and typed literals.

3.1.1 RDF Vocabularies and Graphs

An RDF Vocabulary *V* consists of the following sets of *names*:

- IRIs V_U, (corresponds to the Concepts and Abstract Syntax term "*RDF URI references*"; see the End note on RDF URI references)
- plain literals V_{PL} (i.e., character strings with an optional language tag), and
- typed literals V_{TL} (i.e., pairs of character strings and datatype IRIs).

In addition, there is an infinite set of blank nodes, which is disjoint from the sets of names. See RDF Concepts and Abstract Syntax [RDF-Concepts] for precise definitions of these concepts.

Definition. Given an RDF Vocabulary *V*, a ***generalized RDF triple*** of *V* is a statement of the form `s p o`, where `s`, `p` and `o` are names in *V* or blank nodes. ☐

Definition. Given an RDF Vocabulary *V*, a ***generalized RDF graph*** is a set of ***generalized RDF triples*** of *V*. ☐

(See the End note on generalized RDF graphs)

3.1.2 RIF-RDF Combinations

A RIF-RDF combination consists of a RIF document and zero or more RDF graphs. Formally:

Definition. A ***RIF-RDF combination*** is a pair $< R,\mathbf{S} >$, where *R* is a RIF document and **S** is a set of generalized RDF graphs of a Vocabulary *V*. ☐

When clear from the context, RIF-RDF combinations are referred to simply as ***combinations***.

3.1.3 Datatypes and Typed Literals

Even though RDF allows the use of arbitrary datatype IRIs in typed literals, not all such datatype IRIs are recognized in the semantics. In fact, Simple entailment does not recognize any datatype and RDF and RDFS entailment recognize only the datatype `rdf:XMLLiteral`. To facilitate discussing datatypes, and specifically datatypes supported in specific contexts (required for RIF-D-entailment), the notion of datatype maps [RDF-Semantics] is used.

A ***datatype map*** is a partial mapping from IRIs to datatypes.

RDFS, specifically RIF-D-entailment, allows the use of arbitrary datatype maps, as long as `rdf:XMLLiteral` is in the domain of the map. RIF BLD requires a number of additional datatypes to be included; these are the *RIF-required datatypes* [RIF-DTB].

When checking consistency of a combination $< R,\mathbf{S} >$ or entailment of a graph *S* or RIF formula φ by a combination $< R,\mathbf{S} >$, the set of ***considered datatypes*** is the union of the set of RIF-required datatypes and the sets of datatypes referenced in *R*, the documents imported into *R*, and φ (when considering entailment of φ).

Definition. Let DTS be a set of datatypes. A datatype map D is ***conforming*** with DTS if it satisfies the following conditions:

1. Every IRI identifying a datatype in DTS is in the domain of D.
2. D maps each IRI in its domain to the datatype identified by that IRI in DTS. ☐

Note that it follows from the definition that every datatype used in the RIF document in the combination or the entailed RIF formula (when considering entailment questions) is included in any datatype map conforming to the set of considered datatypes. There may be datatypes used in an RDF graph in the combination that are not included in such a datatype map.

Definition. Given a datatype map D, a typed literal (s, d) is a ***well-typed literal*** if

1. d is in the domain of D and s is in the lexical space of D(d) or
2. d is the IRI of a symbol space required by RIF BLD and s is in the lexical space of the symbol space. ☐

3.2 Semantics of RIF-RDF Combinations

The semantics of RIF-RDF combinations is defined through a combination of the RIF and RDF model theories, using a notion of *common models*. These models are then used to define satisfiability and entailment in the usual way. Combined entailment extends both entailment in RIF and entailment in RDF.

The RDF Semantics document [RDF-Semantics] defines four normative kinds of interpretations, as well as corresponding notions of satisfiability and entailment:

- Simple interpretations, which do not impose any conditions on the RDF and RDFS Vocabularies,
- RDF interpretations, which impose additional conditions on the interpretation of the RDF Vocabulary,
- RDFS interpretations, which impose additional conditions on the interpretation of the RDF and RDFS Vocabularies, and
- D-interpretations, which impose additional conditions on the treatment of datatypes, relative to a datatype map D.

Those four types of interpretations are reflected in the definitions of satisfaction and entailment in this section.

3.2.1 Interpretations

This section defines the notion of *common-RIF-RDF-interpretation*, which is an interpretation of a RIF-RDF combination. This common-RIF-RDF-interpretation is the basis for the definitions of satisfaction and entailment in the following sections.

The correspondence between RIF semantic structures (interpretations) and RDF interpretations is defined through a number of conditions that ensure the correspondence in the interpretation of names (i.e., IRIs and literals) and formulas, i.e., the correspondence between RDF triples of the form `s p o` and RIF frames of the form `s'[p' -> o']`, where `s'`, `p'`, and `o'` are RIF symbols corresponding to the RDF symbols `s`, `p`, and `o`, respectively (cf. the Section Symbols in RIF Versus RDF/OWL).

3.2.1.1 RDF and RIF Interpretations

The notions of RDF interpretation and RIF semantic structure (interpretation) are briefly reviewed below.

As defined in [RDF-Semantics], a ***Simple interpretation*** of a Vocabulary *V* is a tuple I=< IR, IP, IEXT, IS, IL, LV >, where

- IR is a non-empty set of resources (the domain),
- IP is a set of properties,
- IEXT is an extension function, which is a mapping from IP into the power set of IR × IR,
- IS is a mapping from IRIs in *V* into (IR union IP),
- IL is a mapping from typed literals in *V* into IR, and
- LV is the set of literal values, which is a subset of IR, and includes all plain literals in *V*.

RDF-, RDFS-, and D-interpretations are Simple interpretations that satisfy certain conditions:

- A Simple interpretation I of a Vocabulary *V* is an ***RDF-interpretation*** if *V* includes the RDF Vocabulary and I satisfies the RDF axiomatic triples and the RDF semantic conditions.
- An RDF-interpretation I of a Vocabulary *V* is an ***RDFS-interpretation*** if *V* includes the RDFS Vocabulary and I satisfies the RDFS axiomatic triples and the RDFS semantic conditions.
- Given a datatype map D, an RDFS-interpretation I of a Vocabulary *V* is a ***D-interpretation*** if *V* includes the IRIs in the domain of D and I satisfies the general semantic conditions for datatypes for every pair <d, D(d)> such that d is in the domain of D.

As defined in [RIF-BLD], a ***semantic structure*** $\boldsymbol{I}$ is a tuple of the form <$\boldsymbol{TV}$, $\boldsymbol{DTS}$, $\boldsymbol{D}$, $\boldsymbol{D}_{ind}$, $\boldsymbol{D}_{func}$, $\boldsymbol{I}_C$, $\boldsymbol{I}_V$, $\boldsymbol{I}_F$, $\boldsymbol{I}_{NF}$, $\boldsymbol{I}_{list}$, $\boldsymbol{I}_{tail}$, $\boldsymbol{I}_{frame}$, $\boldsymbol{I}_{sub}$, $\boldsymbol{I}_{isa}$, $\boldsymbol{I}_{=}$, $\boldsymbol{I}_{external}$, $\boldsymbol{I}_{truth}$>. The specification of RIF-RDF compatibility is only concerned with $\boldsymbol{DTS}$, $\boldsymbol{D}$, $\boldsymbol{I}_C$, $\boldsymbol{I}_V$, $\boldsymbol{I}_{list}$, $\boldsymbol{I}_{tail}$, $\boldsymbol{I}_{frame}$, $\boldsymbol{I}_{sub}$, $\boldsymbol{I}_{isa}$, and $\boldsymbol{I}_{truth}$. The other mappings that are parts of a semantic structure are not used in the definition of combinations.

Recall that `Const` is the set of constant symbols and `Var` is the set of variable symbols in RIF.

- $\boldsymbol{DTS}$ is the set of datatypes, which have associated datatype identifiers,
- $\boldsymbol{D}$ is a set (the domain),
- $\boldsymbol{D}_{ind}$ is a non-empty subset of $\boldsymbol{D}$,
- $\boldsymbol{D}_{func}$ is a non-empty subset of $\boldsymbol{D}$,
- $\boldsymbol{I}_C$ is a mapping from constants to $\boldsymbol{D}$ such that constants in individual position are mapped to $\boldsymbol{D}_{ind}$ and constants in function positions are mapped to $\boldsymbol{D}_{func}$,
- $\boldsymbol{I}_V$ is a mapping from `Var` to $\boldsymbol{D}_{ind}$,
- $\boldsymbol{I}_{list}$ is an injective mapping from $\boldsymbol{D}_{ind}^{*}$ to $\boldsymbol{D}_{ind}$,
- $\boldsymbol{I}_{tail}$ is a mapping from $\boldsymbol{D}_{ind}^{+} \times \boldsymbol{D}_{ind}$ to $\boldsymbol{D}_{ind}$,
- $\boldsymbol{I}_{frame}$ is a mapping from $\boldsymbol{D}_{ind}$ to functions of the form `SetOfFiniteBags`($\boldsymbol{D}_{ind} \times \boldsymbol{D}_{ind}$) → $\boldsymbol{D}$,
- $\boldsymbol{I}_{sub}$ is a mapping from $\boldsymbol{D}_{ind} \times \boldsymbol{D}_{ind}$ to $\boldsymbol{D}$,
- $\boldsymbol{I}_{isa}$ is a mapping from $\boldsymbol{D}_{ind} \times \boldsymbol{D}_{ind}$ to $\boldsymbol{D}$, and
- $\boldsymbol{I}_{truth}$ is a mapping from $\boldsymbol{D}$ to $\boldsymbol{TV}$.

For the purpose of the interpretation of imported documents, RIF BLD defines the notion of ***semantic multi-structures***, which are nonempty sets of semantic structures of the form {$\boldsymbol{J}$,$\boldsymbol{I}$; $\boldsymbol{I}^{i1}$, $\boldsymbol{I}^{i2}$, ...} that differ only in interpretation of local constants. The structure $\boldsymbol{I}$ in the above is used to interpret document formulas, and will be used to specify RIF combinations.

3.2.1.2 RDF Lists

Syntactically speaking, an RDF list is a set of triples of the form

```
i1 rdf:first d1 .
i1 rdf:rest i2 .
...
in rdf:first dn .
in rdf:rest rdf:nil .
```

Here, `i1` ... `in` provide the structure of the linked list and `d1` ... `dn` are the items. The above list would be written in RIF syntax as `List(d1 ... dn)`.

Given an RDF interpretation I=< IR, IP, IEXT, IS, IL, LV >, we say that an element `l1` ∈ IR refers to an ***RDF list*** (y1,...,yn) if `l1`=IS(`rdf:nil`), in case n=0; otherwise, ∃ `l2`, ..., `ln` such that <`l1`,y1> ∈ IEXT(IS(rdf:first)), <`l1`,`l2`> ∈ IEXT(IS(rdf:rest)), ..., <`ln`,yn> ∈ IEXT(IS(rdf:first)), and <`ln`,IS(rdf:nil)> ∈ IEXT(IS(rdf:rest)).

Note that, if `n > 0`, there may be several lists referred to by `l1`, since there is no restriction, in general, on the `rdf:first` elements and the `rdf:rest` successors.

3.2.1.3 Common RIF-RDF Interpretations

Definition. A ***common-RIF-RDF-interpretation*** is a pair ($\hat{I}$, I), where $\hat{I}$ is a semantic multi-structure of the form {***J***,***I***; I^{i1}, I^{i2}, ...}, and I is a Simple interpretation of a Vocabulary *V*, such that the following conditions hold:

1. (IR union IP) = D_{ind};
2. IP is a superset of the set of all k in D_{ind} such that there exist some a, b in D_{ind} and $I_{truth}(I_{frame}(a)(k,b))$=**t**;
3. LV is a superset of (union of the value spaces of all considered datatypes);
4. IEXT(k) = the set of all pairs (a, b), with a, b, and k in D_{ind}, such that $I_{truth}(I_{frame}(a)(k,b))$=**t**;
5. IS(`i`) = I_C(<i>) for every IRI `i` in V_U;
6. IL((s, d)) = I_C("s"^^d) for every well-typed literal (s, d) in V_{TL};
7. IEXT(IS(`rdf:type`)) is equal to the set of all pairs (a, b) in $D_{ind} \times D_{ind}$ such that $I_{truth}(I_{isa}(a,b))$=**t**; and
8. IEXT(IS(`rdfs:subClassOf`)) is a superset of the set of all pairs (a, b) in $D_{ind} \times D_{ind}$ such that $I_{truth}(I_{sub}(a,b))$=**t**;
9. For any nonnegative integer n and any y1,...,yn ∈ IR, if some `l1` ∈ IR refers to the RDF list (y1,...,yn), then I_{list}(y1,...,yn)=`l1`; and
10. For any nonnegative integer n and any sequence of elements y1,...,yn ∈ IR, an element `l1` ∈ IR refers to the RDF list (y1,...,yn) iff I_{list}(y1,...,yn)=`l1`. ☐

Condition 1 ensures that the combination of resources and properties corresponds exactly to the RIF domain; note that if I is an RDF-, RDFS-, or D-interpretation, IP is a subset of IR, and thus IR=D_{ind}. Condition 2 ensures that the set of RDF properties at least includes all elements that are used as properties in frames in the RIF domain. Condition 3 ensures that all concrete values in D_{ind} are included in LV (by definition, the value spaces of all considered datatypes are included in D_{ind}). Condition 4 ensures that RDF triples are interpreted in the same way as frame formulas. Condition 5 ensures that IRIs are interpreted in the same way. Condition 6 ensures that typed literals are interpreted in the same way. Note that no correspondences are defined for the mapping of names in RDF that are not symbols of RIF, e.g., ill-typed literals and RDF URI references that are not absolute IRIs. Condition 7 ensures that typing in RDF and typing in RIF correspond, i.e., `a rdf:type b` is true iff `a # b` is true. Condition 8 ensures that whenever a RIF subclass statement holds, the corresponding RDF subclass statement holds as well, i.e., `a rdfs:subClassOf b` is true if `a ## b` is true. Finally, condition 9 requires the existence of an RIF list for every RDF list and condition 10 in addition requires the existence of an RDF list for every RIF list.

3.2.2 Satisfaction and Models

The notion of satisfiability refers to the conditions under which a common-RIF-RDF-interpretation ($\hat{I}$, I) is a model of a combination < *R*, **S**>. The notion of satisfiability is defined for all four entailment regimes of RDF (i.e., Simple, RDF, RDFS, and D). The definitions are all analogous. Intuitively, a common-RIF-RDF-interpretation ($\hat{I}$, I) satisfies a combination < *R*, **S**> if $\hat{I}$ is a model of *R* and I satisfies **S**. Formally:

Definition. A common-RIF-RDF-interpretation ($\hat{I}$, I) ***satisfies*** a RIF-RDF combination C=< *R*, **S** > if $\hat{I}$ is a model of *R* and I satisfies every RDF graph *S* in **S**; in this case ($\hat{I}$, I) is called a ***RIF-Simple-model***, or ***model***, of C, and C is ***satisfiable***. ($\hat{I}$, I) satisfies a generalized RDF graph *S* if I satisfies *S*. ($\hat{I}$, I) satisfies a condition formula φ if $TVal_{\hat{I}}(φ)=t$. ☐

RDF-, RDFS-, and RIF-D-satisfiability are defined through additional restrictions on I:

Definition. A model ($\hat{I}$, I) of a combination C is an ***RIF-RDF-model*** of C if I is an RDF-interpretation; in this case C is ***RIF-RDF-satisfiable***.

A model ($\hat{I}$, I) of a combination C is an ***RIF-RDFS-model*** of C if I is an RDFS-interpretation; in this case C is ***RIF-RDFS-satisfiable***.

Let ($\hat{I}$, I) be a model of a combination C and let D be a datatype map conforming with the set ***DTS*** of datatypes in ***I***. ($\hat{I}$, I) is a ***RIF-D-model*** of C if I is a D-interpretation; in this case C is ***RIF-D-satisfiable***. ☐

3.2.3 Entailment

Using the notions of models defined above, entailment is defined in the usual way, i.e., through inclusion of sets of models.

Definition. Let C be a RIF-RDF combination, let *S* be a generalized RDF graph, let φ be a condition formula, and let D be a datatype map conforming with the set of considered datatypes. C ***RIF-D-entails*** *S* if every RIF-D-model of C satisfies *S*. Likewise, C ***RIF-D-entails*** φ if every RIF-D-model of C satisfies φ. ☐

The other notions of entailment are defined analogously:

Definition. A combination C ***RIF-Simple-entails*** *S* (resp., φ) if every Simple model of C satisfies *S* (resp., φ).

A combination C ***RIF-RDF-entails*** *S* (resp., φ) if every RIF-RDF-model of C satisfies *S* (resp., φ).

A combination C ***RIF-RDFS-entails*** *S* (resp., φ) if every RIF-RDFS-model of C satisfies *S* (resp., φ). ☐

Note that simple entailment in combination with an empty ruleset is not the same as simple entailment in RDF, since certain entailments involving datatypes are enforced by the RIF semantics in combinations, cf. the example involving strings and plain literals above.

4 OWL Compatibility

This section specifies how a RIF document interacts with a set of OWL ontologies in a RIF-OWL combination. The semantics of combinations is defined for OWL 2 [OWL2-Syntax]. Since OWL 2 is an extension of OWL 1 [OWL-Reference], the specification in this section applies also to combinations of RIF documents with OWL 1 ontologies.

OWL 2 specifies two different variants of the language: OWL 2 DL [OWL2-Syntax] and OWL 2 Full [OWL2-RDF-Based-Semantics], where the latter are RDF graphs that use OWL Vocabulary; the RDF representation of an OWL 2 DL ontology is also an OWL 2 Full ontology. OWL 1 Lite and OWL 1 DL [OWL-Reference], which are sublanguages of OWL 1, can be seen as syntactical subsets of OWL 2 DL. OWL 2 ontologies may be interpreted under one of two semantics: the Direct Semantics [OWL2-Semantics], which is only defined for OWL 2 DL and is based on standard Description Logic semantics, and the RDF-Based Semantics [OWL2-RDF-Based-Semantics], which is defined for arbitrary OWL 2 Full ontologies.

The syntax of OWL 2 DL is defined in terms of a Structural Specification, and there is a mapping to an RDF representation for interchange. The RDF representation of OWL 2 DL [OWL2-RDF-Mapping] does not extend the RDF syntax, but rather restricts it: every OWL 2 DL ontology in RDF form is an RDF graph, but not every RDF graph is an OWL 2 DL ontology. OWL 2 Full and RDF have the same syntax: every RDF graph is an OWL 2 Full ontology and vice versa. This syntactical difference is reflected in the definition of RIF-OWL compatibility: combinations of RIF with OWL 2 DL are based on the OWL 2 Structural Specification, whereas combinations with OWL 2 Full are based on the RDF syntax.

Since the OWL 2 Full syntax is the same as the RDF syntax and the OWL 2 RDF-Based Semantics is an extension of the RDF Semantics, the definition of RIF-OWL 2 Full compatibility is an extension of RIF-RDF compatibility. However, defining RIF-OWL DL compatibility in the same way would entail losing certain properties of the Direct Semantics. One of the main reasons for this is the difference in the way classes and properties are interpreted in the RDF-Based and Direct Semantics. In the RDF-Based Semantics, classes and properties are interpreted as objects in the domain of interpretation, which are then associated with subsets of, respectively binary relations over the domain of interpretation, using the `rdf:type` property and the extension function IEXT, as in RDF. In the Direct Semantics, classes and properties are directly interpreted as subsets of, respectively binary relations over the domain. This is a key property of the first-order logic nature of Description Logic semantics and enables the use of Description Logic reasoning techniques for processing OWL 2 DL descriptions. Defining RIF-OWL DL compatibility as an extension of RIF-RDF compatibility would define a correspondence between OWL 2 DL statements and RIF frame formulas. Since RIF frame formulas are interpreted using an extension function, as in RDF, defining the correspondence between them and OWL 2 DL statements would change the semantics of OWL statements, even if the RIF document were empty.

A RIF-OWL combination that is faithful to the first-order nature of the OWL 2 Direct Semantics requires interpreting classes and properties as sets and binary relations, respectively, suggesting that a correspondence could be defined with unary and binary predicates. It is, however, also desirable that there be uniform syntax for the RIF component of both RIF-OWL 2 DL and RIF-RDF/OWL 2 Full combinations, because one may not know at the time of constructing the rules which type of inference will be used. Consider, for example, an RDF graph S consisting of the following statements

```
_:x rdf:type owl:Ontology .
a rdf:type C .
```

and a RIF document with the rule

```
Forall ?x (?x[rdf:type -> D] :- ?x[rdf:type -> C])
```

The combination of the two, according to the specification of RDF Compatibility, allows deriving the triple

```
a rdf:type D .
```

Now, the RDF graph S is also an OWL 2 DL ontology. Therefore, one would expect the triple to be implied according to the semantics of RIF-OWL DL combinations as well.

To ensure that the RIF-OWL DL combination is faithful to the OWL 2 Direct Semantics and to enable using the same, or similar, RIF rules in combinations with both OWL 2 DL and RDF/OWL 2 Full, the interpretation of frame formulas `s[p -> o]` in RIF-OWL DL combinations is slightly different from their interpretation in RIF and syntactical restrictions are imposed on the use of variables and function terms in frame formulas.

The remainder of this section formally defines combinations of RIF rules with OWL 2 DL and OWL 2 Full ontologies and the semantics of such combinations. A combination consists of a RIF document and a set of OWL ontologies. The semantics of combinations is defined in terms of combined models, which are pairs of RIF semantic multi-structures and OWL 2 Direct, respectively OWL 2 RDF-Based interpretations. The interaction between the structures and interpretations is defined through a number of conditions. Entailment is defined as model inclusion, as usual.

4.1 Syntax of RIF-OWL Combinations

Since RDF graphs and OWL 2 Full ontologies cannot be distinguished, the syntax of RIF-OWL 2 Full combinations is the same as the syntax of RIF-RDF combinations.

The syntax of OWL ontologies in RIF-OWL DL combinations is given by the Structural Specification of OWL 2 and the restrictions on OWL 2 DL ontologies [OWL2-Syntax]. Certain restrictions are imposed on the syntax of the RIF rules in combinations with OWL 2 DL. Specifically, the only terms allowed in class and property positions in class membership frame formulas are constant symbols. A DL-frame formula is a frame formula a[b_1 -> c_1 ... b_n -> c_n] such that $n \geq 1$ and for every b_i, with $1 \leq i \leq n$, it holds that b_i is a constant symbol and if b_i = `rdf:type`, then c_i is a constant symbol. A DL-class membership formula is a class membership formula a#b such that b is a constant symbol. A DL-subclass formula is a subclass formula b##c such that b and c are constant symbols.

We do not allow subclass formulas in rule conditions in RIF-OWL DL combinations, since at the time of writing there are no known effective and efficient ways of dealing with such subclass formulas in conditions in reasoners.

Definition. A condition formula φ is a ***DL-condition*** if every frame formula in φ is a DL-frame formula, every class membership formula in φ is a DL-class membership formula, and φ does not contain subclass formulas.

A RIF-BLD document formula *R* is a ***RIF-BLD DL-document formula*** if every frame formula in *R* is a DL-frame formula, every class membership formula in *R* is a DL-class membership formula, every subclass formula in *R* is a DL-subclass formula, and *R*does not contain any rules with subclass formulas.

A ***RIF-OWL DL-combination*** is a pair < *R*,**O**>, where *R* is a RIF-BLD DL-document formula and **O** is a set of OWL 2 DL ontologies of a Vocabulary *V* over an OWL 2 datatype map *D*. ☐

When clear from the context, RIF-OWL DL-combinations are referred to simply as *combinations*.

4.1.1 Safeness Restrictions

In the literature, several restrictions on the use of variables in combinations of rules and Description Logics have been identified [Motik05, Rosati06] for the purpose of decidable reasoning. This section specifies such safeness restrictions for RIF-OWL DL combinations.

Given a set of OWL 2 DL ontologies **O**, a variable ?x in a RIF rule **Q** *H* :- *B* is ***DL-safe*** if it occurs in an atomic formula in *B* that is not of the form *s*[*P* -> *o*] or *s*[rdf:type -> *A*], where *s*, *P*, *o*, and *A* are terms (possibly including ?x) and *P* or *A* occurs in one of the ontologies in **O**. A disjunction-free RIF rule **Q** (*H* :- *B*) is ***DL-safe***, given **O**, if every variable that occurs in *H* :- *B* is DL-safe. A disjunction-free RIF rule **Q** (*H* :- *B*) is ***weakly DL-safe***, given **O**, if every variable that occurs in *H* is DL-safe.

Definition. A RIF-OWL DL-combination <*R*,**O**> is ***DL-safe*** if every rule in *R* is DL-safe, given **O**. A RIF-OWL DL-combination <*R*,**O**> is ***weakly DL-safe*** if every rule in *R* is weakly DL-safe, given **O**. ☐

4.1.2 Datatypes in OWL 2

Compared with RDF and the RIF, OWL 2 uses a slightly extended notion of datatype.

In the remainder of this section, a datatype *d* contains, in addition to the lexical space, value space, and lexical-to-value mapping, a ***facet space***, which is a set of pairs of the form (*F*, *v*), where *F* is an IRI and *v* is a data value, and a ***facet-to-value mapping***, which is a mapping from facets to subsets of the value space of *d*.

An ***OWL 2 datatype map*** D is a datatype map that maps the IRIs of the datatypes specified in Section 4 of [OWL2-Syntax] to the corresponding datatypes such that the domain of D does not include `rdfs:Literal`.

We note here that the definitions of datatype and datatype map in the OWL 2 Direct Semantics specification [OWL2-Semantics] are somewhat different. There, a datatype is some entity with some associated IRIs, and the datatype map assigns lexical value, and facet spaces, as well as lexical-to-value and facet-to-value mappings. The definitions of datatype and datatype map we use are isomorphic, and, indeed, the same as in the OWL 2 RDF-Based Semantics specification [OWL2-RDF-Based-Semantics]. The latter does not preclude the use of `rdfs:Literal` in datatype maps. Note that we do not restrict the use of `rdfs:Literal` in OWL 2 ontologies or RDF graphs.

4.2 Semantics of RIF-OWL Combinations

The semantics of RIF-OWL 2 Full combinations is a straightforward extension of the Semantics of RIF-RDF Combinations.

The semantics of RIF-OWL 2 DL combinations cannot straightforwardly extend the semantics of RIF-RDF combinations, because the OWL 2 Direct Semantics does not extend the RDF Semantics. In order to keep the syntax of the rules uniform between RIF-OWL 2 Full and RIF-OWL DL combinations, the semantics of RIF frame formulas is slightly altered in RIF-OWL DL combinations.

4.2.1 OWL RDF-Based Semantics

Given an OWL 2 datatype map D and a Vocabulary *V* that includes the domain of D and the OWL 2 RDF-Based Vocabulary Vocabulary, a D-interpretation I is an ***OWL 2 RDF-Based Interpretation*** of *V* with respect to D if it satisfies the semantic conditions in Section 5 of [OWL2-RDF-Based-Semantics].

The semantics of RIF-OWL 2 Full combinations is a straightforward extension of the semantics of RIF-RDF combinations. It is based on the same notion of common interpretations, but defines additional notions of satisfiability and entailment.

Definition. Let ($\hat{I}$, I) be a common-RIF-RDF-interpretation that is a model of a RIF-RDF combination C=< *R*, **S** > and let D be an OWL 2 datatype map conforming with the set of datatypes in ***I***. ($\hat{I}$, I) is an ***RIF-OWL RDF-Based-model*** of C if I is an OWL 2 RDF-Based Interpretation with respect to D; in this case C is ***RIF-OWL RDF-Based-satisfiable*** with respect to D.

Let C be a RIF-RDF combination, let *S* be a generalized RDF graph, let φ be a condition formula, and let D be an OWL 2 datatype map D conforming with the set of considered datatypes. C ***RIF-OWL RDF-Based-entails*** *S* with respect to D if every RIF-OWL RDF-Based-model of C satisfies *S*. Likewise, C ***RIF-OWL RDF-Based-entails*** φ with respect to D if every RIF-OWL RDF-Based-model of C satisfies φ. ☐

4.2.2 OWL Direct Semantics

The semantics of RIF-OWL DL-combinations is similar in spirit to the semantics of RIF-RDF combinations. Analogous to common-RIF-RDF-interpretations, there is the notion of common-RIF-OWL Direct-interpretations, which are pairs of RIF and OWL 2 Direct interpretations, and which define a number of conditions that relate these interpretations to each other.

4.2.2.1 Modified Semantics for RIF Subclass, Membership, and Frame Formulas

The modification of the semantics of RIF subclass, membership, and frame formulas is achieved by modifying the respective mapping functions ($\boldsymbol{I}_{sub}$), ($\boldsymbol{I}_{isa}$) and ($\boldsymbol{I}_{frame}$). In addition, a new mapping function for constants ($\boldsymbol{I}_{C'}$) is used whenever constants appear in class or property positions.

Frame formulas of the form `s[rdf:type -> o]` and class membership formulas of the form `s#o` are interpreted as membership of `s` in the set denoted by `o` and frame formulas of the form `s[p -> o]`, where `p` is not `rdf:type`, as membership of the pair `(s, o)` in the binary relation denoted by `p`.

Definition. A ***dl-semantic structure*** is a tuple $\boldsymbol{I}$ = <$\boldsymbol{TV}$, $\boldsymbol{DTS}$, $\boldsymbol{D}$, $\boldsymbol{D}_{ind}$, $\boldsymbol{D}_{func}$, $\boldsymbol{I}_C$, $\boldsymbol{I}_{C'}$, $\boldsymbol{I}_V$, $\boldsymbol{I}_F$, $\boldsymbol{I}_{frame}$, $\boldsymbol{I}_{NF}$, $\boldsymbol{I}_{sub}$, $\boldsymbol{I}_{isa}$, $\boldsymbol{I}_=$, $\boldsymbol{I}_{external}$, $\boldsymbol{I}_{truth}$>, where

- $\boldsymbol{I}_{C'}$ is a mapping from `Const` to $\boldsymbol{D}$;
- $\boldsymbol{I}_{frame}$ is a mapping from $\boldsymbol{D}_{ind}$ to total functions of the form `SetOfFiniteBags`($\boldsymbol{D} \times \boldsymbol{D}$) → $\boldsymbol{D}$ such that for each pair `(u, v)` ∈ `SetOfFiniteBags`($\boldsymbol{D} \times \boldsymbol{D}$) it holds that if `u` ≠ $\boldsymbol{I}_{C'}$(`rdf:type`), then `v` ∈ $\boldsymbol{D}_{ind}$;
- $\boldsymbol{I}_{sub}$ is a mapping $\boldsymbol{D} \times \boldsymbol{D} \rightarrow \boldsymbol{D}$;
- $\boldsymbol{I}_{isa}$ is a mapping $\boldsymbol{D_{ind}} \times \boldsymbol{D} \rightarrow \boldsymbol{D}$;
- all other elements of the structure are defined as in RIF semantic structures.

The mapping $\boldsymbol{I}$ from terms to $\boldsymbol{D}$ is defined as follows:

- $\boldsymbol{I}$(`o[a1->v1 ... ak->vk]`) = $\boldsymbol{I}_{frame}(\boldsymbol{I}(o))(\{<\boldsymbol{I}_{C'}(a_1),\boldsymbol{I}_{a_1}(v_1)>, ..., <\boldsymbol{I}_{C'}(a_n),\boldsymbol{I}_{a_n}(v_n)>\})$, where $\boldsymbol{I}_{a_i}=\boldsymbol{I}_{C'}$ if a_i=`rdf:type`; otherwise $\boldsymbol{I}_{a_i}=\boldsymbol{I}$.
- $\boldsymbol{I}$(`c1##c2`) = $\boldsymbol{I}_{sub}(\boldsymbol{I}_{C'}(c1), \boldsymbol{I}_{C'}(c2))$.
- $\boldsymbol{I}$(`o#c`) = $\boldsymbol{I}_{isa}(\boldsymbol{I}(o), \boldsymbol{I}_{C'}(c))$
- the mapping of other terms is defined as in RIF semantic structures.

The truth valuation function $TVal_{I}$ is defined as in BLD semantic structures.

Dl-semantic multi-structures are defined analogous to RIF-BLD semantic multi-structures [RIF-BLD]. Formally, a ***dl-semantic multi-structure*** **Î** is a set of dl-semantic structures {$\boldsymbol{J}$,$\boldsymbol{I}$; $\boldsymbol{I}^{i1}$, $\boldsymbol{I}^{i2}$, ...}, where

- $\boldsymbol{I}$ and $\boldsymbol{J}$ are dl-semantic structures; and
- $\boldsymbol{I}^{i1}$, $\boldsymbol{I}^{i2}$, etc., are dl-semantic structures ***adorned*** with the locators of *distinct* RIF-BLD formulas (one can think of these adorned structures as locator-structure pairs).

All the structures in **Î** (adorned and non-adorned) are identical in all respects except for the following:

- The mappings $\boldsymbol{J}_C$, $\boldsymbol{I}_C$, $\boldsymbol{I}_C^{i1}$, $\boldsymbol{I}_C^{i2}$, ... may differ on the constants in `Const` that belong to the `rif:local` symbol space.
- The mappings $\boldsymbol{J}_V$, $\boldsymbol{I}_V$, $\boldsymbol{I}_V^{i1}$, $\boldsymbol{I}_V^{i2}$ may differ.

The truth valuation function $TVal_{Î}$ is defined as in BLD semantic structures.

Definition. A dl-semantic multi-structure ***Î*** is a ***model*** of a RIF-BLD DL-document formula *R* if $TVal_{Î}(R)$=**t**. ☐

4.2.2.2 Semantics of RIF-OWL DL Combinations

As defined in [OWL2-Semantics], an ***interpretation*** for a Vocabulary *V* over a datatype map D is a tuple *I*=< IR, LV, C, OP, DP, I, DT, LT, FA >, where

- IR is a non-empty set, called the object domain,
- LV is a non-empty set, called the data domain, which includes all value spaces of the datatypes in the range of D,
- C is a mapping from classes to subsets of IR,
- OP is a mapping from object properties to subsets of IR × IR,
- DP is a mapping from object properties to subsets of IR × LV,
- I is a mapping from individuals into IR,
- DT is a mapping from datatypes to subsets of LV,
- LT is a mapping from typed literals in *V* into LV, and
- FA is a mapping from IRI, literal pairs to subsets of value spaces in D.

The OWL semantics imposes a number of further restrictions on the mapping functions to ensure the interpretation of datatypes, literals, and facets conforms with the given datatype map D and to define the semantics of built-in classes and properties (e.g., `owl:Thing`). The mappings DT, LT, and FA are essentially given by the datatype map.

Definition. Given a Vocabulary *V* over an OWL 2 datatype map D, a ***common-RIF-OWL Direct-interpretation*** for *V* over D is a pair (***Î***, *I*), where ***Î*** is a dl-semantic multi-structure of the form {$\boldsymbol{J}$,$\boldsymbol{I}$; $\boldsymbol{I}^{i1}$, $\boldsymbol{I}^{i2}$, ...}, and *I* is an interpretation for *V* over D, such that the following conditions hold.

1. D is conforming with the datatypes in $\boldsymbol{I}$;
2. (IR union LV) is $\boldsymbol{D_{ind}}$;
3. C(c) is the set of all objects k such that $\boldsymbol{I}_{truth}(\boldsymbol{I}_{frame}(k)(\{(\boldsymbol{I}_{C'}(\texttt{rdf:type}),\boldsymbol{I}_{C'}(\texttt{<c>}))\}))$ = **t**, for every IRI c identifying a class in *V*;
4. DT(c) is the set of all objects k such that $\boldsymbol{I}_{truth}(\boldsymbol{I}_{frame}(k)(\{(\boldsymbol{I}_{C'}(\texttt{rdf:type}),\boldsymbol{I}_{C'}(\texttt{<c>}))\}))$ = **t**, for every IRI c identifying a datatype in *V*;
5. OP(p) is the set of all pairs (k, l) such that $\boldsymbol{I}_{truth}(\boldsymbol{I}_{frame}(k)(\{(\boldsymbol{I}_{C'}(\texttt{<p>}), l)\}))$ = **t** (true), for every IRI p identifying an object property in *V*;
6. DP(p) is the set of all pairs (k, l) such that $\boldsymbol{I}_{truth}(\boldsymbol{I}_{frame}(k)(\{(\boldsymbol{I}_{C'}(\texttt{<p>}), l)\}))$ = **t** (true), for every IRI p identifying a data property in *V*;
7. I(i) = $\boldsymbol{I}_C$(<i>) for every IRI i identifying an individual in *V*;
8. C(c) is the set of all objects k such that $\boldsymbol{I}_{truth}(\boldsymbol{I}_{isa}(k,\boldsymbol{I}_{C'}(\texttt{<c>})))$ = **t**, for every IRI c identifying a class in *V*;
9. DT(c) is the set of all objects k such that $\boldsymbol{I}_{truth}(\boldsymbol{I}_{isa}(k,\boldsymbol{I}_{C'}(\texttt{<c>})))$ = **t**, for every IRI c identifying a datatype in *V*;
10. C(c) is a subset of C(d) whenever $\boldsymbol{I}_{truth}(\boldsymbol{I}_{sub}(\boldsymbol{I}_{C'}(\texttt{<c>}),\boldsymbol{I}_{C'}(\texttt{<d>})))$ = **t**, for any two IRIs c and d identifying classes in *V*. ☐

Condition 2 ensures that the relevant parts of the domains of interpretation are the same. Conditions 3 and 4 ensures that the interpretation (extension) of an OWL class or datatype identified by an IRI u corresponds to the interpretation of frames of the form `?x[rdf:type -> <u>]`. Conditions 5 and 6 ensure that the interpretation (extension) of an OWL object or data property identified by an IRI u corresponds to the interpretation of frames of the form `?x[<u> -> ?y]`. Condition 7 ensures that individual identifiers in the OWL ontologies and the RIF documents are interpreted in the same way. Conditions 8 and 9 ensure that typing in OWL and typing in RIF correspond, i.e., `ClassAssertion(b a)` is true iff `a # b` is true. Finally, 10 ensures that whenever a RIF subclass statement holds, the corresponding OWL subclass statement holds as well, i.e., `SubClassOf(a b)` is true if `a ## b` is true.

Using the definition of common-RIF-OWL Direct-interpretation, satisfaction, models, and entailment are defined in the usual way:

Definition. A common-RIF-OWL Direct-interpretation (***Î***, *I*) for a Vocabulary *V* over an OWL 2 datatype map D is an ***RIF-OWL Direct-model*** of a RIF-OWL DL-combination C=< *R*, **O** > if ***Î*** is a model of *R* and *I* is a model of every ontology *O* in **O**; in this case C is ***RIF-OWL Direct-satisfiable*** for *V* over D. (***Î***, *I*) is an ***RIF-OWL Direct-model*** of an OWL 2 DL ontology *O* if *I* is a model of *O*. (***Î***, *I*) is an ***RIF-OWL DL-model*** of a DL-condition formula φ if $TVal_{Î}$(φ)=**t**.

Let C be a RIF-OWL DL-combination, let *O* be an OWL 2 DL ontology, let φ be a DL-condition formula, and let D be an OWL 2 datatype map conforming with the set of considered datatypes, and let *V* be a Vocabulary over D for every ontology in C and for *O*. C ***RIF-OWL Direct-entails*** *O* with respect to D if every common-RIF-OWL Direct-interpretation for *V* over D that is an RIF-OWL Direct-model of C is an RIF-OWL Direct-model of *O*. Likewise, C ***RIF-OWL Direct-entails*** φ with respect to D if every common-RIF-OWL Direct-interpretation for *V* over D that is an RIF-OWL Direct-model of C is an RIF-OWL Direct-model of φ. ☐

Example. In the OWL 2 Direct Semantics, the domains for interpreting individuals respectively, literals (data values), are disjoint. The disjointness entails that data values cannot be members of a class and individuals cannot be members of a datatype.

RIF does not make such distinctions; variable quantification ranges over the entire domain. So, the same variable may be assigned to an abstract individual or a concrete data value. Additionally, RIF constants (e.g., IRIs) denoting individuals can be written in place of a data value, such as the value of a data-valued property or in datatype membership statements; similarly for constants denoting data values. Such statements cannot be satisfied in any common-RIF-OWL Direct-interpretation. The following example illustrates several such statements.

Consider the datatype `xs:string` and a RIF-OWL DL combination consisting of the set containing only an OWL 2 DL ontology that contains

```
ex:myiri rdf:type ex:A .
```

and a RIF document containing the following fact

```
ex:myiri[rdf:type -> xs:string]
```

This combination is not RIF-OWL Direct-satisfiable, because ex:myiri is an individual identifier and S maps individual identifiers to elements in O, which is disjoint from the elements in the datatype `xs:string`.

Consider a RIF-OWL DL combination consisting of the set containing only the OWL 2 DL ontology

```
ex:hasChild rdf:type owl:ObjectProperty .
```

and a RIF document containing the following fact

```
ex:myiri[ex:hasChild -> "John"]
```

This combination is not RIF-OWL Direct-satisfiable, because ex:hasChild is an object property, and values of object properties may not be concrete data values.

Consider a RIF-OWL DL combination consisting of the OWL DL ontology

```
SubClassof(ex:A ex:B)
```

and a RIF document containing the following rule

```
Forall ?x (?x[rdf:type -> ex:A])
```

This combination is not RIF-OWL Direct-satisfiable, because the rule requires every element, including every concrete data value, to be a member of the class `ex:A`. However, since every OWL interpretation requires every member of `ex:A` to be an element of the object domain, concrete data values cannot be members of the object domain. ☐

5 Importing RDF and OWL in RIF

In the preceding sections, RIF-RDF Combinations and RIF-OWL combinations were defined in an abstract way, as pairs consisting of a RIF document and a set of RDF graphs/ OWL ontologies. In addition, different semantics were specified based on the various RDF and OWL entailment regimes. RIF provides a mechanism for explicitly referring to (importing) RDF graphs from documents and specifying the intended profile (entailment regime) through the use of `Import` statements.

This section specifies how RIF documents with such import statements must be interpreted.

A RIF document contains a number of `Import` statements. Unary `Import` statements are used for importing RIF documents, and the interpretation of these statements is defined in Section 3.5 of [RIF-BLD]. This section defines the interpretation of binary `Import` statements:

```
Import(<t1> <p1>)
  ...
Import(<tn> <pn>)
```

Here, `ti` is an absolute IRI referring to an RDF graph to be imported and `pi` is an absolute IRI denoting the profile to be used for the import.

The profile determines which notions of model, satisfiability and entailment must be used. For example, if a RIF document *R* imports an RDF graph *S* with the profile *RDFS*, the notions of RIF-RDFS-model, RIF-RDFS-satisfiability, and RIF-RDFS-entailment must be used for the combination <*R*, {*S*}>.

Profiles are ordered as specified in Section 5.1.1. If several graphs are imported in a document, and these imports specify different profiles, the highest of these profiles is used. For example, if a RIF document *R* imports an RDF graph S_1 with the profile *RDF* and an RDF graph S_2 with the profile *OWL RDF-Based*, the notions of RIF-OWL RDF-Based-model, RIF-OWL RDF-Based-satisfiability, and RIF-OWL RDF-Based-entailment must be used with the combination <*R*, {S_1, S_2}>.

Finally, if a RIF document *R* imports an RDF graph *S* with the profile *OWL Direct*, *R* must be a RIF-BLD DL-document formula, *S* must be the RDF representation of an OWL 2 DL ontology *O*, and the notions of RIF-OWL Direct-model, RIF-OWL Direct-satisfiability, and RIF-OWL Direct-entailment must be used with the combination <*R*, {*O*}>.

5.1 Profiles of Imports

RIF defines specific profiles for the different notions of model, satisfiability and entailment of combinations, as well as one generic profile. The use of a specific profile specifies how a combination should be interpreted. If a specific profile cannot be handled by a receiver, the combination should be rejected. The use of a generic profile implies that a receiver may interpret the combination to the best of its ability.

The use of profiles is not restricted to the profiles specified in this document. Any specific profile that is used with RIF must specify an IRI that identifies it, as well as associated notions of model, satisfiability, and entailment for combinations.

5.1.1 Specific Profiles

The following table lists the specific profiles defined by RIF, the IRIs of these profiles, and the notions of model, satisfiability, and entailment that must be used with the profile.

Specific profiles in RIF.

Profile	IRI of the Profile	Model	Satisfiability	Entailment
Simple	http://www.w3.org/ns/entailment/Simple	RIF-Simple-model	satisfiability	RIF-Simple-entailment
RDF	http://www.w3.org/ns/entailment/RDF	RIF-RDF-model	RIF-RDF-satisfiability	RIF-RDF-entailment
RDFS	http://www.w3.org/ns/entailment/RDFS	RIF-RDFS-model	RIF-RDFS-satisfiability	RIF-RDFS-entailment
D	http://www.w3.org/ns/entailment/D	RIF-D-model	RIF-D-satisfiability	RIF-D-entailment
OWL Direct	http://www.w3.org/ns/entailment/OWL-Direct	RIF-OWL Direct-model	RIF-OWL Direct-satisfiability	RIF-OWL Direct-entailment

OWL RDF-Based	http://www.w3.org/ns/entailment/OWL-RDF-Based	RIF-OWL RDF-Based-model	RIF-OWL RDF-Based-satisfiability	RIF-OWL RDF-Based-entailment

Profiles that are defined for combinations of DL-document formulas and OWL ontologies in abstract syntax form are called *DL profiles*. Of the mentioned profiles, the profile *OWL Direct* is a DL profile.

The profiles are ordered as follows, where '<' reads "is lower than":

Simple < RDF < RDFS < D < OWL RDF-Based

OWL Direct < OWL RDF-Based

5.1.2 Generic Profile

RIF specifies one generic profile. The use of the generic profile does not imply the use of a specific notion of model, satisfiability, and entailment.

Generic profile in RIF.

Profile	IRI of the Profile
Generic	<http://www.w3.org/2007/rif-import-profile#Generic>

5.2 Interpretation of Profiles

Let *R* be a RIF document such that

```
Import(<u1> <p1>)
  ...
Import(<un> <pn>)
```

are all the two-ary import statements in *R* and the documents imported into *R* and let `Profile` be the set of profiles corresponding to the IRIs `p1,...,pn`.

If `pi, 1 ≤ i ≤ n`, corresponds to a DL profile and `ui` refers to an RDF graph that is not the RDF representation of an OWL (2) DL ontology, the document should be rejected.

If `ui, 1 ≤ i ≤ n`, refers to an RDF graph that uses a typed literal of the form `"s"^^rif:iri` or `"s"^^rdf:PlainLiteral`, the document must be rejected.

If `Profile` contains only specific profiles, then:

- If `Profile` does not have a single highest profile, the document must be rejected.
- If `Profile` contains only DL profiles and *R* is not a DL-document formula, it must be rejected.
- If `Profile` contains only DL profiles and the RDF graphs referred to by $u_1,...,u_n$ are RDF representations of the OWL 2 ontologies $O_1,....,O_n$, then the combination C=<*R*,{$O_1,....,O_n$}> must be interpreted according to the highest among the profiles in `Profile`.
- Otherwise, the combination C=<*R*,{$S_1,....,S_n$}>, where $S_1,....,S_n$ are the RDF graphs referred to by $u_1,...,u_n$, must be interpreted according to the highest among the profiles in `Profile`.

If `Profile` contains a generic profile, then the combination C=<*R*,{$S_1,....,S_n$}>, where $S_1,....,S_n$ are the RDF graphs referred to by $u_1,...,u_n$, may be interpreted according to the highest among the specific profiles in `Profile`, if there is one.

6 Conformance Clauses

We define notions of conformance for RIF-RDF and RIF-OWL combinations. We define these notions both for the RIF Core [RIF-Core] and RIF BLD [RIF-BLD] dialects.

Conformance is described in terms of semantics-preserving transformations between the native syntax of a compliant processor and the XML syntax of RIF Core and BLD.

We say that an RDF graph *S* is a *standard RDF graph* if for every triple `s p o` in *S*, `s` is an IRI or blank node, `p` is an IRI, and `o` is an IRI, literal, or blank node. A combination < *R*, **S** > is *standard* if every graph in **S** is standard.

Each RIF processor has sets T, of supported datatypes and symbol spaces that include the symbol spaces listed in [RIF-DTB], and E, of supported external terms that include the built-ins listed in [RIF-DTB]. The datatype map of a RIF processor is the smallest datatype map conforming with the set of datatypes in T.

Now, let P ∈ {Simple, RDF, RDFS, D, OWL RDF-Based} be a specific RDF profile. A RIF-RDF combination C=< *R*, **S** > is a $BLD_{T,E}$-P combination if *R* is a $BLD_{T,E}$ formula and C is a $Core_{T,E}$-P combination if *R* is a $Core_{T,E}$ formula.

A RIF-OWL DL-combination C=< *R*, **O** > is a $BLD_{T,E}$-OWL Direct combination if *R* is a $BLD_{T,E}$ formula and C is a $Core_{T,E}$-OWL Direct combination if *R* is a $Core_{T,E}$ formula.

A RIF processor is a ***conformant*** $BLD_{T,E}$-P ***consumer***, for P ∈ {Simple, RDF, RDFS, D, OWL Direct, OWL RDF-Based}, iff it implements a *semantics-preserving mapping*, μ, from the set of standard $BLD_{T,E}$-P combinations, standard RDF graphs, OWL 2 ontologies, and $BLD_{T,E}$ formulas to the language *L* of the processor (μ does not need to be an "onto" mapping) and, in case P ∈ {OWL Direct, OWL RDF-Based}, its datatype map is an OWL 2 datatype map.

We say that a RIF document *R* is list-safe if *R* is safe and it contains no occurrences of `rdf:first`, `rdf:rest`, or `rdf:nil` in rule consequents. An RDF graph *S* is list-safe if it contains no occurrences of `rdf:first` or `rdf:rest` outside of the property positions, it contains no occurrences of `rdf:nil` outside of triples of the form `... rdf:rest rdf:nil`, and there are no two triples `s rdf:first o1 . s rdf:first o2 .` or `s rdf:rest o1 . s rdf:rest o2 .` in *S*, where `s, o1, o2` are RDF terms and o1≠o2. A combination < *R*, **S** > is ***list-safe*** if *R* is list-safe and the merge of the graphs in **S** is list-safe.

A RIF processor is a ***conformant*** $Core_{T,E}$-P ***consumer***, for P ∈ {Simple, RDF, RDFS, D, OWL Direct, OWL RDF-Based}, iff it implements a *semantics-preserving mapping*, μ, from the set of standard list-safe $Core_{T,E}$-P combinations, standard RDF graphs, OWL 2 ontologies, and $Core_{T,E}$ formulas to the language *L* of the processor (μ does not need to be an "onto" mapping) and, in case P ∈ {OWL Direct, OWL RDF-Based}, its datatype map is an OWL 2 datatype map.

Formally, this means that for any pair (φ, ψ), where φ is a $BLD_{T,E}$-P combination and ψ is an RDF graph, OWL 2 ontology, or $BLD_{T,E}$ formula such that $φ \models_P ψ$ is defined, $φ \models_P ψ$ iff $μ(φ) \models_L μ(ψ)$. Here $\models_P$ denotes P-entailment and $\models_L$ denotes the logical entailment in the language *L* of the RIF processor.

A RIF processor is a ***conformant*** $BLD_{T,E}$-P ***producer*** iff it implements a *semantics-preserving mapping*, ν, from the language *L* of the processor to the set of all $BLD_{T,E}$ formulas, RDF graphs, OWL 2 ontologies, and $BLD_{T,E}$-P combinations (ν does not need to be an "onto" mapping).

A RIF processor is a ***conformant*** $Core_{T,E}$-P ***producer*** iff it implements a *semantics-preserving mapping*, ν, from the language *L* of the processor to the set of all $Core_{T,E}$ formulas, RDF graphs, OWL 2 ontologies, and $Core_{T,E}$-P combinations (ν does not need to be an "onto" mapping).

Formally, this means that for any pair (φ, ψ) of formulas in *L* such that $φ \models_L ψ$ is defined, $φ \models_L ψ$ iff $ν(φ) \models_P ν(ψ)$. Here $\models_P$ denotes P-entailment and $\models_L$ denotes the logical entailment in the language *L* of the RIF processor.

7 Acknowledgements

This document is the product of the Rules Interchange Format (RIF) Working Group (see below), the members of which deserve recognition for their time and commitment to RIF. The editors extend special thanks to: Mike Dean, Michael Kifer, Stella Mitchell, Axel Polleres, and Dave Reynolds, for their thorough reviews and insightful discussions; the working group chairs, Chris Welty and Christian de Sainte-Marie, for their invaluable technical help and inspirational leadership; and W3C staff contact Sandro Hawke, a constant source of ideas, help, and feedback.

W3C Editor's Draft

The following members of the joint RIF-OWL task force have contributed to the OWL Compatibility section in this document: Mike Dean, Peter F. Patel-Schneider, and Ulrike Sattler.

The regular attendees at meetings of the Rule Interchange Format (RIF) Working Group at the time of the publication were: Adrian Paschke (Freie Universitaet Berlin), Axel Polleres (DERI), Chris Welty (IBM), Christian de Sainte Marie (IBM), Dave Reynolds (HP), Gary Hallmark (ORACLE), Harold Boley (NRC), Jos de Bruijn (FUB), Leora Morgenstern (IBM), Michael Kifer (Stony Brook), Mike Dean (BBN), Sandro Hawke (W3C/MIT), and Stella Mitchell (IBM).

8 References

8.1 Normative References

[OWL2-RDF-Based-Semantics]
OWL 2 Web Ontology Language: RDF-Based Semantics, M. Schneider, Editor, W3C Candidate Recommendation, 27 October 2009, http://www.w3.org/TR/2009/REC-owl2-rdf-based-semantics-20091027/. Latest version available at http://www.w3.org/TR/owl2-rdf-based-semantics/.

[OWL2-Semantics]
OWL 2 Web Ontology Language: Direct Semantics, B. Motik, P. F. Patel-Schneider, B. Cuenca Grau, Editors, W3C Candidate Recommendation, 27 October 2009, http://www.w3.org/TR/2009/REC-owl2-direct-semantics-20091027/. Latest version available at http://www.w3.org/TR/owl2-direct-semantics/.

[OWL2-Syntax]
OWL 2 Web Ontology Language: Structural Specification and Functional-Style Syntax, B. Motik, P. F. Patel-Schneider, B. Parsia, Editors, W3C Candidate Recommendation, 27 October 2009, http://www.w3.org/TR/2009/REC-owl2-syntax-20091027/. Latest version available at http://www.w3.org/TR/owl2-syntax/.

[RDF-Concepts]
Resource Description Framework (RDF): Concepts and Abstract Syntax, G. Klyne, J. Carrol, Editors, W3C Recommendation, 10 February 2004, http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/. Latest version available at http://www.w3.org/TR/rdf-concepts/.

[RDF-Semantics]
RDF Semantics, P. Hayes, Editor, W3C Recommendation, 10 February 2004, http://www.w3.org/TR/2004/REC-rdf-mt-20040210/. Latest version available at http://www.w3.org/TR/rdf-mt/.

[RIF-Core]
RIF Core Dialect Harold Boley, Gary Hallmark, Michael Kifer, Adrian Paschke, Axel Polleres, Dave Reynolds, eds. W3C Editor's Draft, 28 October 2012, http://www.w3.org/2005/rules/wg/draft/ED-rif-core-20121028/. Latest version available at http://www.w3.org/2005/rules/wg/draft/rif-core/.

[RIF-BLD]
RIF Basic Logic Dialect Harold Boley, Michael Kifer, eds. W3C Editor's Draft, 28 October 2012, http://www.w3.org/2005/rules/wg/draft/ED-rif-bld-20121028/. Latest version available at http://www.w3.org/2005/rules/wg/draft/rif-bld/.

[RIF-DTB]
RIF Datatypes and Built-Ins 1.0 Axel Polleres, Harold Boley, Michael Kifer, eds. W3C Editor's Draft, 28 October 2012, http://www.w3.org/2005/rules/wg/draft/ED-rif-dtb-20121028/. Latest version available at http://www.w3.org/2005/rules/wg/draft/rif-dtb/.

8.2 Informational References

[CURIE]
CURIE Syntax 1.0, M. Birbeck, S. McCarron, W3C Candidate Recommendation, 16 January 2009, http://www.w3.org/TR/2009/CR-curie-20090116. Latest version available at http://www.w3.org/TR/curie/.

[DLP]
Description Logic Programs: Combining Logic Programs with Description Logics, B. Grosof, R. Volz, I. Horrocks, S. Decker. In Proc. of the 12th International World Wide Web Conference (WWW 2003), 2003.

[Motik05]
Query Answering for OWL-DL with rules, B. Motik, U. Sattler, R. Studer, Journal of Web Semantics 3(1): 41-60, 2005.

[OWL-Reference]
OWL Web Ontology Language Reference, M. Dean, G. Schreiber, Editors, W3C Recommendation, 10 February 2004, http://www.w3.org/TR/2004/REC-owl-ref-20040210/. Latest version available at http://www.w3.org/TR/owl-ref/.

[OWL-Semantics]
OWL Web Ontology Language Semantics and Abstract Syntax, P. F. Patel-Schneider, P. Hayes, I. Horrocks, Editors, W3C Recommendation, 10 February 2004, http://www.w3.org/TR/2004/REC-owl-semantics-20040210/. Latest version available at http://www.w3.org/TR/owl-semantics/.

[OWL2-Profiles]
OWL 2 Web Ontology Language: Profiles, B. Motik, B. Cuenca Grau, I. Horrocks, Z. Wu, A. Fokoue, C. Lutz, Editors, W3C Recommendation, 27 October 2009, http://www.w3.org/TR/2009/REC-owl2-profiles-20091027/. Latest version available at http://www.w3.org/TR/owl2-profiles/.

[OWL2-RDF-Mapping]
OWL 2 Web Ontology Language: Mapping to RDF Graphs, P. F. Patel-Schneider, B. Motik, Editors, W3C Recommendation, 27 October 2009, http://www.w3.org/TR/2009/REC-owl2-mapping-to-rdf-20091027/. Latest version available at http://www.w3.org/TR/owl2-mapping-to-rdf/.

[RDF-Schema]
RDF Vocabulary Description Language 1.0: RDF Schema, D. Brickley, R.V. Guha, Editors, W3C Recommendation, 10 February 2004, http://www.w3.org/TR/2004/REC-rdf-schema-20040210/. Latest version available at http://www.w3.org/TR/rdf-schema/.

[RDF-Syntax]
RDF/XML Syntax Specification (Revised), D. Beckett, Editor, W3C Recommendation, 10 February 2004, http://www.w3.org/TR/2004/REC-rdf-syntax-grammar-20040210/. Latest version available at http://www.w3.org/TR/rdf-syntax-grammar/.

[RFC-3066]
RFC 3066 - Tags for the Identification of Languages, H. Alvestrand, IETF, January 2001. This document is at http://www.ietf.org/rfc/rfc3066.

[RIF-PRD]
RIF Production Rule Dialect Christian de Sainte Marie, Gary Hallmark, Adrian Paschke, eds. W3C Editor's Draft, 28 October 2012, http://www.w3.org/2005/rules/wg/draft/ED-rif-prd-20121028/. Latest version available at http://www.w3.org/2005/rules/wg/draft/rif-prd/.

[RIF-UCR]
RIF Use Cases and Requirements Adrian Paschke, David Hirtle, Allen Ginsberg, Paula-Lavinia Patranjan, Frank McCabe, eds. W3C Editor's Draft, 18 December 2008, http://www.w3.org/2005/rules/wg/draft/ED-rif-ucr-20081218/. Latest version available at http://www.w3.org/2005/rules/wg/draft/rif-ucr/.

[Rosati06]
DL+log: Tight Integration of Description Logics and Disjunctive Datalog, R. Rosati, Proceedings of the 10th International Conference on Principles of Knowledge Representation and Reasoning, pp. 68-78, 2005.

[SWRL]
SWRL: A Semantic Web Rule Language Combining OWL and RuleML, I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B., M. Dean, W3C Member Submission, 21 May 2004, http://www.w3.org/Submission/2004/SUBM-SWRL-20040521/. Latest version available at http://www.w3.org/Submission/SWRL/.

[Turtle]
Turtle - Terse RDF Triple Language, D. Beckett, T. Berners-Lee, W3C Team Submission, 14 January 2008, http://www.w3.org/TeamSubmission/2008/SUBM-turtle-20080114/. Latest version available at http://www.w3.org/TeamSubmission/turtle/.

[XML Schema Datatypes]
W3C XML Schema Definition Language (XSD) 1.1 Part 2: Datatypes. David Peterson, Shudi Gao, Ashok Malhotra, C. M. Sperberg-McQueen, and Henry S. Thompson, eds. W3C Recommendation, 5 April 2012, http://www.w3.org/TR/2012/REC-xmlschema11-2-20120405/. Latest version available as http://www.w3.org/TR/xmlschema11-2/.

9 Appendix: Embeddings (Informative)

RIF-RDF combinations can be embedded into RIF documents in a fairly straightforward way, thereby demonstrating how a RIF-compliant translator without native support for RDF can process RIF-RDF combinations.

RIF-OWL combinations cannot be embedded in RIF, in the general case. However, there is a subset of OWL 2 DL, namely the OWL 2 RL profile [OWL2-Profiles], for which RIF-OWL combinations that can be embedded.

Simple, RDF, RDFS and OWL 2 RL entailment for RIF-RDF combinations are embedded in RIF BLD.

Note that Simple, RDF and RDFS entailments are superficially embeddable within RIF Core. However, condition 7 of the semantics of RIF-RDF combinations cannot be axiomatized in RIF Core due to restrictions on the use isa (#) in rule heads. OWL 2 RL is not embeddable in RIF Core due the the need for equality reasoning.

The embeddings are defined using an embedding function tr that maps symbols, triples, and RDF graphs/OWL ontologies to RIF symbols, statements, and documents, respectively.

To embed consistency checking in RDF(S) and OWL, we use a special 0-ary predicate symbol `rif:error`, which is assumed not to be used in the RIF documents in the combination.

Besides the namespace prefixes defined in the Overview, the following namespace prefix is used in this appendix: pred refers to the RIF namespace for built-in predicates `http://www.w3.org/2007/rif-builtin-predicate#` [RIF-DTB].

To facilitate the definition of the embeddings we define the notion of a *merge* of RIF formulas.

Definition. Let **R**={R_1,...,R_n} be a set of document, group, and rule formulas, such that there are no prefix or base directives, or relative IRIs in **R** and *directive$_{11}$*, ..., *directive$_{nm}$* are all the import directives occurring in document formulas in **R**. The ***merge*** of **R**, denoted merge(**R**), is defined as `Document(`*directive$_{11}$* ... *directive$_{nm}$* `Group(`R^*_1 ... R^*_n`))`, where R^*_i is obtained from R_i in the following way:

- if R_i is a document formula of the form `Document(`*directive$_{i1}$* ... *directive$_{im}$* Γ`)`, then R^*_i=Γ and
- if R_i is a non-document formula (i.e., fact, rule, or group), then R^*_i=R_i. ☐

Note that the requirement that no prefix or based directives, or relative IRIs are included in any of the formulas to be merged is not a limitation, since compact IRIs can be rewritten to absolutes IRIs, as can relative IRIs, by exploiting prefix and base directives, and the location of the document.

9.1 Embedding RIF-RDF Combinations

RIF-RDF combinations are embedded by combining the RIF rules with embeddings of the RDF graphs and an axiomatization of Simple, RDF, and RDFS entailment.

The embedding is not defined for combinations that include infinite RDF graphs and for combinations that include RDF graphs with RDF URI references that are not absolute IRIs (see the End note on RDF URI references) or plain literals without language tags that are not in the lexical space of the `xs:string` datatype [XML-Schema2]. Also, the embedding is not defined for RDF lists.

We define a list-free combination as a combination that does not contain any mention of the symbols `rdf:first`, `rdf:rest`, or `rdf:nil`.

In the remainder of this section we first define the embedding of symbols, triples, and graphs, after which we define the axiomatization of Simple, RDF, and RDFS entailment of combinations and, finally, demonstrate faithfulness of the embeddings.

9.1.1 Embedding Symbols

Given a combination C=< *R*,**S**>, the function tr maps RDF symbols of a Vocabulary *V* and a set of blank nodes *B* to RIF symbols, as defined in the following table. It is assumed that the Vocabulary *V* includes all the IRIs and literals used in the RIF documents and condition formulas under consideration.

In the table, the mapping tr' is an injective function that maps typed literals to new constants in the `rif:local` symbol space, where a new constant is a constant that is not used in the document or its vicinity (i.e., imported or entailed formula, or entailing combination). It "generates" a new constant from a typed literal.

Mapping RDF symbols to RIF.

RDF Symbol	RIF Symbol	Mapping
IRI `i` in V_U	Constant with symbol space `rif:iri`	tr(i) = <i>
Blank node `_:x` in *B*	Variable symbol ?x	tr(_:x) = ?x
Plain literal without a language tag xxx in V_{PL}	Constant with the datatype `xs:string`	tr("xxx") = "xxx"
Plain literal with a language tag "xxx"@lang in V_{PL}	Constant with the datatype `rdf:PlainLiteral`	tr("xxx"@lang) = "xxx@lang"^^`rdf:PlainLiteral`
Well-typed literal "s"^^u in V_{TL}	Constant with the symbol space u	tr("s"^^u) = "s"^^u
Non-well-typed literal "s"^^u in V_{TL}	Local constant `s-u'` that is not used in C and is obtained from "s"^^u	tr("s"^^u) = tr'("s"^^u)

9.1.2 Embedding Triples and Graphs

This section extends the mapping function tr to triples and defines two embedding functions for RDF graphs. In one embedding (tr_R), graphs are embedded as RIF documents and variables (originating from blank nodes) are skolemized, i.e., replaced with new constant symbols. In the other embedding (tr_Q), graphs are embedded as condition formulas and variables (originating from blank nodes) are existentially quantified. The following sections show how these embeddings can be used for reasoning with combinations.

For skolemization we assume a function sk that takes as argument a formula φ and returns a formula φ' that is obtained from φ by replacing every variable symbol ?*x* with <`new-iri`>, where `new-iri` is a new globally unique IRI, i.e., it does not occur in the graph or its vicinity (i.e., entailing combination or entailed graph/formula).

RDF Construct	RIF Construct	Mapping
Triple `s p o .`	Frame formula tr(s)[tr(p) -> tr(o)]	tr(`s p o .`) = tr(s)[tr(p) -> tr(o)]
Graph *S*	Group formula $tr_R(S)$	$tr_R(S)$ = sk(`Document (Group (`tr(t_1) ... tr(t_m)`) )`), where t_1, ..., t_m are the triples in *S*
Graph *S*	Condition formula $tr_Q(S)$	$tr_Q(S)$ = `Exists` tr(x_1) ... tr(x_n) `(And(`tr(t_1) ... tr(t_m)`))`, where x_1, ..., x_n are the blank nodes occurring in *S* and t_1, ..., t_m are the triples in *S*

9.1.3 Embedding Simple Entailment

The semantics of the RDF Vocabulary does not need to be axiomatized for Simple entailment. Nonetheless, the connection between RIF class membership and subclass statements and the RDF type and subclass statements needs axiomatization. We define:

R^{Simple} =

```
Document( Group(
 Forall ?x ?y (?x[rdf:type -> ?y] :- ?x # ?y)
 Forall ?x ?y (?x # ?y :- ?x[rdf:type -> ?y])
 Forall ?x ?y (?x[rdfs:subClassOf -> ?y] :- ?x ## ?y]) ))
```

The following theorem shows how checking RIF-Simple-entailment of combinations can be reduced to checking entailment of RIF conditions by using the embeddings of RDF graphs defined above.

Theorem A list-free RIF-RDF combination C=<R, R^{Simple},{S_1,...,S_n}> RIF-Simple-entails a generalized RDF graph T if and only if merge({R, $tr_R(S_1)$, ..., $tr_R(S_n)$}) entails $tr_Q(T)$; C RIF-Simple-entails a condition formula φ if and only if merge({R, R^{Simple}, $tr_R(S_1)$, ..., $tr_R(S_n)$}) entails φ.

Proof. We prove both directions through contraposition. We first consider condition formulas (the second part of the theorem), after which we consider graphs (the first part of the theorem).
In the proof we abbreviate merge({R, R^{Simple}, $tr_R(S_1)$, ..., $tr_R(S_n)$}) with R'.

(=>) Assume R' does not entail φ. This means there is some semantic multi-structure $\hat{I}$ that is a model of R', but not of φ. Consider the pair ($\hat{I}$, I), where I is the interpretation defined as follows:

- IR is D_{ind},
- IP is the set of all k in D_{ind} such that there exist some a, b in D_{ind} and $I_{truth}(I_{frame}(a)(k,b))$=**t**,
- LV is the union of the value spaces of all considered datatypes,
- IEXT(k) is the set of all pairs (a, b), with a, b, and k in D_{ind}, such that $I_{truth}(I_{frame}(a)(k,b))$=**t**,
- IS(i) is I_C(<i>), for every absolute IRI `i` in V_U, and
- IL((s, d)) is I_C(tr("s"^^d)), for every typed literal (s, d) in V_{TL}.

Clearly, ($\hat{I}$, I) is a common-RIF-RDF-interpretation: conditions 1-6 in the definition are satisfied by construction of I and conditions 7 and 8 are satisfied by condition 4 and by the fact that $\hat{I}$ is a model of R^{Simple}.

Consider a graph S_i in {S_1,...,S_n}. Let x_1,..., x_m be the blank nodes in S_i and let u_1,..., u_m be the new IRIs that were obtained from the variables $?x_1$,..., $?x_m$ through the skolemization in $tr_R(S_i)$, i.e., u_i=sk($?x_i$). Now, let A be a mapping from blank nodes to elements in D_{ind} such that A(x_j)=$I_C(u_j)$ for every blank node x_j in S_i. From the fact that I is a model of $tr_R(S_i)$ and by construction of I it follows that [I+A] satisfies S_i (see Section 1.5 of [RDF-Semantics]), and so I satisfies S_i.

We have that $\hat{I}$ is a model of R, by assumption. So, ($\hat{I}$, I) satisfies C. Again, by assumption, I is not a model of φ. Therefore, C does not entail φ.

Assume now that R' does not entail $tr_Q(T)$, which means there is a semantic multi-structure $\hat{I}$ that is a model of R', but not of $tr_Q(T)$. The common-RIF-RDF-interpretation ($\hat{I}$, I) is obtained in the same way as above, and so it satisfies C.

We proceed by contradiction. Assume I satisfies T. This means there is some mapping A from the blank nodes x_1,...,x_m in T to objects in D_{ind} such that [I+A] satisfies T. Consider now the semantic multi-structure $\hat{I}$*, which is the same as $\hat{I}$, with the exception of the mapping I^*_V on the variables $?x_1$,...,$?x_m$, which is defined as follows: $\hat{I}^*_V(?x_j)$=A(x_j) for each blank node x_j in S. By construction of I and since [I+A] satisfies T we can conclude that I* is a model of `And(tr(t1)... tr(tm))`, and so I is a model of $tr_Q(T)$, violating the assumption that it is not. Therefore, ($\hat{I}$, I) does not satisfy T and C does not entail T.

(<=) Assume C does not Simple-entail φ. This means there is some common-RIF-RDF-interpretation ($\hat{I}$, I) that satisfies C such that I is not a model of φ.

Consider the semantic multi-structure $\hat{I}$', which is like $\hat{I}$, except for the mapping I'_C on the new IRIs that were introduced by the skolemization mapping sk(). The mapping of these new IRIs is defined as follows:
For each graph S_i in {S_1,...,S_n}, let x_1,..., x_m be the blank nodes in S_i and let u_1,..., u_m be the new IRIs that were obtained from the variables $?x_1$,..., $?x_m$ through the skolemization in $tr_R(S_i)$, i.e., u_j=sk($?x_j$). Now, since I satisfies S_i, there must be a mapping A from blank nodes to elements in D_{ind} such that [I+A] satisfies S_i. We define $I'_C(u_j)$=A(x_j) for every blank node x_j in S_i.

By assumption, $\hat{I}$' is a model of R (recall that $\hat{I}$' differs from $\hat{I}$ only on the new IRIs, which are not in R). Clearly, I' is also a model of R^{Simple}, by conditions 7, 8, and 4 in the definition of common-RIF-RDF-interpretation. From the fact that I satisfies S_i and by construction of I' it follows that I' is a model of $tr_R(S_i)$. So, I' is a model of R'. Since I is not a model of φ and φ does not contain any of the new IRIs, I' is not the model of φ. Therefore, R' does not entail φ.

Assume now that C does not entail T, which means there is a common-RIF-RDF-interpretation ($\hat{I}$, I) that satisfies C, but I does not satisfy T. We obtain I' from I in the same way as above, and so it satisfies R'. It can be shown analogous to the (=>) direction that if I' is a model of $tr_Q(T)$, then there is a blank node mapping A such that [I+A] satisfies T, and thus I satisfies T, violating the assumption that it does not. Therefore, I' is not a model of $tr_Q(T)$ and thus R' does not entail $tr_Q(T)$. ☐

Theorem A list-free RIF-RDF combination <R,{S_1,...,S_n}> is satisfiable iff there is a semantic multi-structure $\hat{I}$ that is a model of merge({R, R^{Simple}, $tr_R(S_1)$, ..., $tr_R(S_n)$}).

Proof. The theorem follows immediately from the previous theorem and the observation that a combination (respectively, RIF document) is satisfiable (respectively, has a model) if and only if it does not entail the condition formula `"a"="b"`. ☐

9.1.4 Embedding RDF Entailment

We axiomatize the semantics of the RDF Vocabulary using the following RIF rules.

To finitely embed RDF entailment, we need to consider a subset of the RDF axiomatic triples. Given a combination C, the *context* of C includes C and its vicinity (i.e., all graphs/formulas considered for entailment checking). The set of *RDF finite-axiomatic triples* is the smallest set such that:

- every RDF axiomatic triple not of the form `rdf:_i rdf:type rdf:Property` is an RDF finite-axiomatic triple, where *i* is a positive integer,
- one triple `rdf:_m rdf:type rdf:Property`, for some positive integer *m* such that `rdf:_m` does not occur in the context of C, is an RDF finite-axiomatic triple, and
- if `rdf:_j` occurs in the context of C, for some positive integer *j*, then `rdf:_j rdf:type rdf:Property` is an RDF finite-axiomatic triple.

We assume that none of unary predicate symbols `ex:wellxml` and `ex:illxml` and no datatypes beyond those found in [RIF-DTB] are used in the context of the given combination and `pred:is-literal-anyURI ... pred:is-literal-XMLLiteral` are the positive guard predicates defined in [RIF-DTB].

R^{RDF}	=	merge ((R^{Simple}) union
		
		((tr(`s p o .`)) for every RDF finite-axiomatic triple `s p o .`) union
		((`ex:illxml(tr("s"^^rdf:XMLLiteral))`) for every non-well-typed literal of the form (s, `rdf:XMLLiteral`) in V_{TL}) union
		((`ex:wellxml(tr("s"^^rdf:XMLLiteral))`) for every well-typed literal of the form (s, `rdf:XMLLiteral`) in V_{TL}) union
		(
		`Forall ?x (?x[rdf:type -> rdf:Property] :- Exists ?y ?z (?y[?x -> ?z])),`
		
		`Forall ?x (?x[rdf:type -> rdf:XMLLiteral] :- ex:wellxml(?x)),`
		
		`Forall ?x (rif:error :- And(?x[rdf:type -> rdf:XMLLiteral] ex:illxml(?x))),`
		
		`Forall ?x (rif:error :- And(ex:illxml(?x) Or(pred:is-literal-anyURI(?x) ... pred:is-literal-XMLLiteral(?x))))`
		)

Here, inconsistencies may occur if non-well-typed XML literals, axiomatized using the `ex:illxml` predicate, are in the class extension of `rdf:XMLLiteral`. If this situation occurs, `rif:error` is derived, which signifies an inconsistency in the combination.

Theorem An RIF-RDF-satisfiable list-free RIF-RDF combination C=<R,{S_1,...,S_n}> RIF-RDF-entails a generalized RDF graph T iff merge({R^{RDF}, R, $tr_R(S_1)$, ..., $tr_R(S_n)$}) entails $tr_Q(T)$. C RIF-RDF-entails a condition formula φ iff merge({R^{RDF}, R, $tr_R(S_1)$, ..., $tr_R(S_n)$}) entails φ.

Proof. In the proof we abbreviate merge({R^{RDF}, R, $tr_R(S_1)$, ..., $tr_R(S_n)$}) with R'.
The proof is obtained from the proof of correspondence for Simple entailment in the previous section with the following modifications: (*) in the (=>) direction we additionally need to ensure that I does not satisfy `rif:error`, extend I to ensure it satisfies the RDF axiomatic triples and show that I is an RDF-interpretation, and

(**) in the (<=) direction we need to slightly extend the definition of ***I'*** to account for `ex:wellxml` and `ex:illxml`, and show that ***I'*** is a model of R^{RDF}.

(*) We assume that, for every non-well-typed literal of the form (s, `rdf:XMLLiteral`) in V_{TL}, $\boldsymbol{I}_C$(tr("s"^^rdf:XMLLiteral)) is not in the value space of any of the considered datatypes and tr("s"^^rdf:XMLLiteral)[rdf:type -> rdf:XMLLiteral] is not satisfied in ***I***. Since C is RIF-RDF-satisfiable, one can verify that this does not compromise satisfaction of R'. Finally, we may assume, without loss of generality, that ***I*** does not satisfy `rif:error`. See also the proof of the following theorem.

For any positive integer *j* such that `rdf:_j` does not occur in the context of C, I and ***I*** are extended such that IS(`rdf:_j`)=$\boldsymbol{I}_C$(`rdf:_j`)=$\boldsymbol{I}_C$(`rdf:_m`) (see the definition of finite-axiomatic triples above for the definition of *m*). Clearly, this does not affect satisfaction of R' or non-satisfaction of φ and $tr_Q(T)$.

To show that I is an RDF-interpretation, we need to show that I satisfies the RDF axiomatic triples and the RDF semantic conditions.
Satisfaction of the axiomatic triples follows immediately from the inclusion of tr(t) in R^{RDF} for every RDF finite-axiomatic triple t, the fact that ***I*** is a model of R^{RDF}, and construction of I. Consider the three RDF semantic conditions:

1	x is in IP if and only if <x, I(rdf:Property)> is in IEXT(I(rdf:type))
2	If "xxx"^^rdf:XMLLiteral is in *V* and xxx is a well-typed XML literal string, then (a) IL("xxx"^^rdf:XMLLiteral) is the XML value of xxx; (b) IL("xxx"^^rdf:XMLLiteral) is in LV; (c) IEXT(I(rdf:type)) contains <IL("xxx"^^rdf:XMLLiteral), I(rdf:XMLLiteral)>
3	If "xxx"^^rdf:XMLLiteral is in *V* and xxx is an ill-typed XML literal string, then (a) IL("xxx"^^rdf:XMLLiteral) is not in LV; (b) IEXT(I(rdf:type)) does not contain <IL("xxx"^^rdf:XMLLiteral), I(rdf:XMLLiteral)>.

Satisfaction of condition 1 follows from satisfaction of the first rule in R^{RDF} in ***I*** and construction of I; specifically, the second bullet in the definition.
Consider a well-typed XML literal "xxx"^^rdf:XMLLiteral. By the definition of satisfaction in RIF BLD, $\boldsymbol{I}_C$("xxx"^^rdf:XMLLiteral) is the XML value of xxx (condition 2a), and is clearly in LV (condition 2b), by definition of I. Condition 2c is satisfied by satisfaction of the second rule in R^{RDF} in ***I***.
Satisfaction of 3a and 3b follows straightforwardly from our assumptions on ***I***. This establishes the fact that I is an RDF-interpretation.

(**) Recall that, by assumption, `ex:wellxml` and `ex:illxml` are not used in *R*. Therefore, changing satisfaction of atomic formulas involving `ex:wellxml` and `ex:illxml` does not affect satisfaction of *R*. We assume that $\boldsymbol{I'}_C$(ex:wellxml)=k and $\boldsymbol{I'}_C$(ex:illxml)=l are distinct unique elements, i.e., no other constants is mapped to k and l.
We define $\boldsymbol{I'}_F$(k) and $\boldsymbol{I'}_F$(l) as follows: For every typed literal of the form (s, rdf:XMLLiteral) such that $\boldsymbol{I'}_C$(tr(s^^rdf:XMLLiteral))=u, if (s, rdf:XMLLiteral) is well-typed, $\boldsymbol{I}_{truth}(\boldsymbol{I'}_F(k)(u))$=**t** and $\boldsymbol{I}_{truth}(\boldsymbol{I'}_F(l)(u))$=**f**, otherwise $\boldsymbol{I}_{truth}(\boldsymbol{I'}_F(k)(u))$=**f** and $\boldsymbol{I}_{truth}(\boldsymbol{I'}_F(l)(u))$=**t**; $\boldsymbol{I'}_{truth}(\boldsymbol{I'}_F(k)(v))=\boldsymbol{I}_{truth}(\boldsymbol{I'}_F(l)(v))$=**f** for every other object v in $\boldsymbol{D}_{ind}$.

Consider R^{RDF}. Satisfaction of R^{Simple} was established in the proof in the previous section. Satisfaction of the facts corresponding to the RDF axiomatic triples in ***I'*** follows immediately from the definition of common-RIF-RDF-interpretation and the fact that I is an RDF-interpretation, and thus satisfies all RDF axiomatic triples. Satisfaction of the `ex:wellxml` and `ex:illxml` facts in R^{RDF} follows immediately from the definition of ***I'***. Finally, satisfaction of the rules in R^{RDF} follow straightforwardly from the RDF semantic conditions 1, 2, and 3. This establishes the fact that ***I'*** is a model of R^{RDF}. ☐

Theorem A list-free RIF-RDF combination $<R,\{S_1,...,S_n\}>$ is RIF-RDF-satisfiable iff merge({R^{RDF}, *R*, $tr_R(S_1)$, ..., $tr_R(S_n)$}) does not entail `rif:error`.

Proof. Recall that we assume `rif:error` does not occur in *R*. If $<R,\{S_1,...,S_n\}>$ is not RIF-RDF-satisfiable, then either merge({*R*, $tr_R(S_1)$, ..., $tr_R(S_n)$}) is not consistent, or condition 3a or 3b (see previous proof) is violated. In either case, `rif:error` is entailed. If `rif:error` is entailed, either merge({R^{RDF}, *R*, $tr_R(S_1)$, ..., $tr_R(S_n)$}) is inconsistent, which means merge({*R*, $tr_R(S_1)$, ..., $tr_R(S_n)$}) is not consistent and thus $<R,\{S_1,...,S_n\}>$ is not RIF-RDF-satisfiable, or the body of the second or third rule in R^{RDF} is satisfied in every model, which means either condition 3a or 3b is violated, and so $<R,\{S_1,...,S_n\}>$ is not RIF-RDF-satisfiable. ☐

9.1.5 Embedding RDFS Entailment

We axiomatize the semantics of the RDF(S) Vocabulary using the following RIF rules.

Similar to the RDF case, the set of *RDFS finite-axiomatic triples* is the smallest set such that:

- every RDFS axiomatic triple not of the form `rdf:_i ... ...`, where *i* is a positive integer, is an RDFS finite-axiomatic triple,
- the triples `rdf:_m rdf:type rdfs:ContainerMembershipProperty`, `rdf:_m rdfs:domain rdfs:Resource`, and `rdf:_m rdfs:range rdfs:Resource`, for some positive integer *m* such that `rdf:_m` does not occur in the context of C, are RDFS finite-axiomatic triples, and
- if `rdf:_j` occurs in the context of the combination C, for some positive integer *j*, then `rdf:_j rdf:type rdfs:ContainerMembershipProperty`, `rdf:_j rdfs:domain rdfs:Resource`, and `rdf:_j rdfs:range rdfs:Resource` are RDFS finite-axiomatic triples.

We assume that the unary predicate symbol `ex:welllit` is not used in the context of the given combination.

R^{RDFS}	=	merge((R^{RDF}) union
		
		((tr(`s p o .`) for every RDFS finite-axiomatic triple `s p o .`) union
		((ex:welllit("s"^^u)) for every well-typed literal (s,u) in V_{TL}) union
		((sk(tr(s))[rdf:type -> rdfs:Resource) for every name or blank node s) union
		(
		`Forall ?x (?x[rdf:type -> rdfs:Resource] :- Exists ?y ?z (?x[?y -> ?z])),`
		
		`Forall ?x (?x[rdf:type -> rdfs:Resource] :- Exists ?y ?z (?z[?y -> ?x])),`
		
		`Forall ?u ?v ?x ?y (?u[rdf:type -> ?y] :- And(?x[rdfs:domain -> ?y] ?u[?x -> ?v])),`
		
		`Forall ?u ?v ?x ?y (?v[rdf:type -> ?y] :- And(?x[rdfs:range -> ?y] ?u[?x -> ?v])),`
		
		`Forall ?x (?x[rdfs:subPropertyOf -> ?x] :- ?x[rdf:type -> rdf:Property]),`
		
		`Forall ?x ?y ?z (?x[rdfs:subPropertyOf -> ?z] :- And (?x[rdfs:subPropertyOf -> ?y] ?y[rdfs:subPropertyOf -> ?z])),`
		
		`Forall ?x ?y ?z1 ?z2 (?z1[?y -> ?z2] :- And (?x[rdfs:subPropertyOf -> ?y] ?z1[?x -> ?z2])),`
		
		`Forall ?x (?x[rdfs:subClassOf -> rdfs:Resource] :- ?x[rdf:type -> rdfs:Class]),`
		
		`Forall ?x ?y ?z (?z[rdf:type -> ?y] :- And (?x[rdfs:subClassOf -> ?y] ?z[rdf:type -> ?x])),`
		
		`Forall ?x (?x[rdfs:subClassOf -> ?x] :- ?x[rdf:type -> rdfs:Class]),`
		
		`Forall ?x ?y ?z (?x[rdfs:subClassOf -> ?z] :- And (?x[rdfs:subClassOf -> ?y] ?y[rdfs:subClassOf -> ?z])),`
		
		`Forall ?x (?x[rdfs:subPropertyOf -> rdfs:member] :- ?x[rdf:type -> rdfs:ContainerMembershipProperty]),`
		
		`Forall ?x (?x[rdfs:subClassOf -> rdfs:Literal] :- ?x[rdf:type -> rdfs:Datatype]),`

```
Forall ?x (rif:error :- And(?x[rdf:type -> rdfs:Literal] ex:illxml(?x)))

Forall ?x (?x[rdf:type -> rdfs:Literal] :- ex:welllit(?x))

)
```

In the following theorems it is assumed that, in combinations C=<R,{S_1,...,S_n}>, R does not have mentions of `rdfs:Resource`, S_1,...,S_n do not have mentions of `rdfs:Resource` beyond triples of the form `xxx rdf:type rdfs:Resource`, and entailed graphs T and formulas φ do not have mentions of `rdfs:Resource`.

Theorem A RIF-RDFS-satisfiable list-free RIF-RDF combination C=<R,{S_1,...,S_n}> RIF-RDFS-entails a generalized RDF graph T if and only if merge({R, R^{RDFS}, $tr_R(S_1)$, ..., $tr_R(S_n)$}) entails $tr_Q(T)$; C RIF-RDFS-entails a condition formula φ if and only if merge({R, R^{RDFS}, $tr_R(S_1)$, ..., $tr_R(S_n)$}) entails φ.

Proof. In the proof we abbreviate merge({R, R^{RDFS}, $tr_R(S_1)$, ..., $tr_R(S_n)$}) with R'.
The proof is then obtained from the proof of correspondence for RDF entailment in the previous section with the following modifications: (*) in the (=>) direction we need to slightly amend the definition of I to account for `rdfs:Literal` and `rdfs:Resource`, and show that I is an RDFS-interpretation and (**) in the (<=) direction we need to show that ***I'*** is a model of R^{RDFS}.

(*) In addition to the earlier assumptions about ***I***, we assume that tr(`"s"^^rdf:XMLLiteral)[rdf:type -> rdfs:Literal]` is not satisfied in ***I***, for any typed literal of the form (`s`, `rdf:XMLLiteral`) in V_{TL}. We amend the definition of I by changing the definitions of LV and IEXT to the following:

- LV is (union of the value spaces of all considered datatypes) union (set of all k in $\boldsymbol{D}_{ind}$ such that $\boldsymbol{I}_{truth}(\boldsymbol{I}_{frame}(k)(\boldsymbol{I}_C($`rdf:type`$),\boldsymbol{I}_C($`rdfs:Literal`$)))$=**t**).

Clearly, this change does not affect satisfaction of the RDF axiomatic triples and the semantic conditions 1 and 2. To see that condition 3 is still satisfied, consider some non-well-typed XML literal `t`. By assumption, tr(`t`)`[rdf:type -> rdfs:Literal]` is not satisfied and thus IL(`t`) is not in ICEXT(`rdfs:Literal`). And, since IL(`t`) is not in the value space of any considered datatype, it is not in LV.

- For every k, a, and b ∈ $\boldsymbol{D}_{ind}$ such that k≠$\boldsymbol{I}_C$(`rdf:type`) or b≠$\boldsymbol{I}_C$(`rdfs:Resource`), (a, b) ∈ IEXT(k) iff $\boldsymbol{I}_{truth}(\boldsymbol{I}_{frame}(a)(k,b))$=**t**;
- for every a ∈ $\boldsymbol{D}_{ind}$, (a, $\boldsymbol{I}_C$(`rdfs:Resource`)) ∈ IEXT($\boldsymbol{I}_C$(`rdf:type`)).

Clearly, this change does not affect satisfaction of the RDF axiomatic triples and semantic conditions, nor does it affect satisfaction of the graphs S_1,...,S_n. It also does not affect satisfaction of the entailed graph or condition, since (by assumption) this does not contain a mention of `rdfs:Resource`. To show that I is an RDFS-interpretation, we need to show that I satisfies the RDFS axiomatic triples and the RDFS semantic conditions.
Satisfaction of the axiomatic triples follows immediately from the inclusion of tr(`t`) in R^{RDFS} for every RDFS finite-axiomatic triple `t`, the fact that ***I*** is a model of R^{RDFS}, construction of I, and the extension of I in the proof of the RDF entailment embedding. Consider the RDFS semantic conditions:

1	(a) x is in ICEXT(y) if and only if <x,y> is in IEXT(I(`rdf:type`)) (b) IC = ICEXT(I(`rdfs:Class`)) (c) IR = ICEXT(I(`rdfs:Resource`)) (d) LV = ICEXT(I(`rdfs:Literal`))
2	If <x,y> is in IEXT(I(`rdfs:domain`)) and <u,v> is in IEXT(x) then u is in ICEXT(y)
3	If <x,y> is in IEXT(I(`rdfs:range`)) and <u,v> is in IEXT(x) then v is in ICEXT(y)
4	IEXT(I(`rdfs:subPropertyOf`)) is transitive and reflexive on IP
5	If <x,y> is in IEXT(I(`rdfs:subPropertyOf`)) then x and y are in IP and IEXT(x) is a subset of IEXT(y)
6	If x is in IC then <x, I(`rdfs:Resource`)> is in IEXT(I(`rdfs:subClassOf`))
7	If <x,y> is in IEXT(I(`rdfs:subClassOf`)) then x and y are in IC and ICEXT(x) is a subset of ICEXT(y)
8	IEXT(I(`rdfs:subClassOf`)) is transitive and reflexive on IC
9	If x is in ICEXT(I(`rdfs:ContainerMembershipProperty`)) then: < x, I(`rdfs:member`)> is in IEXT(I(`rdfs:subPropertyOf`))
10	If x is in ICEXT(I(`rdfs:Datatype`)) then <x, I(`rdfs:Literal`)> is in IEXT(I(`rdfs:subClassOf`))

Conditions 1a and 1b are simply definitions of ICEXT and IC, respectively. By definition it is the case that every element k in $\boldsymbol{D}_{ind}$ is in ICEXT(I(`rdfs:Resource`)). Since IR=$\boldsymbol{D}_{ind}$, it follows that IR = ICEXT(I(`rdfs:Resource`)), establishing 1c. Clearly, every object in ICEXT(I(`rdfs:Literal`)) is in LV, by definition. Consider any value k in LV. By definition, either k is in the value space of some considered datatype or $\boldsymbol{I}_{truth}(\boldsymbol{I}_{frame}(k)(\boldsymbol{I}_C($`rdf:type`$),\boldsymbol{I}_C($`rdfs:Literal`$)))$=**t**. In the latter case, clearly k is in ICEXT(I(`rdfs:Literal`)). In the former case, k is in the value space of some datatype with some label D, and thus $\boldsymbol{I}_{truth}(\boldsymbol{I}_F(\boldsymbol{I}_C($`pred:is`$D))(k))$=**t**. By the last rule in R^{RDFS}, it must consequently be the case that $\boldsymbol{I}_{truth}(\boldsymbol{I}_{frame}(k)(\boldsymbol{I}_C($`rdf:type`$),\boldsymbol{I}_C($`rdfs:Literal`$)))$=**t**, and thus k is in ICEXT(I(`rdfs:Literal`)). This establishes satisfaction of condition 1d in I.

Satisfaction in I of conditions 2 through 10 follows immediately from satisfaction in ***I*** of the 2nd through the 12th rule in the definition of R^{RDFS}.

This establishes the fact that I is an RDFS-interpretation.

(**) Consider R^{RDFS}. Satisfaction of R^{RDF} was established in the proof in the previous section. Satisfaction of the facts corresponding to the RDFS axiomatic triples in ***I'*** follows immediately from the definition of common-RIF-RDF-interpretation and the fact that I is an RDFS-interpretation, and thus satisfies all RDFS axiomatic triples.

Satisfaction of the 1st through the 12th rule in R^{RDFS} follow straightforwardly from the RDFS semantic conditions 1 through 10. Satisfaction of the 13th rule follows from the fact that, given an ill-typed XML literal `t`, IL(`t`) is not in LV (by RDF semantic condition 3), ICEXT(`rdfs:Literal`)=LV, and the fact that the `ex:illxml` predicate is only true for ill-typed XML literals. Finally, satisfaction of the last rule in R^{RDFS} follows from the fact that ICEXT(`rdfs:Literal`)=LV, the definition of LV as a superset of the union of the value spaces of all datatypes, and the definition of the `pred:is`*D* predicates. This establishes the fact that ***I'*** is a model of R^{RDFS}. ☐

Theorem A list-free RIF-RDF combination <R,{S_1,...,S_n}> is RIF-RDFS-satisfiable if and only if merge({R, R^{RDFS}, $tr_R(S_1)$, ..., $tr_R(S_n)$}) does not entail `rif:error`.

Proof. The theorem follows immediately from the previous theorem and the observations in the proof of the second theorem in the previous section. ☐

9.2 Embedding RIF-OWL 2 RL Combinations

It is known that expressive Description Logic languages such as OWL 2 DL cannot be straightforwardly embedded into typical rules languages such as RIF BLD [RIF-BLD], because of features such as disjunction and negation.

In this section we consider a subset of OWL 2 DL in RIF-OWL DL combinations, namely, the OWL 2 RL profile [OWL2-Profiles], and show how reasoning with RIF-OWL 2 RL combinations can be reduced to reasoning with RIF.

The embedding of RIF-OWL 2 RL combinations is not defined for combinations that include infinite OWL ontologies.

Since OWL 2 RL includes equality through `ObjectMaxCardinality` and `DataMaxCardinality` restrictions, as well as `FunctionalObjectProperty` `UniverseFunctionalObjectProperty`, `SameIndividual`, and `HasKey` axioms, and there is non-trivial interaction between such equality and the predicates in the RIF rules in the combination, embedding RIF-OWL 2 RL combinations into RIF requires equality. Therefore, the embedding presented in this appendix is not in RIF Core, even if the RIF document in the combination is. If the ontologies in the combination do not contain any of the mentioned constructs, the embedding is in Core. Also, it is well-known that adding equality to a rules language does not increase its expressiveness in the absence of function symbols: one can replace equality = with a new binary predicate symbol, and add rules for reflexivity and the principle of substitutivity (also called the replacement property).

9.2.1 Embedding RIF DL-document formulas into RIF BLD

Recall that the semantics of frame formulas in DL-document formulas is different from the semantics of frame formulas in RIF documents. Nonetheless, DL-document formulas can be embedded into RIF documents, by translating frame formulas to predicate formulas. The mapping tr is the identity mapping on all RIF formulas, with the exception of frame formulas, as defined in the following table.

In the table, the mapping tr' is an injective function that maps constants to new constants, i.e., constants that are not used in the original document or its vicinity (i.e., imported, entailed or entailing formula). It "generates" a new constant from an existing one.

Mapping RIF DL-document formulas to RIF documents.

RIF Construct	Mapping
Term t	tr(t)=t
Atomic formula φ that is not a frame formula	tr(φ)=φ
a[b_1->c_1 ... b_n->c_n], with n≥2	tr(a[b_1->c_1 ... b_n->c_n])=And(tr(a[b_1->c_1]) ... tr(a[b_n->c_n]))
a[b -> c], where a and c are terms and b ≠ rdf:type is a constant	tr(a[b -> c])=tr'(b)(a c)
a[rdf:type -> c], where a is a term and c is a constant	tr(a[rdf:type -> c])=tr'(c)(a)
a#c, where a is a term and c is a constant	tr(a#c)=tr'(c)(a)
b##c, where a,b are constants	tr(b##c) = Forall ?x (tr'(c)(?x) :- tr'(b)(?x))
Exists ?V1 ... ?Vn(φ)	tr(Exists ?V1 ... ?Vn(φ))=Exists ?V1 ... ?Vn(tr(φ))
And($φ_1$ ... $φ_n$)	tr(And($φ_1$ ... $φ_n$))=And(tr($φ_1$) ... tr($φ_n$))
Or($φ_1$ ... $φ_n$)	tr(Or($φ_1$ ... $φ_n$))=Or(tr($φ_1$) ... tr($φ_n$))
$φ_1$:- $φ_2$	tr($φ_1$:- $φ_2$)=tr($φ_1$) :- tr($φ_2$)
Forall ?V1 ... ?Vn(φ)	tr(Forall ?V1 ... ?Vn(φ))=Forall ?V1 ... ?Vn(tr(φ))
Group($φ_1$ ... $φ_n$)	tr(Group($φ_1$ ... $φ_n$))=Group(tr($φ_1$) ... tr($φ_n$))
Document(*$directive_1$* ... *$directive_n$* Γ)	tr(Document(*$directive_1$* ... *$directive_n$* Γ))=Document(*$directive_1$* ... *$directive_n$* tr(Γ))

For the purpose of making statements about this embedding, we define a notion of entailment for DL-document formulas.

Definition. A RIF-BLD DL-document formula *R* ***dl-entails*** a DL-condition φ if for every dl-semantic multi-structure ***Î*** that is a model of *R* it holds that $TVal_Î$(φ)=**t**. ☐

The following lemma establishes faithfulness with respect to entailment of the embedding.

RIF-BLD DL-document formula Lemma A RIF-BLD DL-document formula *R* dl-entails a DL-condition φ if and only if tr(*R*) entails tr(φ).

Proof. We prove both directions by contraposition.

(=>) Assume tr(*R*) does not entail tr(φ). This means there is some semantic multi-structure ***Î*** that is a model of tr(R'), but ***I*** = <***TV***, ***DTS***, ***D***, $\boldsymbol{D}_{ind}$, $\boldsymbol{D}_{func}$, $\boldsymbol{I}_C$, $\boldsymbol{I}_V$, $\boldsymbol{I}_F$, $\boldsymbol{I}_{frame}$, $\boldsymbol{I}_{NF}$, $\boldsymbol{I}_{sub}$, $\boldsymbol{I}_{isa}$, $\boldsymbol{I}_=$, $\boldsymbol{I}_{external}$, $\boldsymbol{I}_{truth}$> is not a model of tr(φ).
Consider the dl-semantic multi-structure ***Î****, which is obtained from ***Î*** as follows: ***I**** = <***TV***, ***DTS***, ***D***, $\boldsymbol{D}_{ind}$, $\boldsymbol{D}_{func}$, $\boldsymbol{I^*}_C$, $\boldsymbol{I^*}_{C'}$, $\boldsymbol{I}_V$, $\boldsymbol{I}_F$, $\boldsymbol{I^*}_{frame}$, $\boldsymbol{I}_{NF}$, $\boldsymbol{I^*}_{sub}$, $\boldsymbol{I^*}_{isa}$, $\boldsymbol{I}_=$, $\boldsymbol{I}_{external}$, $\boldsymbol{I}_{truth}$>, with $\boldsymbol{I^*}_{C'}$, $\boldsymbol{I^*}_{frame}$, $\boldsymbol{I^*}_{sub}$, and $\boldsymbol{I^*}_{isa}$ defined as follows: Let t be an element in ***D*** such that $\boldsymbol{I}_{truth}$(t)=**t** and let f in ***D*** be such that $\boldsymbol{I}_{truth}$(f)=**f**.

- for every constant c, with c'=tr'(c), $\boldsymbol{I^*}_{C'}$(c)=$\boldsymbol{I}_C$(c');
- for every constant c' used as unary predicate symbol in tr(*R*) or tr(φ) such that c'=tr'(c) for some constant c, and every object k in $\boldsymbol{D}_{ind}$, $\boldsymbol{I}_{truth}$($\boldsymbol{I}_F$($\boldsymbol{I}_C$(c'))(k))=**t** iff $\boldsymbol{I^*}_{frame}$(k)(($\boldsymbol{I}_{C'}$(rdf:type), $\boldsymbol{I}_{C'}$(c))=t;
- for every constant b' used as binary predicate symbol in tr(*R*) or tr(φ) such that b'=tr'(b) for some constant b, and every pair (k, l) in $\boldsymbol{D}_{ind}$ × $\boldsymbol{D}_{ind}$, $\boldsymbol{I}_{truth}$($\boldsymbol{I}_F$($\boldsymbol{I}_C$(b'))(k,l))=**t** iff $\boldsymbol{I^*}_{frame}$(k)(($\boldsymbol{I}_{C'}$(b),l))=t,
- if $\boldsymbol{I^*}_{frame}$(k)((b_1,...,b_n))=t and $\boldsymbol{I^*}_{frame}$(k)((c_1,...,c_m))=t for any two finite bags (b_1,...,b_n) and (c_1,...,c_m), then $\boldsymbol{I^*}_{frame}$(k)((b_1,...,b_n,c_1,...,c_m))=t;
- $\boldsymbol{I^*}_{frame}$(b)=f for any other bag b;
- for any two k, l ∈ ***D***, $\boldsymbol{I^*}_{sub}$(k,l)=t if for every u ∈ ***D***, $\boldsymbol{I}_{truth}$($\boldsymbol{I}_F$(k)(u))=**t** implies $\boldsymbol{I}_{truth}$($\boldsymbol{I}_F$(l)(u))=**t**, otherwise $\boldsymbol{I^*}_{sub}$(k,l)=f;
- for any two k, l ∈ ***D***, $\boldsymbol{I^*}_{isa}$(k,l)=t if $\boldsymbol{I}_{truth}$($\boldsymbol{I}_F$(k)(u))=**t**, otherwise $\boldsymbol{I^*}_{isa}$(k,l)=f.

Observe that tr(*R*) and tr(φ) do not include frame formulas.
To show that ***Î**** is a model of *R* and ***I**** is not a model of φ, we only need to show that (+) for any frame formula a[b -> c] that is a DL-condition, ***I**** is a model of a[b -> c] iff ***I*** is a model of tr(a[b -> c]). This argument straightforwardly extends to the case of frames with multiple b_is and c_is, since in RIF semantic structures the following condition is required to hold: $TVal_I$(a[b_1->c_1 ... b_n->c_n]) = **t** if and only if $TVal_I$(a[b_1->c_1]) = ... = $TVal_I$(a[b_n->c_n]) = **t** [RIF-BLD]. The argument for formulas a # b and a ## b is analogous.

Consider the case b=rdf:type. Then,
I* is a model of a[b -> c] iff $\boldsymbol{I}_{truth}$($\boldsymbol{I^*}_{frame}$(***I***(a))($\boldsymbol{I}_{C'}$(rdf:type),$\boldsymbol{I}_{C'}$(c)))=**t**.
From the definition of ***I**** we obtain
$\boldsymbol{I}_{truth}$($\boldsymbol{I^*}_{frame}$(***I***(a))($\boldsymbol{I}_{C'}$(rdf:type),$\boldsymbol{I}_{C'}$(c)))=**t** iff $\boldsymbol{I^*}_{frame}$(***I***(a))($\boldsymbol{I}_{C'}$(rdf:type),$\boldsymbol{I}_{C'}$(c))=t.
By definition of the embedding, we know that tr'(c) is used as unary predicate symbol in tr(*R*) or tr(φ). From the definition of ***I**** we obtain
$\boldsymbol{I^*}_{frame}$(***I***(a))($\boldsymbol{I}_{C'}$(rdf:type),$\boldsymbol{I}_{C'}$(c))=t iff $\boldsymbol{I}_{truth}$($\boldsymbol{I}_F$($\boldsymbol{I}_C$(tr'(c)))(***I***(a)))=**t**.
Finally, since tr(a[b -> c])=tr'(c)(a), we obtain
$\boldsymbol{I}_{truth}$($\boldsymbol{I}_F$($\boldsymbol{I}_C$(tr'(c)))(***I***(a)))=**t** iff ***I*** is a model of tr(a[b -> c]).
From this chain of equivalences follows that ***I**** is a model of a[b -> c] iff ***I*** is a model of tr(a[b -> c]).

The argument for the case b≠rdf:type is analogous, thereby obtaining (+).

(<=) Assume *R* does not dl-entail φ. This means there is some dl-semantic multi-structure ***Î*** that is a model of *R*, but ***I*** = <***TV***, ***DTS***, ***D***, $\boldsymbol{D}_{ind}$, $\boldsymbol{D}_{func}$, $\boldsymbol{I}_C$, $\boldsymbol{I}_{C'}$, $\boldsymbol{I}_V$, $\boldsymbol{I}_F$, $\boldsymbol{I}_{frame}$, $\boldsymbol{I}_{NF}$, $\boldsymbol{I}_{sub}$, $\boldsymbol{I}_{isa}$, $\boldsymbol{I}_=$, $\boldsymbol{I}_{external}$, $\boldsymbol{I}_{truth}$>, is not a model of φ. Let *B* be the set of constant symbols occurring in the frame formulas of the forms a[rdf:type -> b] and a[b -> c] in *R* or φ.
Consider the semantic multi-structure ***Î****, which is obtained from ***Î*** as follows: ***I**** = <***TV***, ***DTS***, ***D***, $\boldsymbol{D}_{ind}$, $\boldsymbol{D}_{func}$, $\boldsymbol{I^*}_C$, $\boldsymbol{I}_V$, $\boldsymbol{I^*}_F$, $\boldsymbol{I^*}_{frame}$, $\boldsymbol{I}_{NF}$, $\boldsymbol{I}_{sub}$, $\boldsymbol{I}_{isa}$, $\boldsymbol{I}_=$, $\boldsymbol{I}_{external}$, $\boldsymbol{I}_{truth}$>. Let t and f in ***D*** be such that $\boldsymbol{I}_{truth}$(t)=**t** and $\boldsymbol{I}_{truth}$(f)=**f**. We define $\boldsymbol{I^*}_C$, $\boldsymbol{I^*}_{frame}$, and $\boldsymbol{I^*}_F$ as follows:

- $\boldsymbol{I^*}_C$(tr'(c))=$\boldsymbol{I}_{C'}$(c) and $\boldsymbol{I^*}_C$(c)=$\boldsymbol{I}_C$(c), for any constant c not in the range of tr';
- $\boldsymbol{I^*}_{frame}$(b)=f for any finite bag b of ***D***, and
- $\boldsymbol{I^*}_F$ is defined as follows:
    - for every constant c, given an object k in $\boldsymbol{D}_{ind}$, if $\boldsymbol{I}_{truth}$($\boldsymbol{I}_{frame}$(k)(($\boldsymbol{I}_{C'}$(rdf:type), $\boldsymbol{I}_{C'}$(c)))=**t**, $\boldsymbol{I^*}_F$($\boldsymbol{I^*}_C$(tr'(c)))(k)=t; $\boldsymbol{I^*}_F$($\boldsymbol{I^*}_C$(tr'(c)))(k')=f for any other k' in $\boldsymbol{D}_{ind}$,
    - for every constant c, given a pair (k, l) in $\boldsymbol{D}_{ind}$ × $\boldsymbol{D}_{ind}$, if $\boldsymbol{I}_{truth}$($\boldsymbol{I}_{frame}$(k)(($\boldsymbol{I}_{C'}$(c),l)))=**t**, $\boldsymbol{I^*}_F$(tr'(c))(k,l)=t;$\boldsymbol{I^*}_F$(tr'(c))(k',l')=f for any other pair (k', l') in $\boldsymbol{D}_{ind}$ × $\boldsymbol{D}_{ind}$, and
    - $\boldsymbol{I^*}_F$(c)=$\boldsymbol{I}_F$(c) for every other constant c.

Observe that *R* and φ do not include predicate formulas involving derived constant symbols tr'(c). The remainder of the proof is analogous to the (=>) direction. ☐

9.2.2 Embedding OWL 2 RL into RIF BLD

The embedding of OWL 2 RL into RIF BLD has two stages: normalization and embedding.

The OWL 2 syntax is given in terms of a Structural Specification, and there is a functional-style syntax that is a serialization of this Structural Specification. For convenience, normalization and embedding in this section are done in terms of the functional-style syntax. That is, the normalization mapping takes as input a functional-style syntax ontology document and produces a normalized ontology document. The embedding mapping takes as input a normalized ontology document and produces an RIF document. We refer to Section 4.2 of [OWL2-Profiles] for the specification of the OWL 2 RL syntax.

9.2.2.1 Normalization of OWL 2 RL

Normalization splits the OWL axioms so that the later mapping to RIF of the individual axioms results in rules. Additionally, it simplifies the axioms and removes annotations.

It is assumed that the normalization process is preceded by a simplification process that removes all namespace prefixes, turns all CURIEs and relative IRIs into absolute IRIs, and removes all annotations, import statements, entity declarations, and annotation axioms.

We note here that, strictly speaking, simplified OWL 2 RL ontologies are not OWL 2 RL ontologies in the general case, because certain entity declarations are required (e.g., those distinguishing data from object properties). It is assumed that such entity declarations are present implicitly, i.e., they do not appear explicitly in the simplified ontology, but they are known. We also note that removing import statements in the simplification does not prohibit importing ontologies in practice; since combinations contain sets of ontologies, all imported ontologies may be added to these sets. The normalization mapping tr_N takes as input a simplified ontology O and produces an equivalent normalized ontology O'.

The names of variables used in the mapping generally correspond to the names of productions in the OWL 2 RL grammar.

Normalization of OWL 2 RL ontologies.

#	Statement	Normalized Statement	Condition on translation
1	tr_N(Ontology([ontologyIRI [versionIRI]] axiom$_1$ ... axiom$_n$))	Ontology(tr_N(axiom$_1$) ... tr_N(axiom$_n$))	
2	tr_N(SubClassOf(subClassExpression ...ObjectIntersectionOf(superClassExpression$_1$ ... superClassExpression$_n$)...))	tr_N(SubClassOf(subClassExpression ...superClassExpression$_1$...)) ... tr_N(SubClassOf(subClassExpression ...superClassExpression$_n$...))	
3	tr_N(SubClassOf(subClassExpression$_1$ ObjectComplementOf(subClassExpression$_1$)))	tr_N(SubClassOf(ObjectIntersectionof(subClassExpression$_1$ subClassExpression$_2$) owl:Nothing))	
4	tr_N(SubClassOf(subClassExpression *X*))	SubClassOf(subClassExpression *X*)	*X* is a superClassExpression that does not contain ObjectIntersectionOf or ObjectComplementOf
5	tr_N(EquivalentClasses(equivClassExpression$_1$ ... equivClassExpression$_m$))	tr_N(SubClassOf(equivClassExpression$_1$ equivClassExpression$_2$)) ... tr_N(SubClassOf(equivClassExpression$_{m-1}$ equivClassExpression$_m$)) tr_N(SubClassOf(equivClassExpression$_m$ equivClassExpression$_1$))	
6	tr_N(DisjointClasses(subClassExpression$_1$ ... subClassExpression$_m$))	tr_N(SubClassOf(ObjectIntersectionOf(subClassExpression$_1$ subClassExpression$_2$) owl:Nothing)) ... tr_N(SubClassOf(ObjectIntersectionOf(subClassExpression$_1$ subClassExpression$_m$) owl:Nothing)) ... tr_N(SubClassOf(ObjectIntersectionOf(subClassExpression$_{m-1}$ subClassExpression$_m$) owl:Nothing))	
7	tr_N(SubObjectPropertyOf(subPropertyExpression superPropertyExpression))	SubObjectPropertyOf(subPropertyExpression superPropertyExpression)	
8	tr_N(SubDataPropertyOf(subPropertyExpression superPropertyExpression))	SubDataPropertyOf(subPropertyExpression superPropertyExpression)	

9	tr$_N$(EquivalentObjectProperties(ObjectPropertyExpression$_1$ ... ObjectPropertyExpression$_m$))	tr$_N$(SubObjectPropertyOf(ObjectPropertyExpression$_1$ ObjectPropertyExpression$_2$)) ... tr$_N$(SubObjectPropertyOf(ObjectPropertyExpression$_{m-1}$ ObjectPropertyExpression$_m$)) tr$_N$(SubObjectPropertyOf(ObjectPropertyExpression$_m$ ObjectPropertyExpression$_1$))	
10	tr$_N$(EquivalentDataProperties(DataPropertyExpression$_1$ ... DataPropertyExpression$_m$))	tr$_N$(SubDataPropertyOf(PropertyExpression$_1$ DataPropertyExpression$_2$)) ... tr$_N$(SubDataPropertyOf(PropertyExpression$_{m-1}$ DataPropertyExpression$_m$)) tr$_N$(SubDataPropertyOf(PropertyExpression$_m$ DataPropertyExpression$_1$))	
11	tr$_N$(DisjointObjectProperties(ObjectPropertyExpression$_1$ ... ObjectPropertyExpression$_m$))	DisjointObjectProperties(ObjectPropertyExpression$_1$ ObjectPropertyExpression$_2$) ... DisjointObjectProperties(ObjectPropertyExpression$_1$ ObjectPropertyExpression$_m$) ... DisjointObjectProperties(ObjectPropertyExpression$_{m-1}$ ObjectPropertyExpression$_m$)	
12	tr$_N$(DisjointDataProperties(DataPropertyExpression$_1$ ... DataPropertyExpression$_m$))	DisjointDataProperties(DataPropertyExpression$_1$ DataPropertyExpression$_2$) ... DisjointDataProperties(DataPropertyExpression$_1$ DataPropertyExpression$_m$) ... DisjointDataProperties(DataPropertyExpression$_{m-1}$ DataPropertyExpression$_m$)	
13	tr$_N$(InverseObjectProperties(PropertyExpression$_1$ PropertyExpression$_2$))	InverseObjectProperties(PropertyExpression$_1$ PropertyExpression$_2$)	
14	tr$_N$(ObjectPropertyDomain(PropertyExpression superClassExpression))	tr$_N$(SubClassOf(ObjectSomeValuesFrom(PropertyExpression owl:Thing) superClassExpression)	
15	tr$_N$(DataPropertyDomain(DataProperty superClassExpression))	tr$_N$(SubClassOf(ObjectSomeValuesFrom(DataProperty owl:Thing) superClassExpression)	
16	tr$_N$(ObjectPropertyRange(ObjectInverseOf(Property) superClassExpression))	tr$_N$(SubClassOf(ObjectSomeValuesFrom(Property owl:Thing) superClassExpression)	
17	tr$_N$(ObjectPropertyRange(Property superClassExpression))	tr$_N$(SubClassOf(ObjectSomeValuesFrom(ObjectInverseOf(Property) owl:Thing) superClassExpression)	Property is not an inverse property expression
18	tr$_N$(DataPropertyRange(DataProperty superClassExpression))	tr$_N$(SubClassOf(owl:Thing DataAllValuesFrom(DataProperty superClassExpression))	
19	tr$_N$(FunctionalObjectProperty(PropertyExpression))	FunctionalObjectProperty(PropertyExpression)	
20	tr$_N$(FunctionalDataProperty(DataProperty))	FunctionalDataProperty(DataProperty)	

21	tr_N(InverseFunctionalObjectProperty(PropertyExpression))	InverseFunctionalObjectProperty(PropertyExpression)
22	tr_N(IrreflexiveObjectProperty(PropertyExpression))	IrreflexiveObjectProperty(PropertyExpression)
23	tr_N(SymmetricObjectProperty(PropertyExpression))	SymmetricObjectProperty(PropertyExpression)
24	tr_N(AsymmetricObjectProperty(PropertyExpression))	AsymmetricObjectProperty(PropertyExpression)
25	tr_N(TransitiveObjectProperty(PropertyExpression))	TransitiveObjectProperty(PropertyExpression)
26	tr_N(DatatypeDefinition(...))	DatatypeDefinition(...)
27	tr_N(HasKey(...))	HasKey(...)
28	tr_N(SameIndividual(Individual$_1$ ... Individual$_m$))	SameIndividual(Individual$_1$ Individual$_2$) ... SameIndividual(Individual$_{m-1}$ Individual$_m$) SameIndividual(Individual$_m$ Individual$_1$)
29	tr_N(DifferentIndividuals(Individual$_1$ ... Individual$_m$))	DifferentIndividuals(Individual$_1$ Individual$_2$) ... SameIndividual(Individual$_1$ Individual$_m$) ... SameIndividual(Individual$_{m-1}$ Individual$_m$)
30	tr_N(ClassAssertion(superClassExpression Individual))	SubClassOf(ObjectOneOf(Individual) superClassExpression)
31	tr_N(ObjectPropertyAssertion(ObjectPropertyExpression source target))	SubClassOf(ObjectOneOf(source) ObjectHasValue(ObjectPropertyExpression target))
32	tr_N(NegativeObjectPropertyAssertion(ObjectPropertyExpression source target))	SubClassOf(ObjectOneOf(source) ObjectComplementOf(ObjectHasValue(ObjectPropertyExpression target)))
33	tr_N(DataPropertyAssertion(DataProperty source target))	SubClassOf(ObjectOneOf(source) DataHasValue(DataProperty target))
34	tr_N(	SubClassOf(ObjectOneOf(source) ObjectComplementOf(DataHasValue(DataProperty target)))

	NegativeDataPropertyAssertion(DataProperty source target))		

We note that normalized OWL 2 RL ontologies are not necessarily OWL 2 RL ontologies, since `owl:Thing` may appear in subclass expressions, as a result of the transformation of `DataPropertyRange` axioms.

The following lemma establishes the fact that, for the purpose of entailment, the ontologies in a combination may be replaced with their normalization.

Normalization Lemma Given a combination $C=\langle R,\{O_1,\ldots,O_n\}\rangle$, where $O_1,\ldots,O_n$ are simplified OWL 2 RL ontologies that do not import ontologies, C RIF-OWL Direct-entails φ iff $C'=\langle R,\{tr_N(O_1),\ldots,tr_N(O_n)\}\rangle$ RIF-OWL Direct-entails φ.

> **Proof.** We prove both directions by contradiction: if the entailment does not hold on the one side, we show that it also does not hold on the other.
>
> (=>) Assume C' does not RIF-OWL Direct-entail φ. This means there is a common-RIF-OWL Direct-interpretation (***Î***, I) that is a model of C', but ***I*** is not a model of φ. By the definition of satisfaction of axioms and assertions in Section 2.3 and the interpretation of object property, data range, and class expressions in Section 2.2 in [OWL2-Semantics] it is easy to verify that, if for every axiom d appearing in $\{O_1,\ldots,O_n\}$, I satisfies tr_N(d), then I satisfies O_1,..., and O_n, and thus (***I***, I) satisfies C. Since ***I*** is not a model of φ, C does not RIF-OWL Direct-entail φ.
>
> (<=) Assume C does not RIF-OWL Direct-entail φ. This means there is a common-RIF-OWL Direct-interpretation (***Î***, I) that is a model of C, but ***I*** is not a model of φ. It is easy to verify, by the definition of satisfaction of axioms and assertions in Section 2.3 and the interpretation of object property, data range, and class expressions in Section 2.2 in [OWL2-Semantics], that I satisfies $tr_N(O_1)$,..., and $tr_N(O_n)$. So, (***Î***, I) is a model of C', and thus C' does not RIF-OWL Direct-entail φ. ☐

9.2.2.2 Embedding Normalized OWL 2 RL

We now proceed with the embedding of normalized OWL 2 RL ontologies into RIF DL-document formulas. The embedding function tr_O takes as input a normalized OWL 2 RL ontology and returns a RIF-BLD DL-document formula. The embeddings of IRIs and literals is as defined in Section 9.1.1 Embedding Symbols. It is assumed that the Vocabulary *V* of the ontologies includes all the constants used in the RIF documents and condition formulas under consideration.

Embedding Normalized OWL 2 RL

#	Normalized OWL	RIF-BLD DL-document formula	Condition on translation

1	tr_O(
	
	Ontology(
	 axiom$_1$
	  ...
	 axiom$_n$
	)
	
	)
		Document(Group(
		
		tr_O(axiom$_1$)
		  ...
		tr_O(axiom$_n$)
		
		))	

2	tr_O(SubClassOf(subClassExpression superClassExpression))	tr_O(subClassExpression,superClassExpression)	
3	tr_O(subClassExpression, [Object|Data]AllValuesFrom(property$_1$ ...[Object|Data]AllValuesFrom(property$_n$ *X*) ...))	Forall ?x ?y$_1$ ... ?y$_n$ (tr_O(*X*, ?y$_n$) :- And(tr_O(subClassExpression, ?x) tr_O(property$_1$, ?x, ?y$_1$) tr_O(property$_2$, ?y$_1$, ?y$_2$) ... tr_O(property$_n$, ?y$_{n-1}$, ?y$_n$) tr_O(*X*, ?y$_n$)))	n≥1 and *X* is not an [Object|Data]AllValuesFrom or [Object|Data]MaxCardinality expression.
3a	tr_O(subClassExpression, *X*)	Forall ?x (tr_O(*X*, ?y$_n$) :- And(tr_O(subClassExpression$_1$, ?x) tr_O(*X*, ?x)))	*X* is not an [Object|Data]AllValuesFrom or [Object|Data]MaxCardinality expression.
4	tr_O(subClassExpression, [Object|Data]AllValuesFrom(property$_1$ ...[Object|Data]AllValuesFrom(property$_n$ [Object|Data]MaxCardinality(0 PropertyExpression ClassExpression) ...))	Forall ?x ?y$_1$ ... ?y$_n$?z (rif:error:- And(tr_O(subClassExpression, ?x) tr_O(property$_1$, ?x, ?y$_1$) tr_O(property$_2$, ?y$_1$, ?y$_2$) ... tr_O(property$_n$, ?y$_{n-1}$, ?y$_n$) tr_O(PropertyExpression, ?y$_n$, ?z) tr_O(ClassExpression, ?z)))	n≥1.
4a	tr_O(subClassExpression, [Object|Data]MaxCardinality(0 PropertyExpression ClassExpression))	Forall ?x ?y (rif:error :- And(tr_O(subClassExpression, ?x) tr_O(PropertyExpression, ?x, ?y) tr_O(ClassExpression, ?y)))	
5	tr_O(subClassExpression, [Object|Data]AllValuesFrom(property$_1$ ...[Object|Data]AllValuesFrom(property$_n$ [Object|Data]MaxCardinality(1 PropertyExpression ClassExpression) ...))	Forall ?x ?y$_1$ ... ?y$_n$?z$_1$?z$_2$ (?z$_1$=?z$_2$:- And(tr_O(subClassExpression, ?x) tr_O(property$_1$, ?x, ?y$_1$) tr_O(property$_2$, ?y$_1$, ?y$_2$) ... tr_O(property$_n$, ?y$_{n-1}$, ?y$_n$) tr_O(PropertyExpression, ?y$_n$, ?z$_1$) tr_O(PropertyExpression, ?y$_n$, ?z$_2$) tr_O(ClassExpression, ?z$_1$) tr_O(ClassExpression, ?z$_2$)))	n≥1.

W3C Editor's Draft

<table>
<tr><td>5a</td><td>tr$_O$(subClassExpression, [Object|Data]MaxCardinality(1 PropertyExpression ClassExpression))</td><td>Forall ?x ?y$_1$?y$_2$ (?y$_1$=?y$_2$:- And(
tr$_O$(subClassExpression, ?x)
tr$_O$(PropertyExpression, ?x, ?y$_1$)
tr$_O$(PropertyExpression, ?x, ?y$_2$)
tr$_O$(ClassExpression, ?y$_1$)
tr$_O$(ClassExpression, ?y$_2$)))</td><td></td></tr>
<tr><td>6</td><td>tr$_O$(A,?x)</td><td>?x[rdf:type -> tr(A)]</td><td>A is a Class or Datatype; x is a variable name</td></tr>
<tr><td>7</td><td>tr$_O$([Object|Data]IntersectionOf(ClassExpression$_1$ ... ClassExpression$_n$), ?x)</td><td>And(tr$_O$(ClassExpression$_1$, ?x) ...
tr$_O$(ClassExpression$_n$, ?x))</td><td>x is a variable name</td></tr>
<tr><td>8</td><td>tr$_O$(ObjectUnionOf(ClassExpression$_1$ ... ClassExpression$_n$), ?x)</td><td>Or(tr$_O$(ClassExpression$_1$, ?x) ...
tr$_O$(ClassExpression$_n$, ?x))</td><td>x is a variable name</td></tr>
<tr><td>9</td><td>tr$_O$([Object|Data]OneOf(Individual$_1$ ... Individual$_n$), ?x)</td><td>Or(?x = tr(Individual$_1$) ... ?x = tr(Individual$_n$))</td><td>x is a variable name</td></tr>
<tr><td>10</td><td>tr$_O$([Object|Data]SomeValuesFrom(PropertyExpression ClassExpression)), ?x)</td><td>Exists ?y(And(tr$_O$(PropertyExpression, ?x, ?y)] tr$_O$(ClassExpression, ?y)))</td><td>x is a variable name</td></tr>
<tr><td>11</td><td>tr$_O$(X, ?x, ?y)</td><td>?x[tr(X) -> ?y]</td><td>X is a Property; x, y are variable names</td></tr>
<tr><td>12</td><td>tr$_O$(ObjectInverseOf(X), ?x, ?y)</td><td>?y[tr(X) -> ?x]</td><td>X is a Property; x, y are variable names</td></tr>
<tr><td>13</td><td>tr$_O$([Object|Data]HasValue(PropertyExpression value), ?x)</td><td>tr$_O$(PropertyExpression, ?x, tr(value))</td><td>x is a variable name</td></tr>
<tr><td>14</td><td>tr$_O$(
SubObjectPropertyOf(ObjectPropertyChain(PropertyExpression$_1$ ... PropertyExpression$_m$) PropertyExpression$_0$)
)</td><td>Forall ?x ?y$_1$ ... ?y$_m$
(tr$_O$(PropertyExpression$_1$, ?x, ?y$_m$) :- And(
tr$_O$(PropertyExpression$_1$, ?x, ?y$_1$)
tr$_O$(PropertyExpression$_2$, ?y$_1$, ?y$_2$)
...
tr$_O$(PropertyExpression$_m$, ?y$_{m-1}$, ?y$_m$)))</td><td></td></tr>
<tr><td>15</td><td>tr$_O$(
Sub[Object|Data]PropertyOf(PropertyExpression$_1$ PropertyExpression$_2$)
)</td><td>Forall ?x ?y (tr$_O$(PropertyExpression$_2$, ?x, ?y) :- tr$_O$(PropertyExpression$_1$, ?x, ?y))</td><td>PropertyExpression$_1$ contains no mention of ObjectPropertyChain</td></tr>
<tr><td>16</td><td>tr$_O$(
Disjoint[Object|Data]Properties(PropertyExpression$_1$ PropertyExpression$_2$)
)</td><td>Forall ?x ?y (rif:error :-
And(tr$_O$(PropertyExpression$_1$, ?x, ?y)
tr$_O$(PropertyExpression$_2$, ?x, ?y)))</td><td></td></tr>
<tr><td>17</td><td>tr$_O$(
InverseObjectProperties(PropertyExpression$_1$ PropertyExpression$_2$)
)</td><td>Forall ?x ?y (tr$_O$(PropertyExpression$_2$, ?y, ?x) :- tr$_O$(PropertyExpression$_1$, ?x, ?y))
Forall ?x ?y (tr$_O$(PropertyExpression$_1$, ?y, ?x) :- tr$_O$(PropertyExpression$_2$, ?x, ?y))</td><td></td></tr>
<tr><td>18</td><td>tr$_O$(
Functional[Object|Data]Property(PropertyExpression)
)</td><td>Forall ?x ?y$_1$?y$_2$ (?y$_1$=?y$_2$:-
And(tr$_O$(PropertyExpression, ?x, ?y$_1$)
tr$_O$(PropertyExpression, ?x, ?y$_2$)))</td><td></td></tr>
<tr><td>19</td><td>tr$_O$(
InverseFunctional[Object|Data]Property(PropertyExpression)
)</td><td>Forall ?x$_1$?x$_2$?y (?x$_1$=?x$_2$:-
And(tr$_O$(PropertyExpression, ?x$_1$, ?y)
tr$_O$(PropertyExpression, ?x$_2$, ?y)))</td><td></td></tr>
<tr><td>20</td><td>tr$_O$(
IrreflexiveObjectProperty(PropertyExpression)
)</td><td>Forall ?x (rif:error :-
tr$_O$(PropertyExpression, ?x, ?x))</td><td></td></tr>
<tr><td>21</td><td>tr$_O$(
SymmetricObjectProperty(PropertyExpression)
)</td><td>Forall ?x ?y (tr$_O$(PropertyExpression, ?y, ?x) :- tr$_O$(PropertyExpression, ?x, ?y))</td><td></td></tr>
<tr><td>22</td><td>tr$_O$(
AsymmetricObjectProperty(PropertyExpression)
)</td><td>Forall ?x ?y (rif:error :-
And(tr$_O$(PropertyExpression, ?x, ?y)
tr$_O$(PropertyExpression, ?y, ?x)))</td><td></td></tr>
</table>

23	tr_O(`TransitiveObjectProperty(PropertyExpression)`)	`Forall ?x ?y ?z (`tr_O`(PropertyExpression, ?x, ?z) :- And(`tr_O`(PropertyExpression, ?x, ?y)` tr_O`(PropertyExpression, ?y, ?z)))`
24	tr_O(`DatatypeDefinition(datatypeIRI DataRange)`)	`Forall ?x (?x[rdf:type -> tr(datatypeIRI)] :- `tr_O`(DataRange, ?x))` `Forall ?x ( `tr_O`(DataRange, ?x):- ?x[rdf:type -> tr(datatypeIRI)] )`
25	tr_O(`HasKey(subClassExpression PropertyExpression`$_1$` ... PropertyExpression`$_m$`)`)	`Forall ?x ?y ?z`$_1$` ... ?z`$_m$` (?x=?y :- And(`tr_O`(subClassExpression, ?x)` tr_O`(subClassExpression, ?y)` tr_O`(PropertyExpression`$_1$`, ?x, ?z`$_1$`)` ... tr_O`(PropertyExpression`$_1$`, ?x, ?z`$_m$`)` tr_O`(PropertyExpression`$_1$`, ?y, ?z`$_1$`)` ... tr_O`(PropertyExpression`$_1$`, ?y, ?z`$_m$`)))`
26	tr_O(`SameIndividual(Individual`$_1$` Individual`$_2$`)`)	`tr(Individual`$_1$`)=tr(Individual`$_2$`)`
27	tr_O(`DifferentIndividuals(Individual`$_1$` Individual`$_2$`)`)	`rif:error` :- `tr(Individual`$_1$`)=tr(Individual`$_2$`)`

Besides the embedding in the previous table, we also need an axiomatization of some of the aspects of the OWL 2 Direct Semantics, e.g., separation between individual and datatype domains. This axiomatization is defined relative to an OWL Vocabulary *V*, which includes all well-typed literals used in the rules, and a datatype map D, which includes all considered datatypes. In the table, for a given datatype d, L2V(d) is the lexical-to-value mapping of d.

$R^{OWL\text{-}Direct}(V,R)$	=	merge({ (i) (`Forall ?x (rif:error :- ?x[rdf:type -> owl:Nothing])`), (ii) `Forall ?x (rif:error :- And(?x[rdf:type -> rdfs:Literal] ?x[rdf:type -> owl:Thing]))`, (iii) (`Forall ?x (?x[rdf:type -> owl:Thing] :- ?x[rdf:type -> C])`) for every class ID C, (iv) (`Forall ?x ?y (?x[rdf:type -> owl:Thing] :- ?x[P -> ?y])`) for every property ID P, (v) (`Forall ?x ?y (?y[rdf:type -> owl:Thing] :- ?x[P -> ?y])`) for every object property ID P, (vi) (`Forall ?x ?y (?y[rdf:type -> rdfs:Literal] :- ?x[P -> ?y])`) for every data property ID P, (vii) (tr(i)`[rdf:type -> owl:Thing]`) for every IRI i in *V*, (viii) (tr(s^^u)`[rdf:type -> u']`) for every well-typed literal s^^u and datatype identifier `u'` in *V* such that L2V(D(u))(s) is in the value space of `u'`, (ix) (`rif:error :- `tr(s^^u)`[rdf:type -> u']`) for every well-typed literal s^^u and datatype identifier `u'` in *V* such that L2V(D(u))(s) is not in the value space of `u'`, (x) (`Forall ?x (?x[rdf:type -> rdfs:Literal] :- ?x[rdf:type -> `*`Diri`*`])`) for every datatype in D with identifier *Diri*, (xi) `"a"="b" :- rif:error`})

We call an OWL 2 RL ontology *O normalized* if it is the same as its normalization, i.e., *O*=tr_N(*O*).

The following lemma establishes faithfulness of the embedding.

Normalized Combination Embedding Lemma Given a datatype map D conforming with T, a RIF-OWL DL-combination C=<*R*,{O_1,...,O_n}>, where {O_1,...,O_n} is an imports-closed set of normalized OWL 2 RL ontologies with vocabulary *V*, RIF-OWL Direct-entails a DL-condition φ with respect to D iff merge({*R*', $R^{OWL\text{-}Direct}$(*V*), tr_O(O_1), ..., tr_O(O_n)}) dl-entails φ, where *R*' is like *R*, except that every subformula of the form a#b has been replaced with `a[rdf:type -> b]`.

Proof. We prove both directions by contraposition.
In the proof we abbreviate merge({*R*', $R^{OWL\text{-}Direct}$(*V*), tr_O(O_1), ..., tr_O(O_n)} with R*.

(=>) Assume R* does not dl-entail φ. This means there is a dl-semantic multi-structure ***Î*** that is a model of R* but ***I*** = <***TV***, ***DTS***, ***D***, $\boldsymbol{D}_{ind}$, $\boldsymbol{D}_{func}$, $\boldsymbol{I}_C$, $\boldsymbol{I}_{C'}$, $\boldsymbol{I}_V$, $\boldsymbol{I}_F$, $\boldsymbol{I}_{frame}$, $\boldsymbol{I}_{NF}$, $\boldsymbol{I}_{sub}$, $\boldsymbol{I}_{isa}$, $\boldsymbol{I}_{=}$, $\boldsymbol{I}_{external}$, $\boldsymbol{I}_{truth}$> is not a model of φ.
We call a structure ***I*** *named for* R* if for every object k ∈ $\boldsymbol{D}_{ind}$ that is not in the value space of some datatype in ***DTS***, k=$\boldsymbol{I}_C$(c), where c is either an IRI identifying an individual in *V* or a constant appearing as an individual in *R*. This definition extends naturally to dl-semantic multi-structures.
We now show that there is a named dl-semantic multi-structure ***Î'*** that is a model of R* such that ***I'*** is not a model of φ.
The set of unnamed individuals in ***I*** is the set of objects k ∈ $\boldsymbol{D}_{ind}$ that are not in the value space of some datatype in ***DTS***, and there is no IRI c identifying an individual in *V* or constant c appearing as an individual in *R* such that k=$\boldsymbol{I}_C$(c).
Let ***Î'*** be obtained from ***Î*** by removing all unnamed individuals from $\boldsymbol{D}_{ind}$ and removing the corresponding tuples in the domains and ranges of the various mapping functions in the structures in ***Î***. Clearly, ***I'*** is not a model of φ: condition formulas do not contain negation, and so every condition formula that is satisfied by ***I'*** is also satisfied by ***I***.
Consider any rule r in R*. If r is a variable-free rule implication or atomic formula it is clearly satisfied ***Î'***, by satisfaction of r in ***Î*** are construction of ***Î'***. A universal fact can be seen as a rule with the empty condition `And()`. Let r be a rule with a condition ψ that is satisfied by ***I'***. Since ψ does not contain negation, ψ is also satisfied by ***I***. Now, if every variable in the conclusion of r appears also in the condition ψ, every variable is mapped to a named individual, and thus the conclusion is satisfied by satisfaction in ***I*** and construction of ***I'***. Now, if there is a variable ?x in the conclusion that does not appear in ψ, ***I*** satisfies the conclusion for every assignment of ?x to any element in $\boldsymbol{D}_{ind}$. Since the individual domain of ***I'*** is a strict subset of $\boldsymbol{D}_{ind}$, the conclusion is also satisfied in ***I'***. Therefore, ***Î'*** is a model of R*. In the remainder we assume ***Î***=***Î'***.
We define CExt(c)={u | u ∈ $\boldsymbol{D}_{ind}$ and $\boldsymbol{I}_{truth}$($\boldsymbol{I}_{frame}$(u)(`rdf:type`, $\boldsymbol{I}_C$(c)))=**t**} as the class extension of the constant c. Furthermore, we define $\boldsymbol{D}_D$=(union of the value spaces of all datatypes in D).
Consider the pair (***Î****,I), where ***Î**** is obtained from ***Î*** as follows: ***I**** = <***TV***, ***DTS***, ***D****, $\boldsymbol{D^*}_{ind}$, $\boldsymbol{D}_{func}$, $\boldsymbol{I^*}_C$, $\boldsymbol{I}_{C'}$, $\boldsymbol{I}_V$, $\boldsymbol{I}_F$, $\boldsymbol{I^*}_{frame}$, $\boldsymbol{I}_{NF}$, $\boldsymbol{I}_{sub}$, $\boldsymbol{I}_{isa}$, $\boldsymbol{I}_{=}$, $\boldsymbol{I}_{external}$, $\boldsymbol{I}_{truth}$>, where t, f ∈ ***D**** such that $\boldsymbol{I}_{truth}$(t)=**t** and $\boldsymbol{I^*}_{truth}$(f)=**f**, and

- $\boldsymbol{D^*}_{ind}$=$\boldsymbol{D}_{ind}$ union (union of the value spaces of all datatypes in D);
- ***D****=***D*** union $\boldsymbol{D^*}_{ind}$;
- $\boldsymbol{I^*}_C$ is like $\boldsymbol{I}_C$ except that it maps all constants with symbol spaces in D***DTS*** to the values in the in the corresponding datatypes, according to the respective lexical-to-value mappings;
- $\boldsymbol{I^*}_{frame}$ is defined as follows:
  - $\boldsymbol{I^*}_{frame}$(k)($\boldsymbol{I}_{C'}$(`rdf:type`),$\boldsymbol{I}_{C'}$(`rdfs:Literal`))=t if k ∈ $\boldsymbol{D}_D$, otherwise $\boldsymbol{I^*}_{frame}$(k)($\boldsymbol{I}_{C'}$(`rdf:type`),$\boldsymbol{I}_{C'}$(`rdfs:Literal`))=f,
  - otherwise $\boldsymbol{I^*}_{frame}$ is like $\boldsymbol{I}_{frame}$;

and I=< IR, LV, C, OP, DP, I, DT, LT, FA > is a tuple defined as follows:

- LV=$\boldsymbol{D}_D$,

- IR=$\boldsymbol{D}_{\text{ind}}$\LV,
- DT(`rdfs:Literal`)=LV,
- DT(`d'`) = the value space of D(`d'`), if `d'` is a datatype identifier in *V* in the domain of D,
- DT(`d'`) = set of all objects k such that $\boldsymbol{I}_{\text{truth}}(\boldsymbol{I}_{\text{frame}}$(k)($\boldsymbol{I}_{C'}$(`rdf:type`),$\boldsymbol{I}_{C'}$(<c>))) = **t**, for every datatype identifier `d'` in *V*, not in the domain of D,
- C(c) = set of all objects k such that $\boldsymbol{I}_{\text{truth}}(\boldsymbol{I}_{\text{frame}}$(k)($\boldsymbol{I}_{C'}$(`rdf:type`),$\boldsymbol{I}_{C'}$(<c>))) = **t**, for every class identifier in *V*,
- OP(p) = set of all pairs (k, `l`) such that $\boldsymbol{I}_{\text{truth}}(\boldsymbol{I}_{\text{frame}}$(k)($\boldsymbol{I}_{C'}$(<p>), `l`))) = **t** (true), for every object property identifier p in *V* and not in {`owl:topObjectProperty`,`owl:bottomObjectProperty`};
- OP(`owl:topObjectProperty`) = IR × IR;
- OP(`owl:bottomObjectProperty`) = { };
- DP(p) = set of all pairs (k, `l`) such that $\boldsymbol{I}_{\text{truth}}(\boldsymbol{I}_{\text{frame}}$(k)($\boldsymbol{I}_{C'}$(<p>), `l`))) = **t** (true), for every datatype property identifier p in *V* and not in {`owl:topDataProperty`,`owl:bottomDataProperty`};
- OP(`owl:topDataProperty`) = IR × LV;
- OP(`owl:bottomDataProperty`) = { };
- LT((s, d)) = $\boldsymbol{I}_C$(`"s"^^d`) for every well-typed literal (s, d) in *V*;
- I(i) = $\boldsymbol{I}_C$(<i>) for every IRI `i` identifying an individual in *V*;
- FA is the empty mapping.

When referring to rules in the remainder we mean rules in $R^{OWL\text{-}Direct}(V,R)$, unless otherwise specified.
We have that ***I**** has a separation between the object and data domains: (+) each object is either in CExt(`owl:Thing`) or in CExt(`rdfs:Literal`) and $\boldsymbol{D}_D$: each non-data value in $\boldsymbol{D}_{\text{ind}}$ is in CExt(`owl:Thing`) by rule (vii) and the fact that ***I**** is a named structure, and each data value is in CExt(`rdfs:Literal`) by construction of ***I****. The two sets are distinct by satisfaction of rule (ii) in ***I***.
It is straightforward to see that $\hat{\boldsymbol{I}}$***** is a model of R* and ***I**** is not a model of φ.
According to its definition, an interpretation with respect to a datatype map D must fulfill the following conditions, where L(d) denotes the lexical space, V(d) denotes the value space and L2V(d) denotes to lexical-to-value mapping of a datatype d:

1. IR is a nonempty set,
2. LV is a nonempty set disjoint with IR and including the value spaces of all datatypes in D,
3. C : $V_C \rightarrow 2^{\text{IR}}$
4. DT : $V_D \rightarrow 2^{\text{LV}}$, where DT is the same as in D for each datatype d
5. OP : $V_{IP} \rightarrow 2^{\text{IR}\times\text{IR}}$
6. DP : $V_{DP} \rightarrow 2^{\text{IR}\times\text{LV}}$
7. LT : TL → LV, where TL is the set of typed literals in V_L and LT((s,d))=L2V(d)(s), for every typed literal (s,d) ∈ V_L
8. I : V_I → IR
9. C(`owl:Thing`) = IR
10. C(`owl:Nothing`) = { }
11. OP(`owl:topObjectProperty`) = IR × IR
12. DP(`owl:topDataProperty`) = IR × LV
13. OP(`owl:bottomObjectProperty`) = { }
14. DP(`owl:bottomDataProperty`) = { }
15. DT(`rdfs:Literal`) = LV

Condition 1 is met because $\boldsymbol{D_{ind}}$ is a nonempty set. Clearly LV disjoint with IR and contains the value space for each datatype in D; therefore, condition 2 is met. Conditions 3 through 9 and 11 through 15 are met by the definitions of ***I**** and I, and the property (+). Finally, condition 10 is satisfied by satisfaction of rule (i) in ***I***. This establishes the fact that I is an interpretation.

Consider now any ontology *O* in {O_1,...,O_n}. To establish that I satisfies *O*, we need to establish that each axiom in the axiom closure of *O* is satisfied in I w.r.t. *O*. Note that, since *O* is normalized, it does not contain import statements, and thus the axiom closure of *O* is equal to *O*.
Consider any axiom d in *O*; d has one of the following forms (cf. the second column of Table Normalization of OWL 2 RL ontologies):

1. subproperty statement,
2. disjoint properties statement,
3. inverse property statement,
4. functional property statement,
5. symmetric property statement,
6. transitive property statement,
7. datatype definition,
8. has-key statement,
9. same-individual statement,
10. different-individuals statement, or
11. subclass statement `SubClassOf(`*X Y*`)`.

Consider a subproperty statement `SubObjectPropertyOf(p q)` and a pair (`k, l`) in OP(<p>). Then, by construction of I, $\boldsymbol{I}_{\text{truth}}(\boldsymbol{I}_{\text{frame}}$(k)($\boldsymbol{I}_{C'}$(<p>), `l`))) = **t**. But, by tr(d), it must be the case that also $\boldsymbol{I}_{\text{truth}}(\boldsymbol{I}_{\text{frame}}$(k)($\boldsymbol{I}_{C'}$(<q>), `l`))) = **t**. But then, (`k`**,**`l`) must be in OP(<q>), by construction of I. This argument extends straightforwardly to subproperty statements with inverse or property-chain expressions. So, I satisfies d. Similar for statements of the forms 2--6.
Consider a datatype definition `DatatypeDefinition( d e )`, with d, e IRIs. This axiom is satisfied in I if *DT(d) = DT(e)*. This definition is translated to the rules Forall ?x (?x[rdf:type -> e] :- ?x[rdf:type -> d]) Forall ?x (?x[rdf:type -> d] :- ?x[rdf:type -> e]) By satisfaction of these rules in ***I**** and construction of I we have that I satisfies the datatype definition. This argument straightforwardly extends to more complex datatype definitions; recall that the only construct available in OWL 2 RL datatype definitions is intersection.
Consider a has-key axiom d. We have that every object in $\boldsymbol{D}^{\boldsymbol{*}}_{\text{ind}}$, and thus also every object in IR is named. It is then straightforward to verify that if tr_O(d) is satisfied in ***I****, the condition in Section 2.3.5 of [OWL2-Semantics] is satisfied. Analogous for same-individual and different-individual axioms.
Consider the case that d is a subclass statement `SubClassOf(`*X Y*`)` and consider any k in C(*X*), where C is as in the Class Expressions Table in [OWL2-Semantics]. We show, by induction, that ***I**** satisfies tr_O(*X*) when ?x is assigned to k.
If *X* is a classID, then satisfaction of tr(*X*) follows by an analogous argument as that for directives of form 1. Similar for value restrictions. If *X* is a some-value restriction of type *Z* on a property p, then there must be some object `l` such that (k,`l`) in OP(p) such that `l` is in C(*Z*). By induction we have satisfaction of tr(*Z*) for some variable assignment. Then, by definition of I, we have $\boldsymbol{I}_{\text{truth}}(\boldsymbol{I}_{\text{frame}}$(k)($\boldsymbol{I}_{C'}$(<p>), `l`)) = **t** (true), thereby establishing satisfaction of tr_O(*X*) in ***I****. This extends straightforwardly to union, intersection, and one-of descriptions.
By satisfaction of tr_O(d), we have that tr_O(*Y*) is necessarily satisfied for ?x assigned to k. By an argument analogous to the argument above, we obtain that k is in C(*Y*).
This establishes satisfaction of d in I.

We obtain that every directive is satisfied in I. Therefore, *O*, and thus every ontology in C, is satisfied in I. We have established earlier that ***I**** satisfies *R* and not φ, so (***I****, I) satisfies *R* and not φ. We conclude that C does not entail φ.

(<=) Assume C does not RIF-OWL Direct-entail φ. This means there is a common-RIF-OWL Direct-interpretation ($\hat{\boldsymbol{I}}$, I) that is a RIF-OWL Direct-model of C, but ***I*** is not a model of φ. To show that R* does not entail φ, we show that ***I*** is a model of R*.
R is satisfied in ***I*** by assumption. Satisfaction of $\text{tr}_O(O_i)$ can be shown analogously to establishment of satisfaction in I of O_i in the (=>) direction. We now establish satisfaction of the rules in $R^{OWL\text{-}Direct}(V,R)$.
(i) follows immediately from the fact that C(`owl:Nothing`)={}. (ii) follows from conditions 2, 9, and 15 on interpretations. (iii) follows from conditions 3 and 9. (iv) follows from conditions 5, 6 and 9. (v) follows from conditions 5 and 9. (vi) follows from conditions 6 and 15. (vii) follows from conditions 8 and 9. (viii) and (ix) follow from condition 7. (x) follows from conditions 4 and 15.

This establishes satisfaction of $R^{OWL\text{-}Direct}(V,R)$, and thus R*, in ***I***. Therefore, R* does not entail φ. ☐

The following theorems establish faithfulness of the full embedding of RIF-OWL 2 RL combinations into RIF.

Theorem Given a datatype map D conforming with T, a RIF-OWL DL-combination C=<*R*,{O_1,...,O_n}>, where {O_1,...,O_n} is an imports-closed set of OWL 2 RL ontologies with Vocabulary *V*, RIF-OWL Direct-entails a DL-condition formula φ with respect to D iff tr(merge({*R*, $R^{OWL\text{-}Direct}(V)$, $\text{tr}_O(\text{tr}_N(O_1))$, ..., $\text{tr}_O(\text{tr}_N(O_n))$})) entails tr(φ).

Proof. By the RIF-BLD DL-document formula Lemma,
tr(merge({*R*, $R^{OWL\text{-}Direct}(V,R)$, $\text{tr}_O(\text{tr}_N(O_1))$, ..., $\text{tr}_O(\text{tr}_N(O_n))$})) entails tr(φ) iff merge({*R*, $R^{OWL\text{-}Direct}(V,R)$, $\text{tr}_O(\text{tr}_N(O_1))$, ..., $\text{tr}_O(\text{tr}_N(O_n))$}) dl-entails φ.
Observe that the mapping tr() does not distinguish between frame formulas of the form `a[rdf:type -> b]` and membership formulas a#b. We may thus safely

W3C Editor's Draft

assume that R has no occurrences of the latter. Then, by the Normalized Combination Embedding Lemma,
merge({R, $R^{OWL\text{-}Direct}(V,R)$, $tr_O(tr_N(O_1))$, ..., $tr_O(tr_N(O_n))$}) dl-entails φ iff <R,{$tr_N(O_1)$,...,$tr_N(O_n)$}> RIF-OWL Direct-entails φ.
Finally, by the Normalization Lemma,
<R,{$tr_N(O_1)$,...,$tr_N(O_n)$}> RIF-OWL Direct-entails φ iff C=<R,{O_1,...,O_n}> RIF-OWL Direct-entails φ.
This chain of equivalences establishes the theorem. ☐

Theorem Given a datatype map D conforming with T, a RIF-OWL DL-combination <R,{O_1,...,O_n}>, where {O_1,...,O_n} is an imports-closed set of OWL 2 RL ontologies with Vocabulary V, is RIF-OWL Direct-satisfiable with respect to D iff tr(merge({R, $R^{OWL\text{-}Direct}(V)$, $tr_O(tr_N(O_1))$, ..., $tr_O(tr_N(O_n))$})) does not entail `rif:error`.

Proof. The theorem follows immediately from the previous theorem and the observation that a combination (respectively, document) is RIF-OWL Direct-satisfiable (respectively, has a model) if and only if it does not entail the condition formula `"a"="b"`. ☐

10 Appendix: Change log (Informative)

Changes since the 11 May 2010 Proposed Recommendation.

In the table in Section 9.2.2.2: The expression $tr_O(X, ?y_n)$ has been added to the third row, second column; omitting this expression had been an oversight. Rows 3--5 did not account for inverse properties; this had been rectified. For the purpose of understandability, rows 3a, 4a, 5a have been added to make the case n=0 of rows 3, 4, 5 explicit.

Changes since the 22 June 2010 Recommendation.

Added a clarification to Section 9 on the restriction for subclass preventing embedding.

11 End Notes

RDF URI References: There are certain RDF URI references that are not IRIs (e.g., those containing spaces). It is possible to use such RDF URI references in RDF graphs that are combined with RIF rules. However, such URI references cannot be represented in RIF rules and their use in RDF is discouraged.

Generalized RDF graphs: Standard RDF graphs, as defined in [RDF-Concepts], do not allow the use of literals in subject and predicate positions and blank nodes in predicate positions. The RDF Core working group has listed two issues questioning the restrictions that literals may not occur in subject and blank nodes may not occur in predicate positions in triples. Anticipating lifting of these restrictions in a possible future version of RDF, we use the more liberal notion of *generalized* RDF graph. We note that the definitions of interpretations, models, and entailment in the RDF Semantics document [RDF-Semantics] also apply to such generalized RDF graphs.

We note that every standard RDF graph is a generalized RDF graph. Therefore, our definition of combinations applies to standard RDF graphs as well.

We note also that the notion of generalized RDF graphs is more liberal than the notion of RDF graphs used by SPARQL; generalized RDF graphs additionally allow blank nodes and literals in predicate positions.