W3C Editor's Draft

RIF Basic Logic Dialect

W3C Editor's Draft 11 May 2010

This version:
http://www.w3.org/2005/rules/wg/draft/ED-rif-bld-20100511/
Latest editor's draft:
http://www.w3.org/2005/rules/wg/draft/rif-bld/
Previous version:
http://www.w3.org/2005/rules/wg/draft/ED-rif-bld-20100510/ (color-coded diff)
Editors:
Harold Boley, National Research Council Canada
Michael Kifer, State University of New York at Stony Brook, USA

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

Abstract

This document, developed by the Rule Interchange Format (RIF) Working Group, specifies the Basic Logic Dialect, RIF-BLD, a format that allows logic rules to be exchanged between rule systems. The RIF-BLD presentation syntax and semantics are specified both directly and as specializations of the *RIF Framework for Logic Dialects*, or RIF-FLD. The XML serialization syntax of RIF-BLD is specified via a mapping from the presentation syntax. A normative XML schema is also provided.

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May Be Superseded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/.

W3C Editor's Draft

Set of Documents

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

1. RIF Overview
2. RIF Core Dialect
3. RIF Basic Logic Dialect (this document)
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
8. OWL 2 RL in RIF
9. RIF Combination with XML data
10. RIF Test Cases

XML Schema Datatypes Dependency

RIF is defined to use datatypes defined in the XML Schema Definition Language (XSD). As of this writing, the latest W3C Recommendation for XSD is version 1.0, with version 1.1 progressing toward Recommendation. RIF has been designed to take advantage of the new datatypes and clearer explanations available in XSD 1.1, but for now those advantages are being partially put on hold. Specifically, until XSD 1.1 becomes a W3C Recommendation, the elements of RIF which are based on it should be considered *optional*, as detailed in Datatypes and Builtins, section 2.3. Upon the publication of XSD 1.1 as a W3C Recommendation, those elements will cease to be optional and are to be considered required as otherwise specified.

We suggest that for now developers and users follow the XSD 1.1 Last Call Working Draft. Based on discussions between the Schema, RIF and OWL Working Groups, we do not expect any implementation changes will be necessary as XSD 1.1 advances to Recommendation.

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 8 June 2010

The W3C Director seeks review and feedback from W3C Advisory Committee representatives, via their review form by 8 June 2010. This will allow the Director to assess consensus and determine whether to issue this document as a W3C Recommendation.

Others are encouraged by the Rule Interchange Format (RIF) Working Group to continue to send reports of implementation experience, and other feedback, to

W3C Editor's Draft

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).

Support

The advancement of this Proposed Recommendation is supported by the disposition of comments on the Candidate Recommendation, the Test Suite, and the list of implementations.

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

This specification develops ***RIF-BLD*** (the **B**asic **L**ogic **D**ialect of the **R**ule **I**nterchange **F**ormat). From a theoretical perspective, RIF-BLD corresponds to the language of definite Horn rules with equality and a standard first-order semantics [CL73]. Syntactically, RIF-BLD has a number of extensions to support features such as objects and frames as in F-logic [KLW95], internationalized resource identifiers (or IRIs, defined by [RFC-3987]) as identifiers for concepts, and XML Schema datatypes [XML-SCHEMA2]. In addition, RIF RDF and OWL Compatibility [RIF-RDF+OWL] defines semantics for the integrated RIF-BLD/RDF and RIF-BLD/OWL languages. These features make RIF-BLD a Web-aware language. However, it should be kept in mind that RIF is designed to enable interoperability among rule languages in general, and its uses are not limited to the Web.

While rule interchange (and not, e.g., execution) is the principle design goal for RIF-BLD, the design clearly indicates a decision to avoid solving the (probably impossible) problem of rule interchange in general. Instead, the design of RIF reflects the rationale of identifying specific kinds of rules within existing rule

systems, called *RIF dialects*, that can be translated into other rule systems without changing their meaning. RIF-BLD is just the first in a series of such dialects. In particular, RIF-BLD has the RIF-Core dialect [RIF-Core] as a subset. It is *not expected* that most rule systems will be able to translate all their rules into RIF-BLD, rather it is expected that only certain kinds of rules will be translatable. Since there are many existing rule languages with useful features that are not supported in RIF-BLD, it is expected that RIF-BLD translators will not translate rules that use such features. This could drive the design of "BLD-specific" rule sets in which rules are specifically written by the implementor to be within the BLD dialect and thus be portable between many rule system implementations.

Among its many influences, RIF shares certain characteristics with ISO Common Logic (CL) [ISO-CL], itself an evolution of KIF [KIF] and Conceptual Graphs [CG]. Like CL, RIF employs XML as its primary normative syntax, uses IRIs as identifiers, specifies integrated RIF-BLD/RDF and RIF-BLD/OWL languages for Semantic Web Compatibility [RIF-RDF+OWL], and provides a rich set of datatypes and built-ins that are designed to be well aligned with Web-aware rule system implementations [RIF-DTB]. Unlike CL, RIF-BLD was designed to be a *simple* dialect with limited expressiveness that lies within the intersection of first-order and logic-programming systems. This is why RIF-BLD does not support negation. More generally, RIF-BLD is part of a coherent array of RIF rule dialects, which encompasses both logic rules -- through the RIF framework for logic dialects [RIF-FLD] also including a variety of rule languages based on non-monotonic theories -- and production rules, as defined in [RIF-PRD]. CL, on the other hand, is strictly first-order; it does not account for non-monotonic semantics (e.g. negation as failure, defaults, priorities, etc.). For rule interchange between CL and RIF dialects, it is expected that partial RIF-CL mappings will be defined.

RIF-BLD also bears some similarity to SPARQL, in particular with respect to RDF Compatibility [RIF-RDF+OWL]. As with the well-known correspondence between a fragment of SQL and Datalog, SPARQL can be partially mapped to Datalog (and thus to the RIF-Core subset of RIF-BLD), see [AP07] and [AG08] for details. A full mapping of SPARQL would need constructs beyond RIF-BLD, such as non-monotonic negation. Likewise, not all of SPARQL's FILTER functions are expressible in RIF-DTB built-in predicates. Not all of RIF-BLD is expressible in SPARQL either, for instance recursive rules over RDF Data are not expressible as SPARQL CONSTRUCT statements.

RIF-BLD is defined in two different ways -- *both normative*:

- As a direct specification, independently of the RIF framework for logic dialects [RIF-FLD], for the benefit of those who desire a direct path to RIF-BLD, e.g., as prospective implementers, and are not interested in extensibility issues. This version of the RIF-BLD specification is given first.
- As a specialization of the RIF framework for logic dialects [RIF-FLD], which is part of the RIF extensibility framework. Building on RIF-FLD, this version of the RIF-BLD specification is comparatively short and is presented in Section RIF-BLD as a Specialization of the RIF Framework at the end of this document. This is intended for the reader who is already

familiar with RIF-FLD and does not need to go through the much longer direct specification of RIF-BLD. This section is also useful for dialect designers, as it is a concrete example of how a non-trivial RIF dialect can be derived from the RIF framework for logic dialects.

Logic-based RIF dialects that specialize or extend RIF-BLD in accordance with the RIF framework for logic dialects [RIF-FLD] include RIF-Core as a specialization of RIF-BLD. It is expected that other specifications will develop further logic dialects based on RIF-BLD such as a RIF-BLD extension capturing uncertainty [URD08].

As a preview, here is a simple complete RIF-BLD example deriving a ternary relation from its inverse.

Example 1 (An introductory RIF-BLD example).

A rule can be written in English to derive the `buy` relationships (rather than store them) from the `sell` relationships that are stored as facts (e.g., as exemplified by the English statement below):

- *A buyer buys an item from a seller if the seller sells the item to the buyer.*
- *John sells LeRif to Mary.*

Intuitively, the fact *Mary buys LeRif from John* should be logically derivable from the above premises. Assuming Web IRIs for the predicates `buy` and `sell`, as well as for the individuals `John`, `Mary`, and `LeRif`, the above English text can be represented in the RIF-BLD Presentation Syntax as follows.

```
Document(
  Base(<http://example.com/people#>)
  Prefix(cpt <http://example.com/concepts#>)
  Prefix(bks <http://example.com/books#>)

  Group
  (
    Forall ?Buyer ?Item ?Seller (
        cpt:buy(?Buyer ?Item ?Seller) :- cpt:sell(?Seller ?Item ?Buyer)
    )

    cpt:sell(<John> bks:LeRif "Mary"^^rif:iri)
  )
)
```

Note that IRIs are represented in several different ways in this example. First, the [CURIE] notation `prefix:suffix` is used to shorten IRI representation. For instance, `cpt:buy` via a `Prefix` directive represents the `rif:iri` constant `"http://example.com/concepts#buy"^^rif:iri`. Another way to shorten this IRI constant is to use the angle-bracketed notation `<http://example.com/concepts#buy>`. The `Base` directive provides yet another shortcut: it applies to all relative IRIs, such as `"Mary"^^rif:iri` and `<John>`. The `Base` directive

expands these relative IRIs to `"http://example.com/people#Mary"^^rif:iri` and `"http://example.com/people#John"^^rif:iri`, respectively.

Whenever a RIF-BLD document falls into the Core subset or can be translated to it, the document should be produced in RIF-Core to allow its interchange with a maximum number of RIF consumers. For instance, the Datalog-like RIF document in Example 1 is also a RIF-Core example.

For the interchange of documents containing RIF-BLD rules (and facts), a concrete RIF-BLD XML Syntax is given in this specification. To formalize their meaning, a model-theoretic RIF-BLD Semantics is specified.

2 Direct Specification of RIF-BLD Presentation Syntax

This section specifies the ***presentation syntax*** of RIF-BLD directly, without relying on [RIF-FLD]. In the first five (normative) subsections, the presentation syntax is defined using "mathematical English," a special form of English for communicating mathematical definitions, examples, etc. In the non-normative subsection EBNF Grammar for the Presentation Syntax of RIF-BLD, a grammar for a superset of the presentation syntax is given using Extended Backus–Naur Form (EBNF). Neither the mathematical English nor the EBNF is intended to be a concrete syntax for RIF-BLD. The mathematical English deliberately leaves out details such as the delimiters of the various syntactic components, escape symbols, parenthesizing, precedence of operators, and the like. The EBNF does not specify context-sensitive syntactic constraints. Since RIF is an interchange format, it uses ***XML as the only concrete syntax***, which will be defined in XML Serialization Syntax for RIF-BLD. Hence RIF-BLD conformance is described in terms of semantics-preserving transformations.

Note to the reader: this section depends on Section Constants, Symbol Spaces, and Datatypes of [RIF-DTB].

2.1 Alphabet of RIF-BLD

Definition (Alphabet). The ***alphabet*** of the presentation language of RIF-BLD consists of

- a countably infinite set of ***constant symbols*** `Const`
- a countably infinite set of ***variable symbols*** `Var` (disjoint from `Const`)
- a countably infinite set of argument names, `ArgNames` (disjoint from `Const` and `Var`)
- connective symbols `And`, `Or`, and `:-`

- quantifiers `Exists` and `Forall`
- the symbols `=`, `#`, `##`, `->`, `External`, `Import`, `Prefix`, and `Base`
- the symbols `Group` and `Document`
- the symbols for representing lists: `List` and `OpenList`.
- the auxiliary symbols `(`, `)`, `[`, `]`, `<`, `>`, and `^^`

The set of connective symbols, quantifiers, `=`, etc., is disjoint from `Const` and `Var`. The argument names in `ArgNames` are written as Unicode strings that must not start with a question mark, "`?`". Variables are written as Unicode strings preceded with the symbol "`?`".

Constants are written as `"literal"^^symspace`, where `literal` is a sequence of Unicode characters and `symspace` is an identifier for a symbol space. Symbol spaces are defined in Section Constants, Symbol Spaces, and Datatypes of [RIF-DTB].

The symbols `=`, `#`, and `##` are used in formulas that define equality, class membership, and subclass relationships. The symbol `->` is used in terms that have named arguments and in frame formulas. The symbol `External` indicates that an atomic formula or a function term is defined externally (e.g., a built-in) and the symbols `Prefix` and `Base` enable compact representations of IRIs [RFC-3987].

The symbol `Document` is used to specify RIF-BLD documents, the symbol `Import` is an import directive, and the symbol `Group` is used to organize RIF-BLD formulas into collections. □

The language of RIF-BLD is the set of formulas constructed using the above alphabet according to the rules given below.

2.2 Terms

RIF-BLD defines several kinds of terms: *constants* and *variables*, *positional* terms, terms with *named arguments*, plus *equality*, *membership*, *subclass*, *frame*, and *external* terms. The word "*term*" will be used to refer to any of these constructs.

To simplify the next definition, we will use the phrase *base term* to refer to simple, positional, or named-argument terms, or to terms of the form `External(t)`, where `t` is a positional or a named-argument term.

Definition (Term).

1. *Constants and variables*. If $t \in$ `Const` or $t \in$ `Var` then `t` is a ***simple term***.
2. *Positional terms*. If $t \in$ `Const` and t_1, ..., t_n, $n \geq 0$, are base terms then $t(t_1 \ldots t_n)$ is a ***positional term***.

Positional terms correspond to the usual terms and atomic formulas of classical first-order logic [Enderton01, Mendelson97].

3. *Terms with named arguments*. A ***term with named arguments*** is of the form $t(s_1->v_1 \ldots s_n->v_n)$, where $n \geq 0$, $t \in Const$ and v_1, ..., v_n are base terms and s_1, ..., s_n are pairwise distinct symbols from the set ArgNames.

   The constant t here represents a predicate or a function; s_1, ..., s_n represent argument names; and v_1, ..., v_n represent argument values. The argument names, s_1, ..., s_n, are required to be pairwise distinct. Terms with named arguments are like positional terms except that the arguments are named and their order is immaterial. Note that a term of the form $f()$ is, trivially, both a positional term and a term with named arguments.

   Terms with named arguments are introduced to support exchange of languages that permit argument positions of predicates and functions to be named (in which case the order of the arguments does not matter).

4. *List terms*. There are two kinds of list terms: *open* and *closed*.
   - A ***closed list*** has the form $List(t_1 \ldots t_m)$, where $m \geq 0$ and t_1, ..., t_m are terms.
   - An ***open list*** (or a list with a tail) has the form $OpenList(t_1 \ldots t_m\ t)$, where $m > 0$ and t_1, ..., t_m, t are terms. Open lists are usually written using the following: $List(t_1 \ldots t_m \mid t)$.

     The last argument, t, represents the tail of the list and so it is normally a list as well. However, the syntax does not restrict t in any way: it could be an integer, a variable, another list, or, in fact, any term. An example is $List(1\ 2 \mid 3)$. This is not an ordinary list, where the last argument, 3, would represent the tail of a list (and thus would also be a list, which 3 is not). Such general open lists correspond to Lisp's dotted lists [Steele90]. Note that they can be the result of instantiating an open list with a variable in the tail, hence are hard to avoid. For instance, $List(1\ 2 \mid 3)$ is $List(1\ 2 \mid ?X)$, where the variable $?X$ is replaced with 3.

   A closed list of the form $List()$ (i.e., a list in which $m=0$, corresponding to Lisp's nil) is called the ***empty list***.

5. *Equality terms*. $t = s$ is an ***equality term***, if t and s are base terms.
6. *Class membership terms* (or just *membership terms*). $t\#s$ is a ***membership term*** if t and s are base terms.
7. *Subclass terms*. $t\#\#s$ is a ***subclass term*** if t and s are base terms.
8. *Frame terms*. $t[p_1->v_1 \ldots p_n->v_n]$ is a ***frame term*** (or simply a ***frame***) if t, p_1, ..., p_n, v_1, ..., v_n, $n \geq 0$, are base terms.

Membership, subclass, and frame terms are used to describe objects and class hierarchies.

9. *Externally defined terms.* If `t` is a positional or a named-argument term then `External(t)` is an ***externally defined term***.

   External terms are used for representing built-in functions and predicates as well as "procedurally attached" terms or predicates, which might exist in various rule-based systems, but are not specified by RIF. ☐

Observe that the argument names of frame terms, p_1, ..., p_n, are base terms and so, as a special case, can be variables. In contrast, terms with named arguments can use only the symbols from `ArgNames` to represent their argument names. They cannot be constants from `Const` or variables from `Var`. The reason for not allowing variables for those is to control the complexity of unification, which is used by several inference mechanisms of first-order logic.

Example 2 (Terms)

a. Positional term: `"http://example.com/ex1"^^rif:iri(1 "http://example.com/ex2"^^rif:iri(?X 5) "abc")`

b. Term with named arguments: `"http://example.com/Person"^^rif:iri(id->"http://example.com/John"^^rif:iri "age"^^rif:local->?X "spouse"^^rif:local->?Y)`

c. Frame term: `"http://example.com/John"^^rif:iri["age"^^rif:local->?X "spouse"^^rif:local->?Y]`

d. Lists

- Empty list: `List()`

- Closed list with variable inside: `List("a"^^xs:string ?Y "c"^^xs:string)`

- Open list with variables: `List("a"^^xs:string ?Y "c"^^xs:string | ?Z)`

- Equality term with lists inside: `List(Head | Tail) = List("a"^^xs:string ?Y "c"^^xs:string)`

- Nested list: `List("a"^^xs:string List(?X "b"^^xs:string) "c"^^xs:string)`

e. Classification terms

- Membership: `?X # ?Y`

- Subclass: `?X ## "http://example.com/ex1"^^rif:iri(?Y)`

- Membership: `"http://example.com/John"^^rif:iri # "http://example.com/Person"^^rif:iri`

- Subclass: `"http://example.com/Student"^^rif:iri ## "http://example.com/Person"^^rif:iri`

f. External term: `External(pred:numeric-greater-than(?diffdays 10)))`

2.3 Formulas

RIF-BLD distinguishes certain subsets of the set `Const` of symbols, including subsets of *predicate symbols* and *function symbols*. Section Well-formed Formulas gives more details, but we do not need those details yet.

Definition (Atomic Formula). Any term (positional or with named arguments) of the form `p(...)`, where `p` is a predicate symbol, is also an ***atomic formula***. Equality, membership, subclass, and frame terms are also atomic formulas. An externally defined term of the form `External(φ)`, where `φ` is an atomic formula, is also an atomic formula, called an ***externally defined*** atomic formula. □

Note that simple terms (constants and variables) are *not* formulas.

More general formulas are constructed from atomic formulas with the help of logical connectives.

Definition (Formula). A ***formula*** can have several different forms and is defined as follows:

1. *Atomic*: If `φ` is an atomic formula then it is also a formula.
2. *Condition formula*: A ***condition formula*** is either an atomic formula or a formula that has one of the following forms:
    - *Conjunction*: If φ_1, ..., φ_n, $n \geq 0$, are condition formulas then so is $\mathtt{And}(\varphi_1 \ldots \varphi_n)$, called a *conjunctive* formula. As a special case, `And()` is allowed and is treated as a tautology, i.e., a formula that is always true.
    - *Disjunction*: If φ_1, ..., φ_n, $n \geq 0$, are condition formulas then so is $\mathtt{Or}(\varphi_1 \ldots \varphi_n)$, called a *disjunctive* formula. As a special case, `Or()` is permitted and is treated as a contradiction, i.e., a formula that is always false.

- *Existentials*: If φ is a condition formula and ?V1, ..., ?Vn, n>0, are distinct variables then `Exists ?V1 ... ?Vn(φ)` is an *existential* formula.

Condition formulas are intended to be used inside the premises of rules. Next we define the notions of rule implications, universal rules, universal facts, groups (i.e., sets of rules and facts), and documents.

3. *Rule implication*: `φ :- ψ` is a formula, called *rule implication*, if:
   - φ is an atomic formula or a *conjunction* of atomic formulas,
   - ψ is a condition formula, and
   - none of the atomic formulas in φ is an externally defined term (i.e., a term of the form `External(...)`). Note: external terms *can* occur in the *arguments* of atomic formulas in the rule conclusion. For instance, `p(func:numeric-add(?X,"2"^^xs:integer)) :- q(?X).`
4. *Universal rule*: If φ is a rule implication and ?V1, ..., ?Vn, n>0, are distinct variables then `Forall ?V1 ... ?Vn(φ)` is a formula, called a *universal rule*. It is required that all the *free* variables in φ occur among the variables `?V1 ... ?Vn` in the quantification part. An occurrence of a variable `?v` is *free* in φ if it is not inside a subformula of φ of the form `Exists ?v (ψ)` and ψ is a formula. Universal rules will also be referred to as ***RIF-BLD rules***.
5. *Universal fact*: If φ is an atomic formula and ?V1, ..., ?Vn, n>0, are distinct variables then `Forall ?V1 ... ?Vn(φ)` is a formula, called a *universal fact*, provided that all the free variables in φ occur among the variables `?V1 ... ?Vn`.

   Universal facts are often considered to be rules without premises.

6. *Group*: If φ1, ..., φn are RIF-BLD rules, universal facts, variable-free rule implications, variable-free atomic formulas, *or* group formulas then `Group(φ1 ... φn)` is a *group formula*. As a special case, the empty group formula, `Group()`, is allowed and is treated as a tautology, i.e., a formula that is always true.

   Non-empty group formulas are used to represent sets of rules and facts. Note that some of the φi's can be group formulas themselves, which means that groups can be nested.

7. *Document*: An expression of the form `Document(`*directive1 ... directiven* `Γ)` is a *RIF-BLD document formula* (or simply a *document formula*), if
   - Γ is an optional group formula; it is called the group formula *associated* with the document.
   - *directive1*, ..., *directiven* is an optional sequence of *directives*. A directive can be a *base directive*, a *prefix directive* or an *import directive*.

- A ***base directive*** has the form `Base(<iri>)`, where `iri` is a Unicode string in the form of an absolute IRI [RFC-3987].

  The `Base` directive defines a syntactic shortcut for expanding relative IRIs into full IRIs, as described in Section Constants, Symbol Spaces, and Datatypes of [RIF-DTB].

- A ***prefix directive*** has the form `Prefix(p <v>)`, where `p` is an alphanumeric string that serves as the prefix name and `v` is an expansion for `p` -- a Unicode sequence of characters that forms an IRI. (An alphanumeric string is a sequence of ASCII characters, where each character is a letter, a digit, or an underscore "_", and the first character is a letter.)

  Like the `Base` directive, the `Prefix` directives define shorthands to allow more concise representation of constants that come from the symbol space `rif:iri` (we will call such constants `rif:iri` ***constants***). This mechanism is explained in [RIF-DTB], Section Constants, Symbol Spaces, and Datatypes.

- An ***import directive*** can have one of these two forms: `Import(<loc>)` or `Import(<loc> <p>)`. Here `loc` is a Unicode sequence of characters that forms an IRI and `p` is another Unicode sequence of characters. The constant `loc` represents the location of another document to be imported; it is called the ***locator*** of the imported document. The argument `p` is called the *profile of import*; it has the form of a Unicode character sequence in the form of an IRI -- see [RIF-RDF+OWL].

  Section Direct Specification of RIF-BLD Semantics of this document defines the semantics for the directive `Import(<loc>)` only. The two-argument directive, `Import(<loc> <p>)`, is intended for importing non-RIF-BLD documents, such as rules from other RIF dialects, RDF data, or OWL ontologies. The profile, `p`, indicates what kind of entity is being imported and under what semantics (for instance, the various RDF entailment regimes have different profiles). The semantics of `Import(<loc> <p>)` (for various `p`) are expected to be given by other specifications on a case-by-case basis. For instance, [RIF-RDF+OWL] defines the semantics for the profiles that are recommended for importing RDF and OWL.

Note that although `Base`, `Prefix`, and `Import` all use symbols of the form <iri> to indicate the connection of these symbols to IRIs, these symbols are *not* `rif:iri` constants, as semantically they are interpreted in a way that is quite different from constants.

A document formula can contain at most one `Base` directive. The `Base` directive, if present, must be first, followed by any number of `Prefix` directives, followed by any number of `Import` directives.

In the definition of a formula, the component formulas φ, φ_i, ψ_i, and Γ are said to be ***subformulas*** of the respective formulas (condition, rule, group, etc.) that are built using these components. □

2.4 RIF-BLD Annotations in the Presentation Syntax

RIF-BLD allows every term and formula (including terms and formulas that occur inside other terms and formulas) to be optionally preceded by *one* ***annotation*** of the form `(* id φ *)`, where `id` is a `rif:iri` constant and φ is a frame formula or a conjunction of frame formulas. Both items inside the annotation are optional. The `id` part represents the identifier of the term or formula to which the annotation is attached and φ is the metadata part of the annotation. RIF-BLD does not impose any restrictions on φ apart from what is stated above. This means that it may include variables, function symbols, constants from the symbol space `rif:local` (often referred to as ***local*** or `rif:local` ***constants***), and so on.

Document formulas with and without annotations will be referred to as ***RIF-BLD documents***.

The following convention is used to avoid a syntactic ambiguity with respect to annotations. The annotation scoping convention associates each annotation to the largest term or formula it precedes. For instance, in `(* id φ *) t[w -> v]` the metadata annotation could be attributed to the term `t` or to the entire frame `t[w -> v]`. The convention specifies that the above annotation is considered to be syntactically attached to the entire frame. Yet, since φ can be a conjunction, some conjuncts can be used to provide metadata targeted to the object part, `t`, of the frame. For instance, `(* And(_foo[meta_for_frame->"this is an annotation for the entire frame"] _bar[meta_for_object->"this is an annotation for t" meta_for_property->"this is an annotation for w"]) *) t[w -> v].`

We suggest to use Dublin Core, RDFS, and OWL properties for metadata, along the lines of Section 7.1 of [OWL-Reference]-- specifically `owl:versionInfo`,

`rdfs:label`, `rdfs:comment`, `rdfs:seeAlso`, `rdfs:isDefinedBy`, `dc:creator`, `dc:description`, `dc:date`, and `foaf:maker`.

2.5 Well-formed Formulas

Not all formulas and thus not all documents are well-formed in RIF-BLD: it is required that no constant appear in more than one context. What this means precisely is explained below. Informally, this means that each constant symbol in RIF-BLD can be either an individual, a plain function, a plain predicate, an externally defined function, or an externally defined predicate. However, symbols can be *polyadic*: the same function or predicate symbol (normal or external) can occur with different numbers of arguments in different places. Note that polyadic symbols could be replaced by non-polyadic symbols with the arity information encoded in the function or predicate names. For instance, the polyadic terms `p(?X)` and `p(?X,?Y)` could be represented as `p_1(?X)` and `p_2(?X,?Y)`, respectively.

The set of all constant symbols, `Const`, is partitioned into the following subsets:

- A subset of individuals.

  The symbols in `Const` that belong to the symbol spaces of Datatypes are required to be individuals.

- A subset of plain (i.e., non-external) function symbols.
- A subset for external function symbols.
- A subset of plain predicate symbols.
- A subset for external predicate symbols.

The above subsets do not differentiate between positional and named argument symbols. Also, as seen from the following definitions, these subsets are not specified explicitly but, rather, are inferred from the occurrences of the symbols.

Definition (Context of a symbol). The ***context of an occurrence*** of a symbol, `s∈Const`, in a formula, `φ`, is determined as follows:

- If `s` occurs as a predicate of the form `s(...)` (positional or named-argument) in an atomic subformula of `φ` then `s` occurs in the *context of a (plain) predicate symbol*.
- If `s` occurs as a function symbol in a non-subformula term of the form `s(...)` then `s` occurs in the *context of a (plain) function symbol*.
- If `s` occurs as a predicate in an atomic subformula `External(s(...))` then `s` occurs in the *context of an external predicate symbol*.
- If `s` occurs as a function in a non-subformula term `External(s(...))` then `s` occurs in the *context of an external function symbol*.

- If `s` occurs in any other context (in a frame: `s[...]`, `...[s->...]`, or `...[...->s]`; or in a positional/named-argument term: `p(...s...)`, `q(...->s...)`), it is said to occur as an *individual*. ☐

Definition (Imported document). Let `Δ` be a document formula and `Import(`*loc*`)` be one of its import directives, where *loc* is a locator of another document formula, `Δ'`. We say that `Δ'` is ***directly imported*** into `Δ`.

A document formula `Δ'` is said to be ***imported*** into `Δ` if it is either directly imported into `Δ` or it is imported (directly or not) into some other formula that is directly imported into `Δ`. ☐

The above definition deals only with one-argument import directives, since only such directives can be used to import other RIF-BLD documents. Two-argument import directives are provided to enable import of other types of documents, and their semantics are supposed to be covered by other specifications, such as [RIF-RDF+OWL].

Definition (Well-formed formula). A formula `φ` is ***well-formed*** iff:

- every constant symbol (whether coming from the symbol space `rif:local` or not) mentioned in `φ` occurs in exactly one context.
- if `φ` is a document formula and `Δ'`$_1$, ..., `Δ'`$_k$ are all of its imported documents, then every non-`rif:local` constant symbol mentioned in `φ` or any of the imported `Δ'`$_i$s must occur in exactly one context (in all of the `Δ'`$_i$s).
- whenever a formula contains a term or a subformula of the form `External(t)`, `t` must be an instantiation of a schema in the coherent set of external schemas (Section Schemas for Externally Defined Terms of [RIF-DTB]) associated with the language of RIF-BLD.
- if `t` is an instantiation of a schema in the coherent set of external schemas associated with the language then `t` can occur only as `External(t)`, i.e., as an external term or atomic formula. ☐

Definition (Language of RIF-BLD). The ***language of RIF-BLD*** consists of the set of all well-formed formulas and is determined by:

- the alphabet of the language and
- a set of coherent external schemas, which determine the available built-ins and other externally defined predicates and functions. ☐

2.6 EBNF Grammar for the Presentation Syntax of RIF-BLD (Informative)

Until now, we have been using mathematical English to specify the syntax of RIF-BLD. Tool developers, however, may prefer EBNF notation, which provides a more succinct view of the syntax. Several points should be kept in mind regarding this notation.

- The syntax of first-order logic is not context-free, so EBNF cannot capture the syntax of RIF-BLD precisely. For instance, it cannot capture some well-formedness conditions, such as the requirement that external terms must be instances of external schemas. As a result, the EBNF grammar defines a strict *superset* of RIF-BLD: not all formulas that are derivable using the EBNF grammar are well-formed formulas in RIF-BLD.
- The EBNF grammar does not address all details of how constants (defined in [RIF-DTB]) and variables are represented, and it is not sufficiently precise about the delimiters and escape symbols. White space is informally used as a delimiter, and is implied in productions that use Kleene star. For instance, `TERM*` is to be understood as `TERM TERM ... TERM`, where each space abstracts from one or more blanks, tabs, newlines, etc. This is so because RIF's presentation syntax is a tool for specifying the semantics and for illustration of the main RIF concepts through examples. It is *not* intended as a concrete syntax for a rule language. RIF defines a concrete syntax only for *exchanging* rules, and that syntax is XML-based, obtained as a refinement and serialization of the presentation syntax.
- For all the above reasons, the EBNF grammar is *not normative*. Recall from the opening paragraph, however, that the RIF-BLD presentation syntax as specified in mathematical English is normative.

The EBNF for the RIF-BLD presentation syntax is given as follows, showing the entire (top-down) context of its three parts for rules, conditions, and annotations.

Rule Language:

```
Document       ::= IRIMETA? 'Document' '(' Base? Prefix* Import* Group? '
Base           ::= 'Base' '(' ANGLEBRACKIRI ')'
Prefix         ::= 'Prefix' '(' NCName ANGLEBRACKIRI ')'
Import         ::= IRIMETA? 'Import' '(' LOCATOR PROFILE? ')'
Group          ::= IRIMETA? 'Group' '(' (RULE | Group)* ')'
RULE           ::= (IRIMETA? 'Forall' Var+ '(' CLAUSE ')') | CLAUSE
CLAUSE         ::= Implies | ATOMIC
Implies        ::= IRIMETA? (ATOMIC | 'And' '(' ATOMIC* ')') ':-' FORMULA
LOCATOR        ::= ANGLEBRACKIRI
PROFILE        ::= ANGLEBRACKIRI
```

Condition Language:

```
FORMULA        ::= IRIMETA? 'And' '(' FORMULA* ')' |
                   IRIMETA? 'Or' '(' FORMULA* ')' |
                   IRIMETA? 'Exists' Var+ '(' FORMULA ')' |
                   ATOMIC |
                   IRIMETA? 'External' '(' Atom ')'
ATOMIC         ::= IRIMETA? (Atom | Equal | Member | Subclass | Frame)
Atom           ::= UNITERM
UNITERM        ::= Const '(' (TERM* | (Name '->' TERM)*) ')'
Equal          ::= TERM '=' TERM
Member         ::= TERM '#' TERM
Subclass       ::= TERM '##' TERM
Frame          ::= TERM '[' (TERM '->' TERM)* ']'
TERM           ::= IRIMETA? (Const | Var | Expr | List | 'External' '(' E
Expr           ::= UNITERM
List           ::= 'List' '(' TERM* ')' | 'List' '(' TERM+ '|' TERM ')'
Const          ::= '"' UNICODESTRING '"^^' SYMSPACE | CONSTSHORT
Var            ::= '?' Name
Name           ::= NCName | '"' UNICODESTRING '"'
SYMSPACE       ::= ANGLEBRACKIRI | CURIE
```

Annotations:

```
IRIMETA        ::= '(*' IRICONST? (Frame | 'And' '(' Frame* ')')? '*)'
```

The following subsections explain and exemplify these parts, starting with the basic language of positive conditions.

2.6.1 EBNF for the Condition Language

The Condition Language represents formulas that can be used in the premises of RIF-BLD rules (also called rule bodies). The EBNF grammar for a superset of the RIF-BLD condition language is shown in the above conditions part.

The production rule for the non-terminal `FORMULA` represents *RIF condition formulas* (defined earlier). The connectives `And` and `Or` define conjunctions and disjunctions of conditions, respectively. `Exists` introduces existentially quantified variables. Here `Var+` stands for the list of variables that are free in `FORMULA`. A RIF-BLD `FORMULA` can also be an `ATOMIC` term, i.e. an `Atom`, `External Atom`, `Equal`, `Member`, `Subclass`, or `Frame`. A `TERM` can be a constant, variable, `Expr`, `List`, or `External Expr`.

The RIF-BLD presentation syntax does not commit to any particular vocabulary and permits arbitrary Unicode strings in constant symbols, argument names, and variables. Constant symbols can have this form: `"UNICODESTRING"^^SYMSPACE`,

where SYMSPACE is an ANGLEBRACKIRI or CURIE that represents the identifier of the symbol space of the constant. UNICODESTRING, ANGLEBRACKIRI, and CURIE are defined in Section Shortcuts for Constants in RIF's Presentation Syntax of [RIF-DTB]. Constant symbols can also have several shortcut forms, which are represented by the non-terminal CONSTSHORT. These shortcuts are also defined in the same section of [RIF-DTB]. One of them is the CURIE shortcut, which is extensively used in the examples in this document. Names are Unicode character sequences. Variables are composed of UNICODESTRING symbols prefixed with a ?-sign.

Equality, membership, and subclass terms are self-explanatory. An Atom and Expr (expression) can either be positional or have named arguments. A frame term is a term composed of an object identifier and a collection of attribute-value pairs. The term External(Atom) is a call to an externally defined predicate. Likewise, External(Expr) is a call to an externally defined function.

Example 3 (RIF-BLD conditions).

This example shows conditions that are composed of atoms, expressions, equalities with lists, frames, and existentials. In frame formulas, variables are shown in the positions of object identifiers, object properties, and property values. For brevity, we use the shortcut CURIE notation prefix:suffix for constant symbols defined in [RIF-DTB]. This is understood as a shorthand for an IRI obtained by concatenation of the prefix definition and suffix. Thus, if bks is a prefix that expands into http://example.com/books# then bks:LeRif is an abbreviation for "http://example.com/books#LeRif"^^rif:iri.

```
Prefix(bks  <http://example.com/books#>)
Prefix(auth <http://example.com/authors#>)
Prefix(cpt  <http://example.com/concepts#>)
```

Positional terms:

```
cpt:book(auth:rifwg bks:LeRif)
Exists ?X (cpt:book(?X bks:LeRif))
```

Terms with named arguments:

```
cpt:book(cpt:author->auth:rifwg  cpt:title->bks:LeRif)
Exists ?X (cpt:book(cpt:author->?X cpt:title->bks:LeRif))
```

Equalities with list terms:

```
?L = List(?X ?Y ?X)
List(?Head | ?Tail) = List("a"^^rif:local ?Y "c"^^rif:local)
```

Frames:

```
bks:wd1[cpt:author->auth:rifwg cpt:title->bks:LeRif]
Exists ?X (bks:wd2[cpt:author->?X  cpt:title->bks:LeRif])
Exists ?X (And (bks:wd2#cpt:book  bks:wd2[cpt:author->?X  cpt:title->bks:
Exists ?I ?X (?I[cpt:author->?X  cpt:title->bks:LeRif])
Exists ?I ?X (And (?I#cpt:book ?I[cpt:author->?X  cpt:title->bks:LeRif]))
Exists ?S (bks:wd2[cpt:author->auth:rifwg ?S->bks:LeRif])
Exists ?X ?S (bks:wd2[cpt:author->?X ?S->bks:LeRif])
Exists ?I ?X ?S (And (?I#cpt:book  ?I[author->?X ?S->bks:LeRif]))
```

2.6.2 EBNF for the Rule Language

The presentation syntax for RIF-BLD rules is based on the syntax in Section EBNF for RIF-BLD Condition Language with the productions shown in the above rules part.

A RIF-BLD `Document` consists of an optional `Base`, followed by any number of `Prefix`es, followed by any number of `Import`s, followed by an optional `Group`. `Base` and `Prefix` serve as shortcut mechanisms for IRIs. `IRI` has the form of an internationalized resource identifier as defined by [RFC-3987]. An `Import` indicates the location of a document to be imported and an optional profile. A RIF-BLD `Group` is a collection of any number of `RULE` elements along with any number of nested `Group`s.

Rules are generated using `CLAUSE` elements. The `RULE` production has two alternatives:

- In the first, a `CLAUSE` is in the scope of the `Forall` quantifier. In that case, all variables mentioned in `CLAUSE` are required to also appear among the variables in the `Var+` sequence.
- In the second alternative, `CLAUSE` appears on its own. In that case, `CLAUSE` cannot have variables.

`Var`, `ATOMIC`, and `FORMULA` were defined as part of the syntax for positive conditions in Section EBNF for RIF-BLD Condition Language. In the `CLAUSE` production, an `ATOMIC` is what is usually called a *fact*. An `Implies` *rule* can have an `ATOMIC` or a conjunction of `ATOMIC` elements as its conclusion; it has a `FORMULA` as its premise. Note that, by the definition of formulas, externally defined atoms (i.e., formulas of the form `External(Atom)`) are not allowed in the conclusion part of a rule (`ATOMIC` does not expand to `External`).

Example 4 (RIF-BLD rules).

This example shows a business rule borrowed from the document RIF Use Cases and Requirements:

- *If an item is perishable and it is delivered to John more than 10 days after the scheduled delivery date then the item will be rejected by him.*

As before, for better readability we use the compact URI notation, the angle-bracket notation, and the `Base` directive defined in [RIF-DTB], Section Constants, Symbol Spaces, and Datatypes. Again, directives are assumed in the preamble to the document. Then, two versions of the main part of the document are given.

```
Base(<http://example.com/people#>)
Prefix(cpt  <http://example.com/concepts#>)
Prefix(func <http://www.w3.org/2007/rif-builtin-function#>)
Prefix(pred <http://www.w3.org/2007/rif-builtin-predicate#>)
```

a. Universal form:

```
Forall ?item ?deliverydate ?scheduledate ?diffduration ?diffdays (
     cpt:reject(<John> ?item) :-
         And(cpt:perishable(?item)
             cpt:delivered(?item ?deliverydate <John>)
             cpt:scheduled(?item ?scheduledate)
             ?diffduration = External(func:subtract-dateTimes(?deliveryd
             ?diffdays = External(func:days-from-duration(?diffduration)
             External(pred:numeric-greater-than(?diffdays 10)))
)
```

b. Universal-existential form:

```
Forall ?item (
     cpt:reject(<John> ?item ) :-
         Exists ?deliverydate ?scheduledate ?diffduration ?diffdays (
              And(cpt:perishable(?item)
                  cpt:delivered(?item ?deliverydate <John>)
                  cpt:scheduled(?item ?scheduledate)
                  ?diffduration = External(func:subtract-dateTimes(?deli
                  ?diffdays = External(func:days-from-duration(?diffdura
                  External(pred:numeric-greater-than(?diffdays 10)))
         )
)
```

2.6.3 EBNF for Annotations

The EBNF grammar production for RIF-BLD annotations is shown in the above annotations part.

As explained in Section RIF-BLD Annotations in the Presentation Syntax, each RIF-BLD formula and term can be preceded by one optional annotation, `IRIMETA`, for identification and metadata. `IRIMETA` is represented using `(*...*)`-brackets that contain an optional `rif:iri` constant, `IRICONST`, as identifier followed by an optional `Frame` or conjunction of `Frame`s as metadata.

An `IRICONST` is `rif:iri` constant, again permitting the shortcut forms defined in [RIF-DTB]. One such specialization is `'"' IRI '"^^' 'rif:iri'` from the `Const` production, where `IRI` is a sequence of Unicode characters that forms an internationalized resource identifier as defined by [RFC-3987].

Example 5 (A RIF-BLD document containing an annotated group).

This example shows a complete document containing a group formula that consists of two RIF-BLD rules. The first of these rules is copied from Example 4a. The group is annotated with an IRI identifier and metadata represented using Dublin Core vocabulary.

```
Document(
  Base(<http://example.com/people#>)
  Prefix(cpt  <http://example.com/concepts#>)
  Prefix(dc   <http://purl.org/dc/terms/>)
  Prefix(func <http://www.w3.org/2007/rif-builtin-function#>)
  Prefix(pred <http://www.w3.org/2007/rif-builtin-predicate#>)
  Prefix(xs   <http://www.w3.org/2001/XMLSchema#>)

  (* "http://sample.org"^^rif:iri _pd[dc:publisher -> "http://www.w3.org/"^
                                     dc:date -> "2008-04-04"^^xs:date] *)
  Group
  (
    Forall ?item ?deliverydate ?scheduledate ?diffduration ?diffdays (
        cpt:reject(<John> ?item) :-
            And(cpt:perishable(?item)
                cpt:delivered(?item ?deliverydate <John>)
                cpt:scheduled(?item ?scheduledate)
                ?diffduration = External(func:subtract-dateTimes(?deliveryd
                ?diffdays = External(func:days-from-duration(?diffduration)
                External(pred:numeric-greater-than(?diffdays 10)))
  )
```

```
        Forall ?item (
            cpt:reject(<Fred> ?item) :- cpt:unsolicited(?item)
        )
    )
)
```

3 Direct Specification of RIF-BLD Semantics

This normative section specifies the semantics of RIF-BLD directly, without relying on [RIF-FLD].

Recall that the presentation syntax of RIF-BLD allows shorthand notation, which is specified via the `Prefix` and `Base` directives, and various shortcuts for integers, strings, and `rif:local` symbols. The semantics, below, is described using the full syntax, i.e., we assume that all shortcuts have already been expanded as defined in [RIF-DTB], Section Constants, Symbol Spaces, and Datatypes.

3.1 Truth Values

The set ***TV*** of truth values in RIF-BLD consists of two values, **t** and **f**.

3.2 Semantic Structures

The key concept in a model-theoretic semantics for a logic language is the notion of a *semantic structure* [Enderton01, Mendelson97]. The definition is slightly more general than what is strictly necessary for RIF-BLD alone. This lays the groundwork for extensions to RIF-BLD and makes the connection with the semantics of the RIF framework for logic-based dialects [RIF-FLD] more obvious.

Definition (Semantic structure). A ***semantic structure***, ***I***, is a tuple of the form <***TV***, ***DTS***, ***D***, $\boldsymbol{D}_{ind}$, $\boldsymbol{D}_{func}$, $\boldsymbol{I}_C$, $\boldsymbol{I}_V$, $\boldsymbol{I}_F$, $\boldsymbol{I}_{NF}$, $\boldsymbol{I}_{list}$, $\boldsymbol{I}_{tail}$, $\boldsymbol{I}_{frame}$, $\boldsymbol{I}_{sub}$, $\boldsymbol{I}_{isa}$, $\boldsymbol{I}_{=}$, $\boldsymbol{I}_{external}$, $\boldsymbol{I}_{truth}$>. Here ***D*** is a non-empty set of elements called the ***domain*** of ***I***, and $\boldsymbol{D}_{ind}$, $\boldsymbol{D}_{func}$ are nonempty subsets of ***D***. $\boldsymbol{D}_{ind}$ is used to interpret the elements of `Const` that occur as individuals and $\boldsymbol{D}_{func}$ is used to interpret the elements of `Const` that occur in the context of function symbols. As before, `Const` denotes the set of all constant symbols and `Var` the set of all variable symbols. ***DTS*** denotes a set of identifiers for datatypes (please refer to Section Datatypes of [RIF-DTB] for the semantics of datatypes).

The other components of ***I*** are *total* mappings defined as follows:

1. $\boldsymbol{I}_C$ maps `Const` to ***D***.

This mapping interprets constant symbols. In addition:

- If a constant, $c \in$ `Const`, is an *individual* then it is required that $I_C(c) \in D_{ind}$.
- If $c \in$ `Const`, is a *function symbol* (positional or with named arguments) then it is required that $I_C(c) \in D_{func}$.

2. I_V maps `Var` to D_{ind}.

   This mapping interprets variable symbols.

3. I_F maps D to total functions $D^*_{ind} \rightarrow D$ (here D^*_{ind} is a set of all finite sequences over the domain D_{ind}).

   This mapping interprets positional terms. In addition:

   - If $d \in D_{func}$ then $I_F(d)$ must be a function $D^*_{ind} \rightarrow D_{ind}$.
   - This means that when a function symbol is applied to arguments that are individual objects then the result is also an individual object.

4. I_{NF} maps D to the set of total functions of the form `SetOfFiniteSets`(`ArgNames` × D_{ind}) → D.

   This mapping interprets function symbols with named arguments. In addition:

   - If $d \in D_{func}$ then $I_{NF}(d)$ must be a function `SetOfFiniteSets`(`ArgNames` × D_{ind}) → D_{ind}.
   - This is analogous to the interpretation of positional terms with two differences:
     - Each pair <`s`, `v`> ∈ `ArgNames` × D_{ind} represents an argument/value pair instead of just a value in the case of a positional term.
     - The arguments of a term with named arguments constitute a finite set of argument/value pairs rather than a finite ordered sequence of simple elements. So, the order of the arguments does not matter.

5. I_{list} and I_{tail} are used to interpret lists. They are mappings of the following form:
   - $I_{list} : D_{ind}^* \rightarrow D_{ind}$
   - $I_{tail} : D_{ind}^+ \times D_{ind} \rightarrow D_{ind}$

   In addition, these mappings are required to satisfy the following conditions:

   - The function I_{list} is injective (one-to-one).
   - The set $I_{list}(D_{ind}^*)$, henceforth denoted D_{list}, is disjoint from the value spaces of all data types in ***DTS***.
   - $I_{tail}(a_1, ..., a_k, I_{list}(a_{k+1}, ..., a_{k+m})) = I_{list}(a_1, ..., a_k, a_{k+1}, ..., a_{k+m})$.

Note that the last condition above restricts I_{tail} only when its last argument is in D_{list}. If the last argument of I_{tail} is not in D_{list}, then the list is a general open one and there are no restrictions on the value of I_{tail} except that it must be in D_{ind}.

6. I_{frame} maps D_{ind} to total functions of the form `SetOfFiniteBags`(D_{ind} × D_{ind}) → D.

   This mapping interprets frame terms. An argument, `d` ∈ D_{ind}, to I_{frame} represents an object and the finite bag {<`a1`,`v1`>, ..., <`ak`,`vk`>} represents a bag of attribute-value pairs for `d`. We will see shortly how I_{frame} is used to determine the truth valuation of frame terms.

   Bags (multi-sets) are used here because the order of the attribute/value pairs in a frame is immaterial and pairs may repeat. Such repetitions arise naturally when variables are instantiated with constants. For instance, `o[?A->?B ?C->?D]` becomes `o[a->b a->b]` if variables `?A` and `?C` are instantiated with the symbol `a` while `?B` and `?D` are instantiated with `b`. (We shall see later that `o[a->b a->b]` is equivalent to `o[a->b]`.)

7. I_{sub} gives meaning to the subclass relationship. It is a mapping of the form D_{ind} × D_{ind} → D.

   I_{sub} will be further restricted in Section Interpretation of Formulas to ensure that the operator `##` is transitive, i.e., that `c1 ## c2` and `c2 ## c3` imply `c1 ## c3`.

8. I_{isa} gives meaning to class membership. It is a mapping of the form D_{ind} × D_{ind} → D.

   I_{isa} will be further restricted in Section Interpretation of Formulas to ensure that the relationships `#` and `##` have the usual property that all members of a subclass are also members of the superclass, i.e., that `o # cl` and `cl ## scl` imply `o # scl`.

9. $I_{=}$ is a mapping of the form D_{ind} × D_{ind} → D.

   It gives meaning to the equality operator.

10. I_{truth} is a mapping of the form D → TV.

    It is used to define truth valuation for formulas.

11. $I_{external}$ is a mapping from the coherent set of schemas for externally defined functions to total functions D^* → D. For each external schema σ = `(?`X_1 `...` `?`X_n`; τ)` in the coherent set of external schemas associated with the language, $I_{external}(\sigma)$ is a function of the form D^n → D.

For every external schema, σ, associated with the language, $I_{external}(\sigma)$ is assumed to be specified externally in some document (hence the name *external schema*). In particular, if σ is a schema of a RIF built-in predicate or function, $I_{external}(\sigma)$ is specified in [RIF-DTB] so that:

- If σ is a schema of a built-in function then $I_{external}(\sigma)$ must be the function defined in [RIF-DTB].
- If σ is a schema of a built-in predicate then $I_{truth} \circ (I_{external}(\sigma))$ (the composition of I_{truth} and $I_{external}(\sigma)$, a truth-valued function) must be as specified in [RIF-DTB].

We also define the following mapping from terms to $\boldsymbol{D}$, which we denote using the same symbol I as the one used for semantic structures. This overloading is convenient and creates no ambiguity.

- I(`k`) = I_C(`k`), if `k` is a symbol in `Const`
- I(`?v`) = I_V(`?v`), if `?v` is a variable in `Var`
- I(`f(`t_1 `...` t_n`)`) = $I_F(I(f))(I(t_1),...,I(t_n))$
- I(`f(`s_1`->`v_1 `...` s_n`->`v_n`)`) = $I_{NF}(I(f))(\{<s_1,I(v_1)>,...,<s_n,I(v_n)>\})$

  Here we use {...} to denote a set of argument/value pairs.

- For list terms, the mapping is defined as follows:
  - I(`List()`) = $I_{list}(<>)$.

    Here <> denotes an empty list of elements of $\boldsymbol{D}_{ind}$. (Note that the domain of I_{list} is $\boldsymbol{D}_{ind}^*$, so $\boldsymbol{D}_{ind}^0$ is an empty list of elements of $\boldsymbol{D}_{ind}$.)

  - I(`List(`t_1 ... t_n`)`) = $I_{list}(I(t_1), ..., I(t_n))$, if `n>0`.
  - I(`List(`t_1 ... t_n `|` t`)`) = $I_{tail}(I(t_1), ..., I(t_n), I(t))$, if `n>0`.
- I(`o[`a_1`->`v_1 `...` a_k`->`v_k`]`) = $I_{frame}(I(o))(\{<I(a_1),I(v_1)>, ..., <I(a_n),I(v_n)>\})$

  Here {...} denotes a bag of attribute/value pairs. Jumping ahead, we note that duplicate elements in such a bag do not affect the truth value of a frame formula. Thus, for instance, `[a->b a->b]` and `o[a->b]` always have the same truth value.

- I(`c1##c2`) = $I_{sub}(I(c1), I(c2))$
- I(`o#c`) = $I_{isa}(I(o), I(c))$
- I(`x=y`) = $I_=(I(x), I(y))$
- I(`External(t)`) = $I_{external}(\sigma)(I(s_1), ..., I(s_n))$, if `t` is an instantiation of the external schema `σ = (?`X_1 `...` `?`X_n`; t)` by substitution `?`X_1`/`s_1 `...` `?`X_n`/`s_1.

  Note that, by definition, `External(t)` is well-formed only if `t` is an instantiation of an external schema. Furthermore, by the definition of

coherent sets of external schemas, `t` can be an instantiation of at most one such schema, so $\boldsymbol{I}$(`External(t)`) is well-defined.

Note that the definitions of $\boldsymbol{I}_{NF}$ and $\boldsymbol{I}$(`x=y`) imply that the terms with named arguments that differ only in the order of their arguments are mapped by $\boldsymbol{I}$ to the same element in the domain. This implies that the equalities like `t(a->1 b->2 c->3) = t(c->3 a->2 b->2)` are tautologies in RIF-BLD.

The effect of datatypes. The set ***DTS*** must include the datatypes described in Section Datatypes of [RIF-DTB].

The datatype identifiers in ***DTS*** impose the following restrictions. Given `dt` $\in$ ***DTS***, let $\boldsymbol{LS}_{dt}$ denote the lexical space of `dt`, $\boldsymbol{VS}_{dt}$ denote its value space, and $\boldsymbol{L}_{dt}$: $\boldsymbol{LS}_{dt}$ $\rightarrow$ $\boldsymbol{VS}_{dt}$ the lexical-to-value-space mapping (for the definitions of these concepts, see Section Datatypes of [RIF-DTB]). Then the following must hold:

- $\boldsymbol{VS}_{dt} \subseteq \boldsymbol{D}_{ind}$; and
- For each constant `"lit"^^dt` such that `lit` $\in$ $\boldsymbol{LS}_{dt}$, $\boldsymbol{I}_C$(`"lit"^^dt`) = $\boldsymbol{L}_{dt}$(`lit`).

That is, $\boldsymbol{I}_C$ must map the constants of a datatype `dt` in accordance with $\boldsymbol{L}_{dt}$.

RIF-BLD does not impose restrictions on $\boldsymbol{I}_C$ for constants in symbol spaces that are not datatypes included in ***DTS***. □

3.3 RIF-BLD Annotations in the Semantics

RIF-BLD annotations are stripped before the mappings that constitute RIF-BLD semantic structures are applied. Likewise, they are stripped before applying the truth valuation, $TVal_I$, defined in the next section. Thus, identifiers and metadata have no effect on the formal semantics.

Note that although identifiers and metadata associated with RIF-BLD formulas are ignored by the semantics, they can be extracted by XML tools. The frame terms used to represent RIF-BLD metadata can then be fed to other RIF-BLD rules, thus enabling reasoning about metadata. RIF-BLD does not define any particular semantics for metadata, however.

3.4 Interpretation of Non-document Formulas

This section defines how a semantic structure, ***I***, determines the truth value $TVal_I(\varphi)$ of a RIF-BLD formula, φ, where φ is any formula other than a document formula. Truth valuation of document formulas is defined in the next section.

We define a mapping, $TVal_I$, from the set of all non-document formulas to ***TV***. Note that the definition implies that $TVal_I(\varphi)$ is defined *only if* the set ***DTS*** of the datatypes of ***I*** includes all the datatypes mentioned in φ and $\boldsymbol{I}_{external}$ is defined on all externally defined functions and predicates in φ.

Definition (Truth valuation). ***Truth valuation*** for well-formed formulas in RIF-BLD is determined using the following function, denoted $TVal_I$:

1. *Positional atomic formulas*: $TVal_I(\texttt{r(}t_1\ \texttt{...}\ t_n\texttt{)}) = \boldsymbol{I}_{truth}(\boldsymbol{I}(\texttt{r(}t_1\ \texttt{...}\ t_n\texttt{)}))$
2. *Atomic formulas with named arguments*: $TVal_I(\texttt{p(}s_1\texttt{->}v_1\ \texttt{...}\ s_k\texttt{->}v_k\texttt{)}) = \boldsymbol{I}_{truth}(\boldsymbol{I}(\texttt{p(}s_1\texttt{->}v_1\ \texttt{...}\ s_k\texttt{->}v_k\texttt{)}))$.
3. *Equality*: $TVal_I(\texttt{x = y}) = \boldsymbol{I}_{truth}(\boldsymbol{I}(\texttt{x = y}))$.
   - To ensure that equality has precisely the expected properties, it is required that:
     - $\boldsymbol{I}_{truth}(\boldsymbol{I}(\texttt{x = y})) = \textbf{t}$ if $\boldsymbol{I}(\texttt{x}) = \boldsymbol{I}(\texttt{y})$ and that $\boldsymbol{I}_{truth}(\boldsymbol{I}(\texttt{x = y})) = \textbf{f}$ otherwise.
   - This is tantamount to saying that $TVal_I(\texttt{x = y}) = \textbf{t}$ if and only if $\boldsymbol{I}(\text{x}) = \boldsymbol{I}(\text{y})$.
4. *Subclass*: $TVal_I(\texttt{sc \#\# cl}) = \boldsymbol{I}_{truth}(\boldsymbol{I}(\texttt{sc \#\# cl}))$.

   To ensure that the operator `##` is transitive, i.e., `c1 ## c2` and `c2 ## c3` imply `c1 ## c3`, the following is required:

   - For all `c1`, `c2`, `c3` $\in \boldsymbol{D}$, if $TVal_I(\texttt{c1 \#\# c2}) = TVal_I(\texttt{c2 \#\# c3}) = \textbf{t}$ then $TVal_I(\texttt{c1 \#\# c3}) = \textbf{t}$.
5. *Membership*: $TVal_I(\texttt{o \# cl}) = \boldsymbol{I}_{truth}(\boldsymbol{I}(\texttt{o \# cl}))$.

   To ensure that all members of a subclass are also members of the superclass, i.e., `o # cl` and `cl ## scl` imply `o # scl`, the following is required:

   - For all `o`, `cl`, `scl` $\in \boldsymbol{D}$, if $TVal_I(\texttt{o \# cl}) = TVal_I(\texttt{cl \#\# scl}) = \textbf{t}$ then $TVal_I(\texttt{o \# scl}) = \textbf{t}$.
6. *Frame*: $TVal_I(\texttt{o[}a_1\texttt{->}v_1\ \texttt{...}\ a_k\texttt{->}v_k\texttt{]}) = \boldsymbol{I}_{truth}(\boldsymbol{I}(\texttt{o[}a_1\texttt{->}v_1\ \texttt{...}\ a_k\texttt{->}v_k\texttt{]}))$.

Since the bag of attribute/value pairs associated with an object `o` represents the conjunction of assertions represented by these pairs, the following is required, if `k > 0`:

- *TVal*$_I$(`o[`a_1`->`v_1 `...` a_k`->`v_k`]`) = **t** if and only if *TVal*$_I$(`o[`a_1`->`v_1`]`) = ... = *TVal*$_I$(`o[`a_k`->`v_k`]`) = **t**.

7. *Externally defined atomic formula*: *TVal*$_I$(`External(t)`) = $I_{truth}(I_{external}(\sigma)(I(s_1), ..., I(s_n)))$, if `t` is an atomic formula that is an instantiation of the external schema `σ = (?X`$_1$ `...` `?X`$_n$`; τ)` by substitution `?X`$_1$`/`s_1 `...` `?X`$_n$`/`s_1.

   Note that, by definition, `External(t)` is well-formed only if `t` is an instantiation of an external schema. Furthermore, by the definition of coherent sets of external schemas, `t` can be an instantiation of at most one such schema, so ***I***(`External(t)`) is well-defined.

8. *Conjunction*: *TVal*$_I$(`And(`c_1 ... c_n`)`) = **t** if and only if *TVal*$_I$(c_1) = ... = *TVal*$_I$(c_n) = **t**. Otherwise, *TVal*$_I$(`And(`c_1 ... c_n`)`) = **f**.

   The empty conjunction is treated as a tautology, so *TVal*$_I$(`And()`) = **t**.

9. *Disjunction*: *TVal*$_I$(`Or(`c_1 ... c_n`)`) = **f** if and only if *TVal*$_I$(c_1) = ... = *TVal*$_I$(c_n) = **f**. Otherwise, *TVal*$_I$(`Or(`c_1 ... c_n`)`) = **t**.

   The empty disjunction is treated as a contradiction, so *TVal*$_I$(`Or()`) = **f**.

10. *Quantification*:
    - *TVal*$_I$(`Exists ?`v_1 `...` `?`v_n `(`φ`)`) = **t** if and only if for some ***I****, described below, *TVal*$_{I^*}$(φ) = **t**.
    - *TVal*$_I$(`Forall ?`v_1 `...` `?`v_n `(`φ`))` = **t** if and only if for every ***I****, described below, *TVal*$_{I^*}$(φ) = **t**.

    Here ***I**** is a semantic structure of the form <***TV***, ***DTS***, ***D***, D_{ind}, D_{func}, I_C, I^*_V, I_F, I_{NF}, I_{list}, I_{tail}, I_{frame}, I_{sub}, I_{isa}, $I_=$, $I_{external}$, I_{truth}>, which is exactly like ***I***, except that the mapping I^*_V, is used instead of I_V. I^*_V is defined to coincide with I_V on all variables except, possibly, on `?`v_1,...,`?`v_n.

11. *Rule implication*:
    - *TVal*$_I$(*conclusion* :- *condition*) = **t**, if either *TVal*$_I$(*conclusion*)=**t** or *TVal*$_I$(*condition*)=**f**.
    - *TVal*$_I$(*conclusion* :- *condition*) = **f** otherwise.
12. *Groups of rules*:

    If Γ is a group formula of the form `Group(`φ$_1$ `...` φ$_n$`)` then

    - *TVal*$_I$(Γ) = **t** if and only if *TVal*$_I$(φ$_1$) = **t**, ..., *TVal*$_I$(φ$_n$) = **t**.
    - *TVal*$_I$(Γ) = **f** otherwise.

This means that a group of rules is treated as a conjunction. In particular, the empty group is treated as a tautology, so $TVal_I$(`Group()`) = **t**. □

3.5 Interpretation of Documents

Document formulas are interpreted using *semantic multi-structures*, which are sets of closely related semantics structures. The need for multi-structures arises due to the fact that a RIF-BLD document can import other documents and thus is essentially a multi-document object. One interesting aspect of the multi-document semantics is that `rif:local` symbols that belong to different documents can have different meanings.

Definition (Semantic multi-structure). A ***semantic multi-structure*** **Î** is a set of semantic structures of the form {***J***,***I***; $\boldsymbol{I}^{i_1}$, $\boldsymbol{I}^{i_2}$, ...}, where

- ***I*** and ***J*** are RIF-BLD semantic structures; and
- $\boldsymbol{I}^{i_1}$, $\boldsymbol{I}^{i_2}$, etc., are 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 $\boldsymbol{J}_C$, $\boldsymbol{I}_C$, $\boldsymbol{I}_C^{i_1}$, $\boldsymbol{I}_C^{i_2}$, ... may differ on the constants in `Const` that belong to the `rif:local` symbol space. □

As will be seen from the next definition, the structure ***I*** in the above is used to interpret document formulas, and the adorned structures of the form $\boldsymbol{I}^{i_k}$ are used to interpret imported documents. The structure ***J*** is used in the definition of entailment for non-document formulas.

The semantics of RIF documents is now defined as follows.

Definition (Truth valuation of document formulas). Let Δ be a document formula and let Δ_1, ..., Δ_n be all the RIF-BLD document formulas that are *imported* (directly or indirectly, according to Definition Imported document) into Δ. Let Γ, Γ_1, ..., Γ_n denote the respective group formulas associated with these documents. Let **Î** = {***J***,***I***; $\boldsymbol{I}^{i_1}$, ..., $\boldsymbol{I}^{i_n}$, ...} be a semantic multi-structure that contains the semantic structures adorned with the locators i_1, ..., i_n of the documents Δ_1, ..., Δ_n. Then we define:

- $TVal_I(\Delta)$ = **t** if and only if $TVal_I(\Gamma)$ = $TVal_{I^{i_1}}(\Gamma_1)$ = ... = $TVal_{I^{i_k}}(\Gamma_n)$ = **t**. □

Note that this definition considers only those document formulas that are reachable via the one-argument import directives. Two argument import directives are not

covered here. Their semantics is defined by the document RIF RDF and OWL Compatibility [RIF-RDF+OWL].

Also note that some of the Γ_i above may be missing since all parts in a document formula are optional. In this case, we assume that Γ_i is a tautology, such as `And()`, and every *TVal* function maps such a Γ_i to the truth value **t**.

For non-document formulas, we extend $TVal_{\mathcal{I}}(\varphi)$ from regular semantic structures to multi-structures as follows. Let $\hat{\mathbf{I}} = \{\boldsymbol{J}, \boldsymbol{I}; ...\}$ be a semantic multi-structure. Then $TVal_{\mathcal{I}}(\varphi) = TVal_{\boldsymbol{J}}(\varphi)$.

The above definitions make the intent behind the `rif:local` constants clear: occurrences of such constants in different documents can be interpreted differently even if they have the same name. Therefore, each document can choose the names for the `rif:local` constants freely and without regard to the names of such constants used in the imported documents.

For the relationship between `rif:local` and RDF blank nodes readers are referred to Section Symbols in RIF Versus RDF/OWL (Informative) of [RIF-RDF+OWL].

3.6 Logical Entailment

We now define what it means for a set of RIF-BLD rules (embedded in a group or a document formula) to entail another RIF-BLD formula. In RIF-BLD we are mostly interested in entailment of RIF condition formulas, which can be viewed as queries to RIF-BLD groups or documents. Entailment of condition formulas provides formal underpinning to RIF-BLD queries.

Definition (Models). A multi-structure $\hat{\mathbf{I}}$ is a ***model*** of a formula, φ, written as $\hat{\mathbf{I}} \models \varphi$, iff $TVal_{\mathcal{I}}(\varphi) = \mathbf{t}$. Here φ can be a document or a non-document formula. □

Definition (Logical entailment). Let φ and ψ be (document or non-document) formulas. We say that φ ***entails*** ψ, written as $\varphi \models \psi$, if and only if for every multi-structure, $\hat{\mathbf{I}}$, $\hat{\mathbf{I}} \models \varphi$ implies $\hat{\mathbf{I}} \models \psi$. □

Note that one consequence of the multi-document semantics of RIF-BLD is that local constants specified in one document cannot be queried from another document. For instance, if one document, Δ', has the fact `"http://example.com/ppp"^^rif:iri("abc"^^rif:local)` while another document formula, Δ, imports Δ' and has the rule `"http://example.com/qqq"^^rif:iri(?X) :- "http://example.com/ppp"^^rif:iri(?X)`, then $\Delta \models$ `"http://example.com/qqq"^^rif:iri("abc"^^rif:local)`

does *not* hold. This is because the symbol `"abc"^^rif:local` in `Δ'` and `Δ` is treated as different constants by semantic multi-structures.

This behavior of local symbols should be contrasted with the behavior of `rif:iri` symbols. Suppose, in the above scenario, `Δ'` also has the fact `"http://example.com/ppp"^^rif:iri("http://cde.example.org"^^rif:iri)`. Then `Δ |= "http://example.com/qqq"^^rif:iri("http://cde.example.org"^^rif:iri)` *does* hold.

4 XML Serialization Syntax for RIF-BLD

The RIF-BLD XML serialization defines

- a *normative* mapping from the RIF-BLD presentation syntax to XML (Section Mapping from the Presentation Syntax to the XML Syntax), and
- a *normative* XML schema for the XML syntax (Appendix XML Schema for BLD).

Recall that the syntax of RIF-BLD is not context-free and thus cannot be fully captured by EBNF or XML Schema. Still, validity with respect to XML Schema can be a useful test. To reflect this state of affairs, we define two notions of syntactic correctness. The weaker notion checks correctness only with respect to XML Schema, while the stricter notion represents "true" syntactic correctness.

Definition (Valid BLD document in XML syntax). A ***valid*** BLD document in the XML syntax is an XML document that is valid with respect to the XML schema in Appendix XML Schema for BLD. □

Definition (Admissible BLD document in XML syntax). An ***admissible*** BLD document in the XML syntax is a valid BLD document in XML syntax that is the image of a well-formed RIF-BLD document in the presentation syntax (see Definition Well-formed formula in Section Formulas) under the presentation-to-XML syntax mapping `Xbld` defined in Section Mapping from the Presentation Syntax to the XML Syntax. □

The XML serialization for RIF-BLD is based on an *alternating* or *striped* syntax [ANF01]. A striped serialization views XML documents as objects and divides all XML tags into class descriptors, called *type tags*, and property descriptors, called *role tags* [TRT03]. We follow the tradition of using capitalized names for type tags and lowercase names for role tags.

The all-uppercase classes in the presentation syntax, such as `FORMULA`, become XML Schema groups in Appendix XML Schema for BLD. They are not visible in

instance markup. The other classes as well as non-terminals and symbols (such as `Exists` or =) become XML elements with optional attributes, as shown below.

RIF-BLD uses [XML1.0] for its XML syntax.

4.1 XML for the Condition Language

XML serialization of RIF-BLD in Section EBNF for RIF-BLD Condition Language uses the following elements.

```
- And       (conjunction)
- Or        (disjunction)
- Exists    (quantified formula for 'Exists', containing declare and formul
- declare   (declare role, containing a Var)
- formula   (formula role, containing a FORMULA)
- Atom      (atom formula, positional or with named arguments)
- External  (external call, containing a content role)
- content   (content role, containing an Atom, for predicates, or Expr, for
- Member    (member formula)
- Subclass  (subclass formula)
- Frame     (Frame formula)
- object    (Member/Frame role, containing a TERM or an object description)
- op        (Atom/Expr role for predicates/functions as operations)
- args      (Atom/Expr positional arguments role, with ordered="yes" attrib
- instance  (Member instance role)
- class     (Member class role)
- sub       (Subclass sub-class role)
- super     (Subclass super-class role)
- slot      (Atom/Expr or Frame slot role, with ordered="yes" attribute, co
- Equal     (prefix version of term equation '=')
- left      (Equal left-hand side role)
- right     (Equal right-hand side role)
- Expr      (expression formula, positional or with named arguments)
- List      (list term, closed or open)
- items     (list items role, with ordered="yes" attribute, containing n TE
- rest      (list rest role, corresponding to '|')
- Const     (individual, function, or predicate symbol, with 'type' attribu
- Name      (name of named argument)
- Var       (logic variable)

- id        (identifier role, containing IRICONST)
- meta      (meta role, containing metadata as a Frame or Frame conjunction
```

The name of a base or prefix is not associated with a RIF/XML element, since it is handled via preprocessing as discussed in Section Mapping of the Rule Language.

The `id` and `meta` elements, which are expansions of the `IRIMETA` element, can occur optionally as the initial children of any Class element.

For the XML Schema definition of the RIF-BLD condition language see Appendix XML Schema for BLD.

The XML syntax for symbol spaces uses the `type` attribute associated with the XML element `Const`. For instance, a literal in the `xs:dateTime` datatype is represented as follows:

```
<Const type="&xs;dateTime">2007-11-23T03:55:44-02:30</Const>
```

The `xml:lang` attribute, as defined by 2.12 Language Identification of XML 1.0 or its successor specifications in the W3C recommendation track, is optionally used to identify the language for the presentation of the `Const` to the user. It is allowed only in association with constants of the type `rdf:plainLiteral`. A compliant implementation MUST ignore the `xml:lang` attribute if the type of the `Const` is not `rdf:plainLiteral`.

RIF-BLD also uses the `ordered="yes"` attribute to indicate that the children of `args` and `slot` elements are ordered.

Example 6 (A RIF condition and its XML serialization).

This example illustrates XML serialization for RIF conditions. As before, the compact URI notation is used for better readability. Assume that the following prefix directives are found in the preamble to the document, whose XML form will be illustrated in Example 8:

```
Prefix(bks    <http://example.com/books#>)
Prefix(cpt    <http://example.com/concepts#>)
Prefix(curr   <http://example.com/currencies#>)
Prefix(rif    <http://www.w3.org/2007/rif#>)
Prefix(xs     <http://www.w3.org/2001/XMLSchema#>)
```

RIF condition:

```
And (Exists ?Buyer (cpt:purchase(?Buyer ?Seller
                                 cpt:book(?Author bks:LeRif)
                                 curr:USD(49)))
     ?Seller=?Author )
```

XML serialization:

```
<And>
  <formula>
```

```
          <Exists>
            <declare><Var>Buyer</Var></declare>
            <formula>
              <Atom>
                <op><Const type="&rif;iri">&cpt;purchase</Const></op>
                <args ordered="yes">
                  <Var>Buyer</Var>
                  <Var>Seller</Var>
                  <Expr>
                    <op><Const type="&rif;iri">&cpt;book</Const></op>
                    <args ordered="yes">
                      <Var>Author</Var>
                      <Const type="&rif;iri">&bks;LeRif</Const>
                    </args>
                  </Expr>
                  <Expr>
                    <op><Const type="&rif;iri">&curr;USD</Const></op>
                    <args ordered="yes"><Const type="&xs;integer">49</Const></
                  </Expr>
                </args>
              </Atom>
            </formula>
          </Exists>
        </formula>
        <formula>
          <Equal>
            <left><Var>Seller</Var></left>
            <right><Var>Author</Var></right>
          </Equal>
        </formula>
      </And>
```

Example 7 (An XML serialization of a RIF condition with a frame and a named-argument term).

This example illustrates XML serialization of RIF conditions that involve terms with named arguments. As in Example 6, we assume the following prefix directives, whose XML form will be illustrated in Example 8:

```
Prefix(bks    <http://example.com/books#>)
Prefix(cpt    <http://example.com/concepts#>)
Prefix(curr   <http://example.com/currencies#>)
Prefix(rif    <http://www.w3.org/2007/rif#>)
Prefix(xs     <http://www.w3.org/2001/XMLSchema#>)
```

RIF condition:

```
And (Exists ?Buyer ?P (
               And (?P#cpt:purchase
                    ?P[cpt:buyer->?Buyer
                       cpt:seller->?Seller
                       cpt:item->cpt:book(cpt:author->?Author cpt:title->
                       cpt:price->49
                       cpt:currency->curr:USD]))
     ?Seller=?Author)
```

XML serialization:

```
<And>
  <formula>
    <Exists>
      <declare><Var>Buyer</Var></declare>
      <declare><Var>P</Var></declare>
      <formula>
        <And>
          <formula>
            <Member>
              <instance><Var>P</Var></instance>
              <class><Const type="&rif;iri">&cpt;purchase</Const></class
            </Member>
          </formula>
          <formula>
            <Frame>
              <object>
                <Var>P</Var>
              </object>
              <slot ordered="yes">
                <Const type="&rif;iri">&cpt;buyer</Const>
                <Var>Buyer</Var>
              </slot>
              <slot ordered="yes">
                <Const type="&rif;iri">&cpt;seller</Const>
                <Var>Seller</Var>
              </slot>
              <slot ordered="yes">
                <Const type="&rif;iri">&cpt;item</Const>
                <Expr>
                  <op><Const type="&rif;iri">&cpt;book</Const></op>
                  <slot ordered="yes">
                    <Name>&cpt;author</Name>
                    <Var>Author</Var>
                  </slot>
                  <slot ordered="yes">
```

```
                    <Name>&cpt;title</Name>
                    <Const type="&rif;iri">&bks;LeRif</Const>
                  </slot>
                </Expr>
              </slot>
              <slot ordered="yes">
                <Const type="&rif;iri">&cpt;price</Const>
                <Const type="&xs;integer">49</Const>
              </slot>
              <slot ordered="yes">
                <Const type="&rif;iri">&cpt;currency</Const>
                <Const type="&rif;iri">&curr;USD</Const>
              </slot>
            </Frame>
          </formula>
        </And>
      </formula>
    </Exists>
  </formula>
  <formula>
    <Equal>
      <left><Var>Seller</Var></left>
      <right><Var>Author</Var></right>
    </Equal>
  </formula>
</And>
```

4.2 XML for the Rule Language

We now extend the set of RIF-BLD serialization elements from Section XML for RIF-BLD Condition Language by including rules, along with their enclosing groups and documents, as described in Section EBNF for RIF-BLD Rule Language. The extended set includes the tags listed below.

```
- Document  (document, containing optional directive and payload roles)
- directive (directive role, containing Import)
- payload   (payload role, containing Group)
- Import    (importation, containing location and optional profile)
- location  (location role, containing ANYURICONST)
- profile   (profile role, containing PROFILE)
- Group     (nested collection of sentences)
- sentence  (sentence role, containing RULE or Group)
- Forall    (quantified formula for 'Forall', containing declare and formul
- Implies   (implication, containing if and then roles)
- if        (antecedent role, containing FORMULA)
- then      (consequent role, containing ATOMIC or conjunction of ATOMICs)
```

The XML Schema Definition of RIF-BLD is given in Appendix XML Schema for BLD.

While there is a RIF-BLD element tag for the `Import` directive, the `Base` and `Prefix` directives are not represented by RIF/XML tags: they are handled as follows (see also Section Mapping of the Rule Language). A `Base` directive in the presentation syntax becomes an `xml:base` attribute [XML-Base] in the XML `Document` tag. The base IRI specified as the value of that attribute applies to content of the RIF/XML element that deals with `rif:iri` constants, namely to relative-IRI content of the `<Const type="&rif;iri">` element. A collection of `Prefix` directives in the presentation syntax becomes a `DOCTYPE` DTD [XML1.0] preceding the RIF-BLD `Document` and containing a declaration of an `ENTITY` for each `Prefix` directive.

Example 8 (Serializing a RIF-BLD document containing an annotated group).

This example shows a serialization for the document from Example 5. For convenience, the presentation syntax is reproduced at the top, and is followed by its serialization. The base IRI `http://example.com/people#` applies to the relative `rif:iri` constants with content `John` (twice) and `Fred` (once).

Presentation syntax:

```
Document(
  Base(<http://example.com/people#>)
  Prefix(cpt  <http://example.com/concepts#>)
  Prefix(dc   <http://purl.org/dc/terms/>)
  Prefix(rif  <http://www.w3.org/2007/rif#>)
  Prefix(func <http://www.w3.org/2007/rif-builtin-function#>)
  Prefix(pred <http://www.w3.org/2007/rif-builtin-predicate#>)
  Prefix(xs   <http://www.w3.org/2001/XMLSchema#>)

  (* "http://sample.org"^^rif:iri _pd[dc:publisher -> "http://www.w3.org/"^
                                     dc:date -> "2008-04-04"^^xs:date] *)
  Group
  (
    Forall ?item ?deliverydate ?scheduledate ?diffduration ?diffdays (
        cpt:reject(<John> ?item) :-
            And(cpt:perishable(?item)
                cpt:delivered(?item ?deliverydate <John>)
                cpt:scheduled(?item ?scheduledate)
                ?diffduration = External(func:subtract-dateTimes(?deliveryd
                ?diffdays = External(func:days-from-duration(?diffduration)
                External(pred:numeric-greater-than(?diffdays 10)))
    )

    Forall ?item (
```

```
        cpt:reject(<Fred> ?item) :- cpt:unsolicited(?item)
    )
  )
)
```

XML syntax:

```
<!DOCTYPE Document [
  <!ENTITY cpt  "http://example.com/concepts#">
  <!ENTITY dc   "http://purl.org/dc/terms/">
  <!ENTITY rif  "http://www.w3.org/2007/rif#">
  <!ENTITY func "http://www.w3.org/2007/rif-builtin-function#">
  <!ENTITY pred "http://www.w3.org/2007/rif-builtin-predicate#">
  <!ENTITY xs   "http://www.w3.org/2001/XMLSchema#">
]>

<Document
    xml:base="http://example.com/people#"
    xmlns="http://www.w3.org/2007/rif#"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xs="http://www.w3.org/2001/XMLSchema#">
  <payload>
   <Group>
    <id>
      <Const type="&rif;iri">http://sample.org</Const>
    </id>
    <meta>
      <Frame>
        <object>
          <Const type="&rif;local">pd</Const>
        </object>
        <slot ordered="yes">
          <Const type="&rif;iri">&dc;publisher</Const>
          <Const type="&rif;iri">http://www.w3.org/</Const>
        </slot>
        <slot ordered="yes">
          <Const type="&rif;iri">&dc;date</Const>
          <Const type="&xs;date">2008-04-04</Const>
        </slot>
      </Frame>
    </meta>
    <sentence>
     <Forall>
       <declare><Var>item</Var></declare>
       <declare><Var>deliverydate</Var></declare>
       <declare><Var>scheduledate</Var></declare>
```

```
<declare><Var>diffduration</Var></declare>
<declare><Var>diffdays</Var></declare>
<formula>
  <Implies>
    <if>
      <And>
        <formula>
          <Atom>
            <op><Const type="&rif;iri">&cpt;perishable</Const></op>
            <args ordered="yes"><Var>item</Var></args>
          </Atom>
        </formula>
        <formula>
          <Atom>
            <op><Const type="&rif;iri">&cpt;delivered</Const></op>
            <args ordered="yes">
              <Var>item</Var>
              <Var>deliverydate</Var>
              <Const type="&rif;iri">John</Const>
            </args>
          </Atom>
        </formula>
        <formula>
          <Atom>
            <op><Const type="&rif;iri">&cpt;scheduled</Const></op>
            <args ordered="yes">
              <Var>item</Var>
              <Var>scheduledate</Var>
            </args>
          </Atom>
        </formula>
        <formula>
          <Equal>
            <left><Var>diffduration</Var></left>
            <right>
              <External>
                <content>
                  <Expr>
                    <op><Const type="&rif;iri">&func;subtract-dateTi
                    <args ordered="yes">
                      <Var>deliverydate</Var>
                      <Var>scheduledate</Var>
                    </args>
                  </Expr>
                </content>
              </External>
            </right>
          </Equal>
```

```
            </formula>
            <formula>
              <Equal>
                <left><Var>diffdays</Var></left>
                <right>
                  <External>
                    <content>
                      <Expr>
                        <op><Const type="&rif;iri">&func;days-from-durat
                        <args ordered="yes">
                          <Var>diffduration</Var>
                        </args>
                      </Expr>
                    </content>
                  </External>
                </right>
              </Equal>
            </formula>
            <formula>
              <External>
                <content>
                  <Atom>
                    <op><Const type="&rif;iri">&pred;numeric-greater-tha
                    <args ordered="yes">
                      <Var>diffdays</Var>
                      <Const type="&xs;integer">10</Const>
                    </args>
                  </Atom>
                </content>
              </External>
            </formula>
          </And>
        </if>
        <then>
          <Atom>
            <op><Const type="&rif;iri">&cpt;reject</Const></op>
            <args ordered="yes">
              <Const type="&rif;iri">John</Const>
              <Var>item</Var>
            </args>
          </Atom>
        </then>
      </Implies>
    </formula>
  </Forall>
</sentence>
<sentence>
  <Forall>
```

```
        <declare><Var>item</Var></declare>
        <formula>
          <Implies>
            <if>
              <Atom>
                <op><Const type="&rif;iri">&cpt;unsolicited</Const></op>
                <args ordered="yes"><Var>item</Var></args>
              </Atom>
            </if>
            <then>
              <Atom>
                <op><Const type="&rif;iri">&cpt;reject</Const></op>
                <args ordered="yes">
                  <Const type="&rif;iri">Fred</Const>
                  <Var>item</Var>
                </args>
              </Atom>
            </then>
          </Implies>
        </formula>
      </Forall>
    </sentence>
  </Group>
 </payload>
</Document>
```

4.3 Mapping from the Presentation Syntax to the XML Syntax

This section defines a normative mapping, χ_{bld}, from the presentation syntax to the XML syntax of RIF-BLD. The mapping is given via tables where each row specifies the mapping of a particular syntactic pattern in the presentation syntax. These patterns appear in the first column of the tables and the ***bold-italic*** symbols represent metavariables. The second column represents the corresponding XML patterns, which may contain applications of the mapping χ_{bld} to these metavariables. When an expression $\chi_{bld}(\boldsymbol{metavar})$ occurs in an XML pattern in the right column of a translation table, it should be understood as a recursive application of χ_{bld} to the presentation syntax represented by the metavariable. The XML syntax result of such an application is substituted for the expression $\chi_{bld}(\boldsymbol{metavar})$. A sequence of terms containing metavariables with subscripts is indicated by an ellipsis. For the subscript m it is understood that $m \geq 1$, i.e. the ellipsis indicates at least one term. For the subscript n it is understood that $n \geq 0$, i.e. the ellipsis indicates zero or more terms. A metavariable or a well-formed XML subelement is marked as optional by appending a bold-italic question mark, ***?***, on its right.

4.3.1 Mapping of the Condition Language

The χ_{bld} mapping from the presentation syntax to the XML syntax of the RIF-BLD Condition Language is specified by the table below. Each row indicates a translation χ_{bld}(Presentation) = XML. The function *remove-outer-quotes* used in the translation removes enclosing double quotes from a string and leaves unquoted strings untouched. Since the presentation syntax of RIF-BLD is context sensitive, the mapping must differentiate between the terms that occur in the position of individuals and the terms that occur as atomic formulas. To this end, in the translation table, the positional and named-argument terms that occur in the context of atomic formulas are denoted by expressions of the form ***pred***(...) and the terms that occur as individuals are denoted by expressions of the form ***func***(...). In the table, each metavariable for an (unnamed) positional $\boldsymbol{argument_i}$ is assumed to be instantiated to values unequal to the instantiations of named arguments $\boldsymbol{name_j}$ -> $\boldsymbol{filler_j}$. Regarding the last but first row, we assume that shortcuts for constants [RIF-DTB] have already been expanded to their full form ("...")^^***symspace***).

Presentation Syntax	XML Syntax
And ($\boldsymbol{conjunct_1}$ . . . $\boldsymbol{conjunct_n}$)	<And> <formula>χ_{bld}(***conjunct*$_1$**)</formula> . . . <formula>χ_{bld}(***conjunct*$_n$**)</formula> </And>
Or ($\boldsymbol{disjunct_1}$ . . . $\boldsymbol{disjunct_n}$)	<Or> <formula>χ_{bld}(***disjunct*$_1$**)</formula> . . . <formula>χ_{bld}(***disjunct*$_n$**)</formula> </Or>
Exists $\boldsymbol{variable_1}$ . . . $\boldsymbol{variable_n}$ (***premise***)	<Exists> <declare>χ_{bld}(***variable*$_1$**)</declare> . . . <declare>χ_{bld}(***variable*$_n$**)</declare> <formula>χ_{bld}(***premise***)</formula> </Exists>

External (
  atomexpr
)
	<External>
	  <content>χ_{bld}(***atomexpr***)</content>
	</External>

pred (
  argument $_1$
  . . .
  argument $_n$
)
	<Atom>
	  <op>χ_{bld}(***pred***)</op>
	  <args ordered="yes">
	  χ_{bld}(***argument*** $_1$)
	  . . .
	  χ_{bld}(***argument*** $_n$)
	  </args>
	</Atom>

func (
  argument $_1$
  . . .
  argument $_n$
)
	<Expr>
	  <op>χ_{bld}(***func***)</op>
	  <args ordered="yes">
	  χ_{bld}(***argument*** $_1$)
	  . . .
	  χ_{bld}(***argument*** $_n$)
	  </args>
	</Expr>

List (
  element $_1$
  . . .
  element $_n$
)
	<List>
	  <items ordered="yes">
	  χ_{bld}(***element*** $_1$)
	  . . .
	  χ_{bld}(***element*** $_n$)
	  </items>
	</List>

List (
  element $_1$
  . . .
  element $_n$
  |
  remainder
)
	<List>
	  <items ordered="yes">
	  χ_{bld}(***element*** $_1$)
	  . . .
	  χ_{bld}(***element*** $_n$)
	  </items>
	  <rest>χ_{bld}(***remainder***)</rest>
	</List>

pred (
  name $_1$ -> ***filler*** $_1$
  . . .
	<Atom>
	  <op>χ_{bld}(***pred***)</op>
	  <slot ordered="yes">
	  <Name>χ_{bld}(***name*** $_1$)</Name>

name$_n$ -> *filler*$_n$)	χ_{bld}(*filler*$_1$) </slot> . . . <slot ordered="yes"> <Name>χ_{bld}(*name*$_n$)</Name> χ_{bld}(*filler*$_n$) </slot> </Atom>
func (*name*$_1$ -> *filler*$_1$ . . . *name*$_n$ -> *filler*$_n$)	<Expr> <op>χ_{bld}(*func*)</op> <slot ordered="yes"> <Name>χ_{bld}(*name*$_1$)</Name> χ_{bld}(*filler*$_1$) </slot> . . . <slot ordered="yes"> <Name>χ_{bld}(*name*$_n$)</Name> χ_{bld}(*filler*$_n$) </slot> </Expr>
inst [*key*$_1$ -> *filler*$_1$ . . . *key*$_n$ -> *filler*$_n$]	<Frame> <object>χ_{bld}(*inst*)</object> <slot ordered="yes"> χ_{bld}(*key*$_1$) χ_{bld}(*filler*$_1$) </slot> . . . <slot ordered="yes"> χ_{bld}(*key*$_n$) χ_{bld}(*filler*$_n$) </slot> </Frame>
inst # *class*	<Member> <instance>χ_{bld}(*inst*)</instance> <class>χ_{bld}(*class*)</class> </Member>
sub ## *super*	<Subclass> _{χ_{bld}(*sub*)} <super>χ_{bld}(*super*)</super> </Subclass>

left = ***right***	<Equal> <left>χ_{bld}(***left***)</left> <right>χ_{bld}(***right***)</right> </Equal>
"***unicodestring***"^^***symspace***	<Const type="***symspace***">***unicodestring***</Con
?***name*** $_1$	<Var>χ_{bld}(***name*** $_1$)</Var>
name $_i$	*remove-outer-quotes*(***name*** $_i$)

4.3.2 Mapping of the Rule Language

The χ_{bld} mapping from the presentation syntax to the XML syntax of the RIF-BLD Rule Language is specified by the table below. It extends the translation table of Section Mapping of the Condition Language. While the `Import` directive is handled by the presentation-to-XML syntax mapping, the `Prefix` and `Base` directives are not. Instead, these directives should be handled by expanding the associated shortcuts (compact URIs). Namely, a prefix name declared in a `Prefix` directive is expanded into the associated IRI, while relative IRIs are completed using the IRI declared in the `Base` directive. The mapping χ_{bld} applies only to such expanded documents. RIF-BLD also allows other treatments of `Prefix` and `Base` provided that they produce equivalent XML documents. One such treatment is employed in the examples in this document, especially Example 8. It replaces prefix names with definitions of XML entities as follows. Each `Prefix` declaration becomes an `ENTITY` declaration [XML1.0] within a `DOCTYPE` DTD attached to the RIF-BLD `Document`. The `Base` directive is mapped to the `xml:base` attribute [XML-Base] in the XML `Document` tag. Compact URIs of the form `prefix:suffix` are then mapped to `&prefix;suffix`.

Presentation Syntax	XML Syntax
Document(Import(***loc*** $_1$ ***prfl*** $_1$**?**) . . . Import(***loc*** $_n$ ***prfl*** $_n$**?**)	<Document> <directive> <Import> <location>χ_{bld}(***loc*** $_1$)</location> <profile>χ_{bld}(***prfl*** $_1$)</profile>**?**

group?)	</Import> </directive> . . . <directive> <Import> <location>χ_{bld}(*loc_n*)</location> <profile>χ_{bld}(*$prfl_n$*)</profile>*?* </Import> </directive> <payload>χ_{bld}(*group*)</payload>*?* </Document>
Group(*$clause_1$* . . . *$clause_n$*)	<Group> <sentence>χ_{bld}(*$clause_1$*)</sentence> . . . <sentence>χ_{bld}(*$clause_n$*)</sentence> </Group>
Forall *$variable_1$* . . . *$variable_n$* (*rule*)	<Forall> <declare>χ_{bld}(*$variable_1$*)</declare> . . . <declare>χ_{bld}(*$variable_n$*)</declare> <formula>χ_{bld}(*rule*)</formula> </Forall>
conclusion :- *condition*	<Implies> <if>χ_{bld}(*condition*)</if> <then>χ_{bld}(*conclusion*)</then> </Implies>

4.3.3 Mapping of Annotations

The χ_{bld} mapping from RIF-BLD annotations in the presentation syntax to the XML syntax is specified by the table below. It extends the translation tables of Sections Mapping of the Condition Language and Mapping of the Rule Language. The metavariable ***Typetag*** in the presentation and XML syntaxes stands for any of the class names `And`, `Or`, `External`, `Document`, or `Group`, and ***Quantifier*** for `Exists` or `Forall`. The dollar sign, **$**, stands for any of the binary infix operator names `#`, `##`, `=`, or `:-`, while ***Binop*** stands for their respective class names `Member`, `Subclass`, `Equal`, or `Implies`. Again, each metavariable for an (unnamed) positional ***$argument_i$*** is assumed to be instantiated to values unequal to the instantiations of named arguments ***$name_j$*** -> ***$filler_j$***.

Presentation Syntax	
(* ***iriconst? frameconj?*** *) ***Typetag*** (e_1 . . . e_n)	<***Typetag***> <id>χbld(***iriconst*** <meta>χbld(***frameo*** e_1' . . . e_n' </***Typetag***> where e_1', . . ., χbld(***Typetag***(e_1 . .
(* ***iriconst? frameconj?*** *) ***Quantifier variable***$_1$ . . . ***variable***$_n$ (***formula***)	<***Quantifier***> <id>χbld(***iriconst*** <meta>χbld(***frameo*** <declare>χbld(***var*** . . . <declare>χbld(***var*** <formula>χbld(***for*** </***Quantifier***>
(* ***iriconst? frameconj?*** *) ***pred*** (***argument***$_1$ . . . ***argument***$_n$)	<Atom> <id>χbld(***iriconst*** <meta>χbld(***frameo*** <op>χbld(***pred***)</o <args ordered="y χbld(***argument***$_1$) . . . χbld(***argument***$_n$) </args> </Atom>
(* ***iriconst? frameconj?*** *) ***func*** (***argument***$_1$ . . . ***argument***$_n$)	<Expr> <id>χbld(***iriconst*** <meta>χbld(***frameo*** <op>χbld(***func***)</o <args ordered="y χbld(***argument***$_1$) . . . χbld(***argument***$_n$) </args> </Expr>
(* ***iriconst? frameconj?*** *) List (	<List> <id>χbld(***iriconst***

***element*$_1$** . . . ***element*$_n$**)	<meta>χbld(***frame*** <items ordered=" χbld(***element*$_1$**) . . . χbld(***element*$_n$**) </items> </List>
(* ***iriconst? frameconj?*** *) List (***element*$_1$** . . . ***element*$_n$** | ***remainder***)	<List> <id>χbld(***iriconst*** <meta>χbld(***frame*** <items ordered=" χbld(***element*$_1$**) . . . χbld(***element*$_n$**) </items> <rest>χbld(***remai*** </List>
(* ***iriconst? frameconj?*** *) ***pred*** (***name*$_1$** -> ***filler*$_1$** . . . ***name*$_n$** -> ***filler*$_n$**)	<Atom> <id>χbld(***iriconst*** <meta>χbld(***frame*** <op>χbld(***pred***)</o <slot ordered="y <Name>χbld(***name*** χbld(***filler*$_1$**) </slot> . . . <slot ordered="y <Name>χbld(***name*** χbld(***filler*$_n$**) </slot> </Atom>
(* ***iriconst? frameconj?*** *) ***func*** (***name*$_1$** -> ***filler*$_1$** . . . ***name*$_n$** -> ***filler*$_n$**)	<Expr> <id>χbld(***iriconst*** <meta>χbld(***frame*** <op>χbld(***func***)</o <slot ordered="y <Name>χbld(***name*** χbld(***filler*$_1$**) </slot> . . . <slot ordered="y <Name>χbld(***name*** χbld(***filler*$_n$**)

	</slot> </Expr>
(* ***iriconst? frameconj?*** *) ***inst*** [***key*** $_1$ -> ***filler*** $_1$ . . . ***key*** $_n$ -> ***filler*** $_n$]	<Frame> <id>χ_{bld}(***iriconst*** <meta>χ_{bld}(***framec*** <object>χ_{bld}(***inst*** <slot ordered="y χ_{bld}(***key*** $_1$) χ_{bld}(***filler*** $_1$) </slot> . . . <slot ordered="y χ_{bld}(***key*** $_n$) χ_{bld}(***filler*** $_n$) </slot> </Frame>
(* ***iriconst? frameconj?*** *) ***e*** $_1$ ***$ e*** $_2$	<***Binop***> <id>χ_{bld}(***iriconst*** <meta>χ_{bld}(***framec*** ***e*** $_1$***' e*** $_2$***'*** </***Binop***> where ***Binop, e*** $_1$***',*** χ_{bld}(***e*** $_1$ ***$ e*** $_2$) = <***Bi***
(* ***iriconst? frameconj?*** *) ***unicodestring^^symspace***	<Const type="***symsp*** <id>χ_{bld}(***iriconst*** <meta>χ_{bld}(***framec*** ***unicodestring*** </Const>
(* ***iriconst? frameconj?*** *) ?***name*** $_1$	<Var> <id>χ_{bld}(***iriconst*** <meta>χ_{bld}(***framec*** χ_{bld}(***name*** $_1$) </Var>

5 Conformance Clauses

RIF-BLD does not require or expect conformant systems to implement the RIF-BLD presentation syntax. Instead, conformance is described in terms of semantics-preserving transformations between the native syntax of a compliant system and the XML syntax of RIF-BLD.

Let T be a set of datatypes and symbol spaces that includes the datatypes specified in [RIF-DTB], and the symbol spaces `rif:iri`, and `rif:local`. Suppose E is a coherent set of external schemas that includes the built-ins listed in [RIF-DTB]. We say that a formula φ is a $BLD_{T,E}$ formula iff

- it is a well-formed BLD formula,
- all datatypes and symbol spaces used in φ are in T, and
- all externally defined terms used in φ are instantiations of external schemas from E.

A RIF processor is a ***conformant*** $BLD_{T,E}$ ***consumer*** iff it implements a *semantics-preserving mapping*, μ, from the set of all $BLD_{T,E}$ formulas to the language *L* of the processor (μ does not need to be an "onto" mapping).

Formally, this means that for any pair φ, ψ of $BLD_{T,E}$ formulas for which $\varphi \models_{BLD} \psi$ is defined, $\varphi \models_{BLD} \psi$ iff $\mu(\varphi) \models_{L} \mu(\psi)$. Here $\models_{BLD}$ denotes the logical entailment in RIF-BLD and $\models_{L}$ is the logical entailment in the language *L* of the RIF processor.

A RIF processor is a ***conformant*** $BLD_{T,E}$ ***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 (ν does not need to be an "onto" mapping).

Formally, this means that for any pair φ, ψ of formulas in *L* for which $\varphi \models_{L} \psi$ is defined, $\varphi \models_{L} \psi$ iff $\nu(\varphi) \models_{BLD} \nu(\psi)$.

An ***admissible document*** is one which conforms to all the syntactic constraints of RIF-BLD, including ones that cannot be checked by an XML Schema validator (cf. Definition Admissible BLD document in XML syntax).

The above definitions are specializations to BLD of the general conformance clauses defined in the RIF framework for logic dialects [RIF-FLD]. The following clauses are further restrictions that are specific to RIF-BLD.

RIF-BLD specific clauses

- Conformant BLD producers and consumers are required to support only the entailments of the form $\varphi \models_{BLD} \psi$, where ψ is a *closed RIF condition formula*, i.e., a RIF condition in which every variable, `?V`, is in the scope of a quantifier of the form `Exists ?V`. In addition, conformant BLD producers and consumers *should* preserve all annotations where possible.
- A ***conformant BLD consumer*** must reject any document containing features it does not support.

- A ***conformant BLD producer*** is a conformant $BLD_{T,E}$ producer, which produces documents that include only the datatypes and externals that are required by BLD.

RIF-BLD supports a wide variety of syntactic forms for terms and formulas, which creates infrastructure for exchanging syntactically diverse rule languages. It is important to realize, however, that the above conformance statements make it possible for systems that do not support some of the syntax directly to still support it through syntactic transformations. For instance, disjunctions in rule premises can be eliminated through a standard transformation, such as replacing `p :- Or(q r)` with a pair of rules `p :- q, p :- r`. Terms with named arguments can be reduced to positional terms by ordering the arguments by their names and incorporating the ordered argument names into the predicate name. For instance, `p(bb->1 aa->2)` can be represented as `p_aa_bb(2 1)`.

6 RIF-BLD as a Specialization of the RIF Framework for Logic Dialects [RIF-FLD]

This normative section describes RIF-BLD by specializing RIF-FLD. The reader is assumed to be familiar with RIF-FLD as described in RIF framework for logic dialects [RIF-FLD]. The reader who is not interested in how RIF-BLD is derived from the framework can skip this section.

6.1 The Presentation Syntax of RIF-BLD as a Specialization of RIF-FLD

This section defines the precise relationship between the presentation syntax of RIF-BLD and the syntactic framework of RIF-FLD.

The presentation syntax of the RIF Basic Logic Dialect is defined by specialization from the presentation syntax of the RIF Syntactic Framework for Logic Dialects described in [RIF-FLD]. Section Syntax of a RIF Dialect as a Specialization of the RIF Framework in [RIF-FLD] lists the parameters of the syntactic framework in mathematical English, which we will now specialize for RIF-BLD.

1. *Extension points*.

   All extension points of RIF-FLD are removed (specialized by replacing them with zero objects).

2. *Alphabet*.

   The alphabet of the RIF-BLD presentation syntax is the alphabet of RIF-FLD with the symbols `Dialect`, `Neg`, and `Naf` excluded.

3. *Assignment of signatures to each constant and variable symbol*.

   The signature set of RIF-BLD contains the following signatures:

   a. Basic
      - `individual{ }`
      - `atomic{ }`

      The signature `individual{ }` represents the context in which individual objects (but not atomic formulas) can appear.
      The signature `atomic{ }` represents the context where atomic formulas can occur.

   b. Signatures for lists
      - The signature `list` for closed lists. It has the following arrow expressions: `() ⇒ individual`, `(individual) ⇒ individual`, `(individual individual) ⇒ individual`, ...
      - The signature `openlist` for open lists (that have tails). It has the following arrow expressions: `(individual individual) ⇒ individual`, `(individual individual individual) ⇒ individual`, ...

   c. Signatures for functions, predicates, and external functions and predicates
      - Function signature `f`. This signature has the arrow expressions for positional functions: `() ⇒ individual`, `(individual) ⇒ individual`, `(individual individual) ⇒ individual`, ..., plus the arrow expressions for functions with named arguments: `(s1->individual ... sk->individual) ⇒ individual}`, for all k>0 and all `s1, ..., sk` ∈ `ArgNames`.
      - Predicate signature `p`. This signature has the arrow expressions for positional predicates: `() ⇒ atomic`, `(individual) ⇒ atomic`, `(individual individual) ⇒ atomic`, ..., plus the arrow expressions for predicates with named arguments: `(s1->individual ... sk->individual) ⇒ atomic}`, for all k>0 and all `s1, ..., sk` ∈ `ArgNames`.

      In the RIF-BLD specialization of RIF-FLD, the argument names `s1`, ..., `sk` must be pairwise distinct.

   d. Every symbol in `Const` has exactly one signature: `individual`, `f`, or `p`.

A constant cannot have the signature `atomic` -- only complex terms can have such signatures. Thus, by itself a symbol, `s`, cannot be a proposition in RIF-BLD, but a term of the form `s()` can.

According to the above, each constant symbol in RIF-BLD can be either an individual, a function, or a predicate. However, the same function or predicate symbol (normal or external) can occur with different numbers of arguments in different places. Also, the additional restrictions spelled out below ensure that the same symbol cannot represent both an externally defined predicate or function and a regular predicate or function.

e. The constant symbols that correspond to RIF datatypes (XML Schema datatypes, `rdf:XMLLiteral`, `rdf:PlainLiteral`, etc.) all have the signature `individual` in RIF-BLD.
f. The symbols of type `rif:iri` and `rif:local` can have the following signatures in RIF-BLD: `individual`, `f`, or `p`.
g. All variables are associated with signature `individual`, so they can range only over individuals.
h. The signature for equality is `=\{(individual individual) ⇒ atomic\}`.

   This means that equality can compare only those terms whose signature is `individual`; it cannot compare predicate or function symbols. Equality terms are also not allowed to occur inside other terms, since the above signature implies that any term of the form `t = s` has signature `atomic` and not `individual`.

i. The frame signature, `->`, is `->\{(individual individual individual) ⇒ atomic\}`.

   Note that this precludes the possibility that a frame term might occur as an argument to a predicate, a function, or inside some other term.

j. The membership signature, `#`, is `#\{(individual individual) ⇒ atomic\}`.

   Note that this precludes the possibility that a membership term might occur as an argument to a predicate, a function, or inside some other term.

k. The signature for the subclass relationship is `##\{(individual individual) ⇒ atomic\}`.

As with frames and membership terms, this precludes the possibility that a subclass term might occur inside some other term.

RIF-BLD uses no special syntax for declaring signatures. Instead, the rule author specifies signatures *contextually*. That is, since RIF-BLD requires that each symbol is associated with a unique signature, the signature is determined from the context in which the symbol is used. If a symbol is used in more than one context, the parser must treat this as a syntax error. If no errors are found, all terms and atomic formulas are guaranteed to be well-formed. Thus, signatures are *not* part of the RIF-BLD language, and `individual`, `atomic`, `list`, etc., are not reserved keywords.

4. *Supported types of terms*.
   - RIF-BLD supports the following types of terms defined by the syntactic framework (see the Section Terms of [RIF-FLD]):
     a. constants
     b. variables
     c. positional
     d. with named arguments
     e. equality
     f. frame
     g. membership
     h. subclass
     i. list
     j. external
     k. formula
   - Compared to RIF-FLD, terms (both positional and with named arguments) have significant restrictions, which are intended to keep BLD relatively simple.
     - *Variables*: The signature for the variable symbols does not permit them to occur in the context of predicates, functions, or formulas. In particular, in the RIF-BLD specialization of RIF-FLD, a variable is not an atomic formula.
     - *Symbols as atomic formulas*: A symbol cannot be an atomic formula by itself. That is, if `s` ∈ `Const` then `s` is not a well-formed atomic formula. However, `s()` can be an atomic formula. Note that Propositional Logic can thus be expressed in RIF-BLD using zero-argument predicate formulas.
     - *Functions and predicates*: Signatures permit only constant symbols to occur in the context of functions or predicate. Indeed, RIF-BLD signatures ensure that all variables have the signature `individual{ }` and all other terms, except for the constants from `Const`, can have either the signature `individual{ }` or `atomic{ }`. Therefore, if `t` is a (non-`Const`) term then `t(...)` is not a well-formed term.

- *External*:
  - A term, `t`, may appear as an external term or an external atomic formula (i.e., in the context `External(t)`) *if and only if* `t` is an instantiation of an externally defined schema from the coherent set of external schemas associated with the language.

    It thus follows that a predicate or a function symbol, `s`, that occurs in an external term `External(s(...))` cannot also occur as a non-external symbol.

  - In an externally defined term, `External(t)`, the subterm `t` can be only a positional or a named-argument term of the form `s(...)` where `s` has the signature `f` or `p`. Compared to RIF-FLD, this restricts `t` so that it cannot be a constant, a frame, an equality, or a classification term. RIF-FLD's two-argument form of `External` is not supported in RIF-BLD either.
- *Formula*: The formula terms are restricted, as specified in Supported formulas below. In particular, they cannot appear as arguments to predicates, functions, etc.

5. No aggregate terms or module terms are allowed. (In RIF-FLD, formula terms correspond to compound formulas that involve logical connectives and quantifiers.)
6. *Required symbol spaces*.

   RIF-BLD requires the symbol spaces defined in Section Constants, Symbol Spaces, and Datatypes of [RIF-DTB].

7. *Supported formulas*.

   RIF-BLD supports the following types of formulas (see Well-formed Terms and Formulas in [RIF-FLD] for the definitions):

   - ***RIF-BLD condition***

     A RIF-BLD condition is an atomic formula, and external atomic formula, or a conjunctive or disjunctive combination of such atomic formulas. All these formulas and their combinations can be optionally preceded with existential quantifiers.

   - ***RIF-BLD rule***

     A RIF-BLD rule is a universally quantified RIF-FLD rule with the following restrictions:

        - The conclusion of the rule is an atomic formula or a conjunction of atomic formulas.
        - None of the atomic formulas mentioned in the rule conclusion is externally defined (i.e., cannot have the form `External(...)`).
        - The premise of the rule is a RIF-BLD condition.
        - All free (non-quantified) variables in the rule must be quantified with `Forall` outside of the rule (i.e., `Forall ?vars (conclusion :- premise)`).
    - ***Universal fact***

      A universal fact is a universally quantified atomic formula with no free variables.

    - ***RIF-BLD group***

      A RIF-BLD group is a RIF-FLD group that contains only RIF-BLD rules, universal facts, variable-free rule implications, variable-free atomic formulas, and RIF-BLD groups.

    - ***RIF-BLD document***

      A RIF-BLD document is a RIF-FLD document that consists of directives and a RIF-BLD group formula. There is no `Dialect` or `Module` directives, and the `Import(<loc>)` directive (with one argument) can import RIF-BLD documents only. Here `loc` must be a Unicode character sequence that forms an IRI. There are no BLD-specific restrictions on the two-argument directive `Import` except that the second argument must be a Unicode sequence of characters of the form `<loc>`, where `loc` is an IRI.

Negation (`Neg` and `Naf`) is not allowed in RIF-BLD rules: neither in rule conclusions nor in premises.

6.2 The Semantics of RIF-BLD as a Specialization of RIF-FLD

This normative section defines the precise relationship between the semantics of RIF-BLD and the semantic framework of RIF-FLD. Specification of the semantics that does not rely on RIF-FLD is given in Section Direct Specification of RIF-BLD Semantics.

The semantics of the RIF Basic Logic Dialect is defined by specialization from the semantics of the semantic framework for logic dialects of RIF. Section Semantics of a RIF Dialect as a Specialization of the RIF Framework in [RIF-FLD] lists the

parameters of the semantic framework that can be specialized. Thus, for RIF-BLD, we need to look at the following parameters:

- *The effect of the syntax*.

  RIF-BLD does not support negation. This is the only obvious simplification with respect to RIF-FLD as far as the semantics is concerned. The restrictions on the signatures of symbols in RIF-BLD do not affect the semantics in a significant way.

- *Truth values*.

  The set ***TV*** of truth values in RIF-BLD consists of two values, **t** and **f**, such that **f** $<_t$ **t**.

- *Datatypes*.

  RIF-BLD supports the datatypes listed in Section Datatypes of [RIF-DTB].

- *Logical entailment*.

  Recall that logical entailment in RIF-FLD is defined with respect to an unspecified set of intended semantic structures and that dialects of RIF must make this notion concrete. For RIF-BLD, this set is defined as the set of all models.

- *Import directive*.

  The semantics of the two-argument `Import` directive is given in [RIF-RDF+OWL]. The semantics of the one-argument directive is the same as in RIF-FLD.

6.3 The XML Serialization of RIF-BLD as a Specialization of RIF-FLD

Section Mapping from the RIF-FLD Presentation Syntax to the XML Syntax of [RIF-FLD] defines a mapping, χ_{fld}, from the presentation syntax of RIF-FLD to its XML serialization. When restricted to well-formed RIF-BLD formulas, χ_{fld} coincides with the BLD-to-XML mapping χ_{bld}. In this way, the XML serialization of RIF-BLD is a specialization of the RIF-FLD XML Serialization Framework defined in [RIF-FLD].

6.4 RIF-BLD Conformance as a Specialization of RIF-FLD

If T is a set of datatypes and symbol spaces and E a coherent set of external schemas for functions and predicates, then the general definition of conformance in RIF-FLD yields the notion of conformant $BLD_{T,E}$ producers and consumers.

BLD further requires *strictness*, i.e., that a conformant producer produces only the documents where T are precisely the datatypes/symbol spaces and E are the external schemas specified in [RIF-DTB], and that a conformant consumer consumes only such documents.

7 Acknowledgements

This document is the product of the Rules Interchange Format (RIF) Working Group (see below) whose members deserve recognition for their time and commitment. The editors extend special thanks to: Jos de Bruijn, Gary Hallmark, Stella Mitchell, Leora Morgenstern, Adrian Paschke, 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.

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), Changhai Ke (IBM), Chris Welty (IBM), Christian de Sainte Marie (IBM), Dave Reynolds (HP), Gary Hallmark (ORACLE), Harold Boley (NRC), Hassan Aït-Kaci (IBM), Jos de Bruijn (FUB), Leora Morgenstern (IBM), Michael Kifer (Stony Brook), Mike Dean (BBN), Sandro Hawke (W3C/MIT), and Stella Mitchell (IBM). We would also like to thank past members of the working group, Allen Ginsberg, David Hirtle, Igor Mozetic, and Paula-Lavinia Patranjan.

8 References

8.1 Normative References

[RDF-CONCEPTS]
Resource Description Framework (RDF): Concepts and Abstract Syntax, Klyne G., Carroll J. (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/.

[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.

[RFC-3987]
RFC 3987 - Internationalized Resource Identifiers (IRIs), M. Duerst and M. Suignard, IETF, January 2005. This document is http://www.ietf.org/rfc/rfc3987.txt.

[RIF-Core]
RIF Core Dialect Harold Boley, Gary Hallmark, Michael Kifer, Adrian Paschke, Axel Polleres, Dave Reynolds, eds. W3C Editor's Draft, 11 May 2010, http://www.w3.org/2005/rules/wg/draft/ED-rif-core-20100511/. Latest version available at http://www.w3.org/2005/rules/wg/draft/rif-core/.

[RIF-DTB]
RIF Datatypes and Built-Ins 1.0 Axel Polleres, Harold Boley, Michael Kifer, eds. W3C Editor's Draft, 11 May 2010, http://www.w3.org/2005/rules/wg/draft/ED-rif-dtb-20100511/. Latest version available at http://www.w3.org/2005/rules/wg/draft/rif-dtb/.

[RIF-FLD]
RIF Framework for Logic Dialects Harold Boley, Michael Kifer, eds. W3C Editor's Draft, 11 May 2010, http://www.w3.org/2005/rules/wg/draft/ED-rif-fld-20100511/. Latest version available at http://www.w3.org/2005/rules/wg/draft/rif-fld/.

[RIF-PRD]
RIF Production Rule Dialect Christian de Sainte Marie, Gary Hallmark, Adrian Paschke, eds. W3C Editor's Draft, 11 May 2010, http://www.w3.org/2005/rules/wg/draft/ED-rif-prd-20100511/. Latest version available at http://www.w3.org/2005/rules/wg/draft/rif-prd/.

[RIF-RDF+OWL]
RIF RDF and OWL Compatibility Jos de Bruijn, editor. W3C Editor's Draft, 11 May 2010, http://www.w3.org/2005/rules/wg/draft/ED-rif-rdf-owl-20100511/. Latest version available at http://www.w3.org/2005/rules/wg/draft/rif-rdf-owl/.

[XML1.0]
Extensible Markup Language (XML) 1.0 (Fourth Edition), W3C Recommendation, World Wide Web Consortium, 16 August 2006, edited in place 29 September 2006. This version is http://www.w3.org/TR/2006/REC-xml-20060816/.

[XML-Base]
XML Base, W3C Recommendation, World Wide Web Consortium, 27 June 2001. This version is http://www.w3.org/TR/2001/REC-xmlbase-20010627/. The latest version is available at http://www.w3.org/TR/xmlbase/.

[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 Candidate Recommendation, 30 April 2009, http://www.w3.org/TR/2009/CR-xmlschema11-2-20090430/. Latest version available as http://www.w3.org/TR/xmlschema11-2/.

8.2 Informational References

[ANF01]
Normal Form Conventions for XML Representations of Structured Data, Henry S. Thompson. October 2001. Available at http://www.ltg.ed.ac.uk/~ht/normalForms.html.

[AG08]
The Expressive Power of SPARQL, Renzo Angles and Claudio Gutierrez. *International Semantic Web Conference 2008*: 114-129

[AP07]
From SPARQL to rules (and back), Axel Polleres. *WWW 2007*: 787-796

[CG]
Information Processing, John Sowa, in *Mind and Machine.* JReading, MA: Addison-Wesley Publ., 1984.

[CL73]
Symbolic Logic and Mechanical Theorem Proving, C.L. Chang and R.C.T. Lee. Academic Press, 1973.

[CL07]
ISO/IEC 24707:2007. The ISO Common Logic Standard. Available through http://common-logic.org

[CURIE]
CURIE Syntax 1.0: A syntax for expressing Compact URIs, Mark Birbeck, Shane McCarron. W3C Candidate Recommendation 16 January 2009. Available at http://www.w3.org/TR/curie/.

[Enderton01]
A Mathematical Introduction to Logic, Second Edition, H. B. Enderton. Academic Press, 2001.

[KIF]
Knowledge Interchange Format, M. Genesereth, et al. 1998. Available at http://logic.stanford.edu/kif/dpans.html

[KLW95]
Logical foundations of object-oriented and frame-based languages, M. Kifer, G. Lausen, J. Wu. Journal of ACM, July 1995, pp. 741--843.

[Mendelson97]
Introduction to Mathematical Logic, Fourth Edition, E. Mendelson. Chapman & Hall, 1997.

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

[RDFSYN04]
RDF/XML Syntax Specification (Revised), Dave 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/.

[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/.

[Steele90]
Common LISP: The Language, Second Edition, G. L. Steele Jr. Digital Press, 1990.

[TRT03]
Object-Oriented RuleML: User-Level Roles, URI-Grounded Clauses, and Order-Sorted Terms, H. Boley. Springer LNCS 2876, Oct. 2003, pp. 1-16. Preprint at http://iit-iti.nrc-cnrc.gc.ca/publications/nrc-46502_e.html.

[URD08]
Uncertainty Treatment in the Rule Interchange Format: From Encoding to Extension, J. Zhao, H. Boley. 4th Int'l Workshop on Uncertainty Reasoning for the Semantic Web (URSW 2008). Available at http://c4i.gmu.edu/ursw/2008/papers/URSW2008_F9_ZhaoBoley.pdf.

[vEK76]
The semantics of predicate logic as a programming language, M. van Emden and R. Kowalski. Journal of the ACM 23 (1976), pp. 733-742.

9 Appendix: XML Schema for RIF-BLD

The **namespace** of RIF is "http://www.w3.org/2007/rif#".

XML schemas for the RIF-BLD sublanguages are defined below and are also available at http://www.w3.org/2010/rif-schema/bld/ with additional examples.

9.1 Condition Language

```
<?xml version="1.0" encoding="UTF-8"?>

 <xs:schema
```

```
 xmlns:xs="http://www.w3.org/2001/XMLSchema"
 xmlns:xml="http://www.w3.org/XML/1998/namespace"
 xmlns="http://www.w3.org/2007/rif#"
 targetNamespace="http://www.w3.org/2007/rif#"
 elementFormDefault="qualified"
 version="Id: BLDCond.xsd, v. 1.6, 2010-05-08, dhirtle/hboley">

<xs:import namespace='http://www.w3.org/XML/1998/namespace'
           schemaLocation='http://www.w3.org/2001/xml.xsd'/>

 <xs:annotation>
   <xs:documentation>
   This is the XML schema for the Condition Language as defined by
   the Last Call Draft of the RIF Basic Logic Dialect.

   The schema is based on the following EBNF for the RIF-BLD Condition Lan

 FORMULA        ::= IRIMETA? 'And' '(' FORMULA* ')' |
                    IRIMETA? 'Or' '(' FORMULA* ')' |
                    IRIMETA? 'Exists' Var+ '(' FORMULA ')' |
                    ATOMIC |
                    IRIMETA? 'External' '(' Atom ')'
 ATOMIC         ::= IRIMETA? (Atom | Equal | Member | Subclass | Frame)
 Atom           ::= UNITERM
 UNITERM        ::= Const '(' (TERM* | (Name '->' TERM)*) ')'
 Equal          ::= TERM '=' TERM
 Member         ::= TERM '#' TERM
 Subclass       ::= TERM '##' TERM
 Frame          ::= TERM '[' (TERM '->' TERM)* ']'
 TERM           ::= IRIMETA? (Const | Var | Expr | List | 'External' '(' E
 Expr           ::= UNITERM
 List           ::= 'List' '(' TERM* ')' | 'List' '(' TERM+ '|' TERM ')'
 Const          ::= '"' UNICODESTRING '"^^' SYMSPACE | CONSTSHORT
 Var            ::= '?' Name
 Name           ::= NCName | '"' UNICODESTRING '"'
 SYMSPACE       ::= ANGLEBRACKIRI | CURIE

 IRIMETA        ::= '(*' IRICONST? (Frame | 'And' '(' Frame* ')')? '*)'

   </xs:documentation>
 </xs:annotation>

 <xs:group name="FORMULA">
   <!--
 FORMULA        ::= IRIMETA? 'And' '(' FORMULA* ')' |
                    IRIMETA? 'Or' '(' FORMULA* ')' |
                    IRIMETA? 'Exists' Var+ '(' FORMULA ')' |
                    ATOMIC |
```

```
                    IRIMETA? 'External' '(' Atom ')'
  -->
  <xs:choice>
    <xs:element ref="And"/>
    <xs:element ref="Or"/>
    <xs:element ref="Exists"/>
    <xs:group ref="ATOMIC"/>
    <xs:element name="External" type="External-FORMULA.type"/>
  </xs:choice>
</xs:group>

<xs:complexType name="External-FORMULA.type">
  <!-- sensitive to FORMULA (Atom) context-->
  <xs:sequence>
    <xs:group ref="IRIMETA" minOccurs="0" maxOccurs="1"/>
    <xs:element name="content" type="content-FORMULA.type"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="content-FORMULA.type">
  <!-- sensitive to FORMULA (Atom) context-->
  <xs:sequence>
    <xs:element ref="Atom"/>
  </xs:sequence>
</xs:complexType>

<xs:element name="And">
  <xs:complexType>
    <xs:sequence>
      <xs:group ref="IRIMETA" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="formula" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="Or">
  <xs:complexType>
    <xs:sequence>
      <xs:group ref="IRIMETA" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="formula" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="Exists">
  <xs:complexType>
    <xs:sequence>
      <xs:group ref="IRIMETA" minOccurs="0" maxOccurs="1"/>
```

```
      <xs:element ref="declare" minOccurs="1" maxOccurs="unbounded"/>
      <xs:element ref="formula"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="formula">
  <xs:complexType>
    <xs:sequence>
      <xs:group ref="FORMULA"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="declare">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="Var"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:group name="ATOMIC">
  <!--
ATOMIC         ::= IRIMETA? (Atom | Equal | Member | Subclass | Frame)
  -->
  <xs:choice>
    <xs:element ref="Atom"/>
    <xs:element ref="Equal"/>
    <xs:element ref="Member"/>
    <xs:element ref="Subclass"/>
    <xs:element ref="Frame"/>
  </xs:choice>
</xs:group>

<xs:element name="Atom">
  <!--
Atom           ::= UNITERM
  -->
  <xs:complexType>
    <xs:sequence>
      <xs:group ref="UNITERM"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:group name="UNITERM">
  <!--
```

```
UNITERM        ::= Const '(' (TERM* | (Name '->' TERM)*) ')'
  -->
  <xs:sequence>
    <xs:group ref="IRIMETA" minOccurs="0" maxOccurs="1"/>
    <xs:element ref="op"/>
    <xs:choice>
      <xs:element ref="args" minOccurs="0" maxOccurs="1"/>
      <xs:element name="slot" type="slot-UNITERM.type" minOccurs="0" maxO
    </xs:choice>
  </xs:sequence>
</xs:group>

<xs:element name="op">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="Const"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="args">
  <xs:complexType>
    <xs:sequence>
      <xs:group ref="TERM" minOccurs="1" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="ordered" type="xs:string" fixed="yes"/>
  </xs:complexType>
</xs:element>

<xs:complexType name="slot-UNITERM.type">
  <!-- sensitive to UNITERM (Name) context-->
  <xs:sequence>
    <xs:element ref="Name"/>
    <xs:group ref="TERM"/>
  </xs:sequence>
  <xs:attribute name="ordered" type="xs:string" fixed="yes"/>
</xs:complexType>

<xs:element name="Equal">
  <!--
Equal          ::= TERM '=' TERM
  -->
  <xs:complexType>
    <xs:sequence>
      <xs:group ref="IRIMETA" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="left"/>
      <xs:element ref="right"/>
    </xs:sequence>
```

```
  </xs:complexType>
</xs:element>

<xs:element name="left">
  <xs:complexType>
    <xs:sequence>
      <xs:group ref="TERM"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="right">
  <xs:complexType>
    <xs:sequence>
      <xs:group ref="TERM"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="Member">
  <!--
Member          ::= TERM '#' TERM
  -->
  <xs:complexType>
    <xs:sequence>
      <xs:group ref="IRIMETA" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="instance"/>
      <xs:element ref="class"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="Subclass">
  <!--
Subclass        ::= TERM '##' TERM
  -->
  <xs:complexType>
    <xs:sequence>
      <xs:group ref="IRIMETA" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="sub"/>
      <xs:element ref="super"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="instance">
  <xs:complexType>
    <xs:sequence>
```

```
      <xs:group ref="TERM"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="class">
  <xs:complexType>
    <xs:sequence>
      <xs:group ref="TERM"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="sub">
  <xs:complexType>
    <xs:sequence>
      <xs:group ref="TERM"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="super">
  <xs:complexType>
    <xs:sequence>
      <xs:group ref="TERM"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="Frame">
  <!--
Frame          ::= TERM '[' (TERM '->' TERM)* ']'
  -->
  <xs:complexType>
    <xs:sequence>
      <xs:group ref="IRIMETA" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="object"/>
      <xs:element name="slot" type="slot-Frame.type" minOccurs="0" maxOcc
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="object">
  <xs:complexType>
    <xs:sequence>
      <xs:group ref="TERM"/>
    </xs:sequence>
  </xs:complexType>
```

```
</xs:element>

<xs:complexType name="slot-Frame.type">
  <!-- sensitive to Frame (TERM) context-->
  <xs:sequence>
    <xs:group ref="TERM"/>
    <xs:group ref="TERM"/>
  </xs:sequence>
  <xs:attribute name="ordered" type="xs:string" fixed="yes"/>
</xs:complexType>

<xs:group name="TERM">
  <!--
TERM           ::= IRIMETA? (Const | Var | Expr | List | 'External' '(' E
  -->
    <xs:choice>
       <xs:element ref="Const"/>
       <xs:element ref="Var"/>
       <xs:element ref="Expr"/>
       <xs:element ref="List"/>
       <xs:element name="External" type="External-TERM.type"/>
    </xs:choice>
</xs:group>

<xs:element name="List">
  <!--
List           ::= 'List' '(' TERM* ')' | 'List' '(' TERM+ '|' TERM ')'
           rewritten as
List           ::= 'List' '(' LISTELEMENTS? ')'
  -->
  <xs:complexType>
    <xs:sequence>
      <xs:group ref="IRIMETA" minOccurs="0" maxOccurs="1"/>
      <xs:group ref="LISTELEMENTS" minOccurs="0" maxOccurs="1"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:group name="LISTELEMENTS">
  <!--
LISTELEMENTS   ::= TERM+ ('|' TERM)?
  -->
  <xs:sequence>
    <xs:element ref="items"/>
    <xs:element ref="rest" minOccurs="0" maxOccurs="1"/>
  </xs:sequence>
</xs:group>
```

```
<xs:element name="items">
  <xs:complexType>
    <xs:sequence>
      <xs:group ref="TERM" minOccurs="1" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="ordered" type="xs:string" fixed="yes"/>
  </xs:complexType>
</xs:element>

<xs:element name="rest">
  <xs:complexType>
    <xs:sequence>
      <xs:group ref="TERM"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:complexType name="External-TERM.type">
  <!-- sensitive to TERM (Expr) context-->
  <xs:sequence>
    <xs:group ref="IRIMETA" minOccurs="0" maxOccurs="1"/>
    <xs:element name="content" type="content-TERM.type"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="content-TERM.type">
  <!-- sensitive to TERM (Expr) context-->
  <xs:sequence>
    <xs:element ref="Expr"/>
  </xs:sequence>
</xs:complexType>

<xs:element name="Expr">
  <!--
Expr           ::= UNITERM
  -->
  <xs:complexType>
    <xs:sequence>
      <xs:group ref="UNITERM"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="Const">
  <!--
Const          ::= '"' UNICODESTRING '"^^' SYMSPACE | CONSTSHORT
  -->
  <xs:complexType mixed="true">
```

```
    <xs:sequence>
      <xs:group ref="IRIMETA" minOccurs="0" maxOccurs="1"/>
    </xs:sequence>
    <xs:attribute name="type" type="xs:anyURI" use="required"/>
    <xs:attribute ref="xml:lang"/>
  </xs:complexType>
</xs:element>

<xs:element name="Name" type="xs:string">
  <!--
Name           ::= NCName | '"' UNICODESTRING '"'
... i.e., 'Name' stands for either the NCName string or the UNICODESTRING
  -->
</xs:element>

<xs:element name="Var">
  <!--
Var            ::= '?' Name
  -->
  <xs:complexType mixed="true">
    <xs:sequence>
      <xs:group ref="IRIMETA" minOccurs="0" maxOccurs="1"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:group name="IRIMETA">
  <!--
IRIMETA   ::= '(*' IRICONST? (Frame | 'And' '(' Frame* ')')? '*)'
  -->
  <xs:sequence>
    <xs:element ref="id" minOccurs="0" maxOccurs="1"/>
    <xs:element ref="meta" minOccurs="0" maxOccurs="1"/>
  </xs:sequence>
</xs:group>

<xs:element name="id">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="Const" type="IRICONST.type"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="meta">
  <xs:complexType>
   <xs:choice>
     <xs:element ref="Frame"/>
```

```
      <xs:element name="And" type="And-meta.type"/>
    </xs:choice>
  </xs:complexType>
 </xs:element>

 <xs:complexType name="And-meta.type">
 <!-- sensitive to meta (Frame) context-->
   <xs:sequence>
     <xs:element name="formula" type="formula-meta.type" minOccurs="0" max
   </xs:sequence>
 </xs:complexType>

 <xs:complexType name="formula-meta.type">
   <!-- sensitive to meta (Frame) context-->
   <xs:sequence>
     <xs:element ref="Frame"/>
   </xs:sequence>
 </xs:complexType>

 <xs:complexType name="IRICONST.type" mixed="true">
   <!-- sensitive to location/id context-->
   <xs:sequence/>
   <xs:attribute name="type" type="xs:anyURI" use="required" fixed="http:/
 </xs:complexType>

</xs:schema>
```

9.2 Rule Language

```
<?xml version="1.0" encoding="UTF-8"?>

 <xs:schema
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:xml="http://www.w3.org/XML/1998/namespace"
  xmlns="http://www.w3.org/2007/rif#"
  targetNamespace="http://www.w3.org/2007/rif#"
  elementFormDefault="qualified"
  version="Id: BLDRule.xsd, v. 1.6, 2010-02-02, dhirtle/hboley">

  <xs:annotation>
    <xs:documentation>
    This is the XML schema for the Rule Language as defined by
    the Last Call Draft of the RIF Basic Logic Dialect.

    The schema is based on the following EBNF for the RIF-BLD Rule Language

  Document  ::= IRIMETA? 'Document' '(' Base? Prefix* Import* Group? ')'
  Base      ::= 'Base' '(' ANGLEBRACKIRI ')'
```

```
Prefix    ::= 'Prefix' '(' NCName ANGLEBRACKIRI ')'
Import    ::= IRIMETA? 'Import' '(' LOCATOR PROFILE? ')'
Group     ::= IRIMETA? 'Group' '(' (RULE | Group)* ')'
RULE      ::= (IRIMETA? 'Forall' Var+ '(' CLAUSE ')') | CLAUSE
CLAUSE    ::= Implies | ATOMIC
Implies   ::= IRIMETA? (ATOMIC | 'And' '(' ATOMIC* ')') ':-' FORMULA
LOCATOR   ::= ANGLEBRACKIRI
PROFILE   ::= ANGLEBRACKIRI

  Note that this is an extension of the syntax for the RIF-BLD Condition
  </xs:documentation>
</xs:annotation>

<!-- The Rule Language includes the Condition Language from the same dire
<xs:include schemaLocation="BLDCond.xsd"/>

<xs:element name="Document">
  <!--
Document  ::= IRIMETA? 'Document' '(' Base? Prefix* Import* Group? ')'
  -->
  <xs:complexType>
    <xs:sequence>
      <xs:group ref="IRIMETA" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="directive" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element ref="payload" minOccurs="0" maxOccurs="1"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="directive">
  <!--
Base and Prefix represented directly in XML
  -->
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="Import"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="payload">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="Group"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

```
<xs:element name="Import">
  <!--
Import    ::= IRIMETA? 'Import' '(' LOCATOR PROFILE? ')'
LOCATOR   ::= ANGLEBRACKIRI
PROFILE   ::= ANGLEBRACKIRI
  -->
  <xs:complexType>
    <xs:sequence>
      <xs:group ref="IRIMETA" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="location"/>
      <xs:element ref="profile" minOccurs="0" maxOccurs="1"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="location" type="xs:anyURI"/>

<xs:element name="profile" type="xs:anyURI"/>

<xs:element name="Group">
  <!--
Group     ::= IRIMETA? 'Group' '(' (RULE | Group)* ')'
  -->
  <xs:complexType>
    <xs:sequence>
      <xs:group ref="IRIMETA" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="sentence" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="sentence">
 <xs:complexType>
   <xs:choice>
     <xs:group ref="RULE"/>
     <xs:element ref="Group"/>
   </xs:choice>
 </xs:complexType>
</xs:element>

<xs:group name="RULE">
  <!--
RULE      ::= (IRIMETA? 'Forall' Var+ '(' CLAUSE ')') | CLAUSE
  -->
  <xs:choice>
    <xs:element ref="Forall"/>
    <xs:group ref="CLAUSE"/>
  </xs:choice>
```

```
</xs:group>

<xs:element name="Forall">
  <xs:complexType>
    <xs:sequence>
      <xs:group ref="IRIMETA" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="declare" minOccurs="1" maxOccurs="unbounded"/>
      <!-- different from formula in And, Or and Exists -->
      <xs:element name="formula">
        <xs:complexType>
          <xs:group ref="CLAUSE"/>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:group name="CLAUSE">
  <!--
CLAUSE   ::= Implies | ATOMIC
  -->
  <xs:choice>
    <xs:element ref="Implies"/>
    <xs:group ref="ATOMIC"/>
  </xs:choice>
</xs:group>

<xs:element name="Implies">
  <!--
Implies   ::= IRIMETA? (ATOMIC | 'And' '(' ATOMIC* ')') ':-' FORMULA
  -->
  <xs:complexType>
    <xs:sequence>
      <xs:group ref="IRIMETA" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="if"/>
      <xs:element ref="then"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="if">
  <xs:complexType>
    <xs:sequence>
      <xs:group ref="FORMULA"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

```
  <xs:element name="then">
    <xs:complexType>
     <xs:choice>
       <xs:group ref="ATOMIC"/>
       <xs:element name="And" type="And-then.type"/>
     </xs:choice>
    </xs:complexType>
  </xs:element>

  <xs:complexType name="And-then.type">
    <!-- sensitive to then (ATOMIC) context-->
    <xs:sequence>
      <xs:element name="formula" type="formula-then.type" minOccurs="0" max
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="formula-then.type">
    <!-- sensitive to then (ATOMIC) context-->
    <xs:sequence>
      <xs:group ref="ATOMIC"/>
    </xs:sequence>
  </xs:complexType>

</xs:schema>
```

10 Appendix: Change Log (Informative)

This appendix summarizes the main changes to this document.

Changes since the draft of July 30, 2008.

- The definition of entailment in Section Logical Entailment relied on the notion of a "query document," which was not defined. The definitions in Sections Interpretation of Documents and Logical Entailment has been changed to eliminate this and related problems.
- Section EBNF Grammar for the Presentation Syntax of RIF-BLD now presents the EBNF of the entire language in one place. Previously the EBNF was given forst for the condition sublanguage and then repeated again when the entire rule language is defined.
- Symbols can now have many different arities.
- Lists have been added.
- Import: the first argument is now a character sequence that forms and IRI.
- Base, Prefix, Import: IRIs are now delimited with angle brackets.
- Numerous clarifications and explanations were added in response to the public comments.
- A number of typos were found and fixed.

Changes since the draft of July 3, 2009.

- A definedness condition was added to the definition of logical entailment.
- The bag of attribute/value pairs example was better explained.
- IRICONST was replaced with ANYURICONST in BLDRule.xsd, v. 1.5.
- A number of typos were found and fixed.
- "instance" of an external schema was replaced with "instantiation" of an external schema.

Changes since the Candidate Recommendation of October 1, 2009.

- Import's anyURIs were moved directly into location and profile.
- A number of typos was fixed and various clarifications added.
- Simplified notion of conformant BLD consumer.
- Fixed List by permitting IRIMETA and aligning syntax to Expr and Atom.