W3C Editor's Draft

RIF Basic Logic Dialect

W3C Editor's Draft 18 December 2008

This version:
http://www.w3.org/2005/rules/wg/draft/ED-rif-bld-20081218/
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-20080922/ (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May Be Superseded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/.*

Intermediate Snapshot Only (Not For Review)

The latest version of this document approved for full publication by the Working Group was the 30 July 2008 Last Call Working Draft. That document, however, does not include several HTML anchors which today's version of RIF-Core requires. This archival snapshot has those anchors, in order to allow RIF-Core to be properly reviewed, but is not itself being published for review. It should be used only to help understand the links from RIF-Core.

No Endorsement

Patents

Contents

- 1 Overview
- 2 Direct Specification of RIF-BLD Presentation Syntax
    - 2.1 Alphabet of RIF-BLD
    - 2.2 Terms
    - 2.3 Formulas
    - 2.4 RIF-BLD Annotations in the Presentation Syntax
    - 2.5 Well-formed Formulas
    - 2.6 EBNF Grammar for the Presentation Syntax of RIF-BLD
        - 2.6.1 EBNF for the Condition Language
        - 2.6.2 EBNF for the Rule Language
        - 2.6.3 EBNF for Annotations
- 3 Direct Specification of RIF-BLD Semantics
    - 3.1 Truth Values
    - 3.2 Semantic Structures

    - 3.3 RIF-BLD Annotations in the Semantics
    - 3.4 Interpretation of Non-document Formulas
    - 3.5 Interpretation of Documents
    - 3.6 Logical Entailment
- 4 XML Serialization Syntax for RIF-BLD
    - 4.1 XML for the Condition Language
    - 4.2 XML for the Rule Language
    - 4.3 Mapping from the Presentation Syntax to the XML Syntax
        - 4.3.1 Mapping of the Condition Language
        - 4.3.2 Mapping of the Rule Language
        - 4.3.3 Mapping of Annotations
- 5 Conformance Clauses
- 6 RIF-BLD as a Specialization of the RIF Framework [RIF-FLD]
    - 6.1 The Presentation Syntax of RIF-BLD as a Specialization of RIF-FLD
    - 6.2 The Semantics of RIF-BLD as a Specialization of RIF-FLD
    - 6.3 The XML Serialization of RIF-BLD as a Specialization of RIF-FLD
    - 6.4 RIF-BLD Conformance as a Specialization of RIF-FLD
- 7 Acknowledgements
- 8 References
    - 8.1 Normative References
    - 8.2 Informational References
- 9 Appendix: XML Schema for RIF-BLD
    - 9.1 Condition Language
    - 9.2 Rule Language
- 10 Appendix: RIF Media Type Registration
- 11 Appendix: Changes Since the Last Call Version (2008-07-30)

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 the syntax and semantics of 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. 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 builtins 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 consistent array of RIF rule dialects, which encompasses both logic rules -- 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, partial RIF-CL mappings will eventually 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 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 expresible 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] will be developed in other specifications by the RIF working group.

To give 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.*

The fact *Mary buys LeRif from John* can be logically derived by a *modus ponens* argument. 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 RIF-BLD Presentation Syntax as follows.

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

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

    cpt:sell(ppl:John bks:LeRif ppl:Mary)
  )
)
```

For the interchange of such rule (and fact) documents, an equivalent RIF-BLD XML Syntax is given in this specification. To formalize their meaning, a RIF-BLD Semantics is specified.

2 Direct Specification of RIF-BLD Presentation Syntax

This normative section specifies the syntax of RIF-BLD directly, without relying on [RIF-FLD]. We define both the ***presentation syntax*** (below) and an XML syntax in Section XML Serialization Syntax for RIF-BLD. The presentation syntax is normative, but is **not intended to be a concrete syntax** for RIF-BLD. It is defined in "mathematical English," a special form of English for communicating mathematical definitions, examples, etc. The presentation syntax **deliberately leaves out details** such as the delimiters of the various syntactic components, escape symbols, parenthesizing, precedence of operators, and the like. Since RIF is an interchange format, it uses XML as its concrete syntax and 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 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 and Symbol Spaces 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 abridged 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 $\mathtt{t} \in \mathtt{Const}$ or $\mathtt{t} \in \mathtt{Var}$ then `t` is a ***simple term***.
2. *Positional terms*. If $\mathtt{t} \in \mathtt{Const}$ and $\mathtt{t_1}$, ..., $\mathtt{t_n}$, $\mathtt{n}\geq\mathtt{0}$, are base terms then $\mathtt{t(t_1\ \ldots\ t_n)}$ is a ***positional term***.
3. *Terms with named arguments*. A ***term with named arguments*** is of the form $\mathtt{t(s_1{-}{>}v_1\ \ldots\ s_n{-}{>}v_n)}$, where $\mathtt{n}\geq\mathtt{0}$, $\mathtt{t} \in \mathtt{Const}$ and $\mathtt{v_1}$, ..., $\mathtt{v_n}$ are base terms and $\mathtt{s_1}$, ..., $\mathtt{s_n}$ are pairwise distinct symbols from the set `ArgNames`.

   The constant `t` here represents a predicate or a function; $\mathtt{s_1}$, ..., $\mathtt{s_n}$ represent argument names; and $\mathtt{v_1}$, ..., $\mathtt{v_n}$ represent argument values. The argument names, $\mathtt{s_1}$, ..., $\mathtt{s_n}$, 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.

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

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

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

   Such 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.

   Note that not only predicates and functions, but also frame terms can be externally defined. Therefore, external information sources can be modeled in an object-oriented way via frames. For instance, `External("http://example.com/acme"^^rif:iri["http://example.com/mycompany/president"^^rif:iri(?Year) -> ?Pres])` could be a representation of an externally defined method `"http://example.com/mycompany/president"^^rif:iri` in an external object `"http://example.com/acme"^^rif:iri`. □

   Feature At Risk #1: External frames

   Note: This feature is "at risk" and may be removed from this specification based on feedback. Please send feedback to public-rif-comments@w3.org.

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 this restriction has to do with the complexity of unification, which is used by several inference mechanisms of first-order logic.

2.3 Formulas

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

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 out of the 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 $\mathtt{And}(\varphi_1\ ...\ \varphi_n)$, 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 $\mathtt{Or}(\varphi_1\ ...\ \varphi_n)$, 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 $?V_1$, ..., $?V_n$, $n>0$, are variables then $\mathtt{Exists}\ ?V_1\ ...\ ?V_n(\varphi)$ is an *existential* formula.

   Condition formulas are intended to be used inside the premises of rules. Next we define the notion of RIF-BLD rules, sets of rules, and RIF documents.

3. *Rule implication*: $\varphi\ \mathtt{:-}\ \psi$ 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.)

   Feature At Risk #2: Equality in the rule conclusion (φ in the above)

   Note: This feature is "at risk" and may be removed from this specification based on feedback. Please send feedback to public-rif-comments@w3.org.

4. *Universal rule*: If φ is a rule implication and $?V_1$, ..., $?V_n$, $n>0$, are variables then $\mathtt{Forall}\ ?V_1\ ...\ ?V_n(\varphi)$ is a formula, called a *universal rule*. It is required that all the *free* variables in φ occur among the variables $?V_1\ ...\ ?V_n$ in the quantification part. An occurrence of a variable `?v` is *free* in φ if it is not inside a substring of the form $Q\ ?v\ (\psi)$ of φ, where Q

is a quantifier (`Forall` or `Exists`) 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 then `Forall` $?V_1 \ \ldots \ ?V_n(\varphi)$ is a formula, called a *universal fact*, provided that all the free variables in φ occur among the variables $?V_1 \ \ldots \ ?V_n$.

   Universal facts are often considered to be rules without premises (or having **true** as their premises).

6. *Group*: If φ_1, ..., φ_n are RIF-BLD rules, universal facts, variable-free rule implications, variable-free atomic formulas, *or* group formulas then `Group`$(\varphi_1 \ \ldots \ \varphi_n)$ is a *group formula*.

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

7. *Document*: An expression of the form `Document`$(directive_1 \ \ldots \ directive_n \ \Gamma)$ 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.
   - $directive_1$, ..., $directive_n$ 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 IRI [RFC-3987].

       The `Base` directive defines a syntactic shortcut for expanding relative IRIs into full IRIs, as described in Section Constants and Symbol Spaces 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 string 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 and Symbol Spaces.

     - An ***import directive*** can have one of these two forms: `Import(t)` or `Import(t p)`. Here `t` is a `rif:iri` constant and `p` is a term. The constant `t` indicates the location of another document to be imported and `p` is called the *profile of import*.

Section Direct Specification of RIF-BLD Semantics of this document defines the semantics for the directive `Import(t)` only. The two-argument directive, `Import(t 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(t 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.

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 an ***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***.

A convention is used to avoid a syntactic ambiguity in the above definition. 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 in RIF-BLD is 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]`. Generally, the convention associates each annotation to the largest term or formula it precedes.

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.

The set of all constant symbols, `Const`, is partitioned into several subsets as follows:

- A subset of individuals.

  The symbols in `Const` that belong to the primitive datatypes are required to be individuals.

- A number of subsets for predicate symbols such that there is one subset per symbol arity (defined below) for externally defined predicates and one for non-external predicates.

  Note that this implies that symbols used for external predicate names cannot be used for other predicates. Also, the definition of arity, below, implies that the arities for positional predicate symbols and for predicate symbols with named arguments are distinct even if the numbers of arguments are the same. Therefore, symbols that are used for positional predicates cannot be used for predicates with named arguments, and vice versa.

- A number of subsets of function symbols. As with predicate symbols, there are disjoint subsets for symbols with different arities; function symbols with named arguments and externally defined functions are in their own subsets. The only exception is the case of nullary symbols, which take zero arguments as in `f()`, since they are considered to be both positional and named-argument symbols.

Each predicate and function symbol that take at least one argument has precisely one ***arity***. For positional predicate and function symbols, an arity is a non-negative integer that tells how many arguments the symbol can take. For symbols that take named arguments, an arity is a set $\{s_1 \ \ldots \ s_k\}$ of argument names ($s_i \in$ `ArgNames`) that are allowed for that symbol. Nullary symbols (which take zero arguments) are said to have the arity 0.

An important point is that neither the above partitioning of constant symbols nor the arity are specified explicitly. Instead, the arity of a symbol and its type is determined by the context in which the symbol is used.

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 in an atomic subformula of the form `s(...)` with arity α then `s` occurs in the *context of a predicate symbol with arity* α.
- If `s` occurs as a function symbol in a term (not subformula) of the form `s(...)` with arity α then `s` occurs in the *context of a function symbol with arity* α.
- If `s` occurs as a predicate in an atomic subformula `External(s(...))` with arity α then `s` occurs in the *context of an external predicate symbol with arity* α.
- If `s` occurs as a function in a term (not subformula) `External(s(...))` with arity α then `s` occurs in the *context of an external function symbol with arity* α.
- 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(`*t*`)` be one of its import directives, where *t* is a `rif:iri` constant that identifies 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 instance 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 instance 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

Until now, we have used mathematical English to specify the syntax of RIF-BLD. Tool developers, however, may prefer EBNF notation, which provides a more succinct overview 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 the section on well-formedness conditions, i.e., the requirement that each symbol in RIF-BLD can occur in at most one context. 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, 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' '(' IRI ')'
Prefix         ::= 'Prefix' '(' Name IRI ')'
Import         ::= IRIMETA? 'Import' '(' IRICONST PROFILE? ')'
Group          ::= IRIMETA? 'Group' '(' (RULE | Group)* ')'
RULE           ::= (IRIMETA? 'Forall' Var+ '(' CLAUSE ')') | CLAUSE
CLAUSE         ::= Implies | ATOMIC
Implies        ::= IRIMETA? (ATOMIC | 'And' '(' ATOMIC* ')') ':-' FORMULA
PROFILE        ::= TERM
```

Condition Language:

```
FORMULA        ::= IRIMETA? 'And' '(' FORMULA* ')' |
                   IRIMETA? 'Or' '(' FORMULA* ')' |
                   IRIMETA? 'Exists' Var+ '(' FORMULA ')' |
                   ATOMIC |
                   IRIMETA? 'External' '(' Atom | Frame ')'
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 | 'External' '(' Expr ')'
Expr           ::= UNITERM
Const          ::= '"' UNICODESTRING '"^^' SYMSPACE | CONSTSHORT
Name           ::= UNICODESTRING
Var            ::= '?' 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`. RIF-BLD conditions permit only existential variables. 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`, 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, and `UNICODESTRING` is a Unicode string from the lexical space of that symbol space. `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 with named arguments. A frame term is a term composed of an object Id and a collection of attribute-value pairs. An `External(Atom)` is a call to an externally defined predicate; `External(Frame)` is a call to an externally defined frame. Likewise, `External(Expr)` is a call to an externally defined function.

Example 2 (RIF-BLD conditions).

This example shows conditions that are composed of atoms, expressions, frames, and existentials. In frame formulas variables are shown in the positions of object Ids, object properties, and property values. For brevity, we use the shortcut notation `prefix:suffix` for constant symbols, which 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`. This and other shortcuts are defined in [RIF-DTB].

```
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))

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 a definition in Section Formulas, formulas that query 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 3 (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 defined in [RIF-DTB], Section Constants and Symbol Spaces. Again, prefix directives are assumed in the preamble to the document. Then, two versions of the main part of the document are given.

```
Prefix(ppl  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(ppl:John ?item) :-
            And(cpt:perishable(?item)
                cpt:delivered(?item ?deliverydate ppl: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(ppl:John ?item ) :-
            Exists ?deliverydate ?scheduledate ?diffduration ?diffdays (
```

```
              And(cpt:perishable(?item)
                  cpt:delivered(?item ?deliverydate ppl: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, RIF-BLD formulas and terms can be prefixed with optional annotations, `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 the special case of a `Const` with the symbol space `rif:iri`, 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 4 (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 3a. The group is annotated with an IRI identifier and frame-represented Dublin Core metadata.

```
Document(
  Prefix(ppl  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(ppl:John ?item) :-
                And(cpt:perishable(?item)
                    cpt:delivered(?item ?deliverydate ppl: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(ppl: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 and Symbol Spaces.

3.1 Truth Values

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

3.2 Semantic Structures

The key concept in a model-theoretic semantics of a logic language is the notion of a *semantic structure* [Enderton01, Mendelson97]. The definition, below, is a bit more general than necessary. This is done in order to better see the connection with the semantics of the RIF framework described in [RIF-FLD].

Definition (Semantic structure). A ***semantic structure***, ***I***, is a tuple of the form $<\boldsymbol{TV}, \boldsymbol{DTS}, \boldsymbol{D}, \boldsymbol{D}_{\mathrm{ind}}, \boldsymbol{D}_{\mathrm{func}}, \boldsymbol{I}_{\mathrm{C}}, \boldsymbol{I}_{\mathrm{V}}, \boldsymbol{I}_{\mathrm{F}}, \boldsymbol{I}_{\mathrm{frame}}, \boldsymbol{I}_{\mathrm{NF}}, \boldsymbol{I}_{\mathrm{sub}}, \boldsymbol{I}_{\mathrm{isa}}, \boldsymbol{I}_{=}, \boldsymbol{I}_{\mathrm{external}}, \boldsymbol{I}_{\mathrm{truth}}>$. Here ***D*** is a non-empty set of elements called the ***domain*** of ***I***, and $\boldsymbol{D}_{\mathrm{ind}}$, $\boldsymbol{D}_{\mathrm{func}}$ are nonempty subsets of ***D***. $\boldsymbol{D}_{ind}$ is used to interpret the elements of `Const` that are individuals and $\boldsymbol{D}_{func}$ is used to interpret the elements of `Const` that are function symbols. As before, `Const` denotes the set of all constant symbols and `Var` the set of all variable symbols. ***TV*** denotes the set of truth values that the semantic structure uses and ***DTS*** is a set of identifiers for primitive 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` ∈ `Const`, is an *individual* then it is required that $\boldsymbol{I}_C$(`c`) ∈ $\boldsymbol{D}_{ind}$.
   - If `c` ∈ `Const`, is a *function symbol* (positional or with named arguments) then it is required that $\boldsymbol{I}_C$(`c`) ∈ $\boldsymbol{D}_{func}$.
2. $\boldsymbol{I}_V$ maps `Var` to $\boldsymbol{D}_{ind}$.

   This mapping interprets variable symbols.

3. $\boldsymbol{I}_F$ maps ***D*** to functions $\boldsymbol{D}^*_{ind} \rightarrow \boldsymbol{D}$ (here $\boldsymbol{D}^*_{ind}$ is a set of all sequences of any finite length over the domain $\boldsymbol{D}_{ind}$).

   This mapping interprets positional terms. In addition:

   - If `d` ∈ $\boldsymbol{D}_{func}$ then $\boldsymbol{I}_F$(`d`) must be a function $\boldsymbol{D}^*_{ind} \rightarrow \boldsymbol{D}_{ind}$.
   - This means that when a function symbol is applied to arguments that are individual objects then the result is also an individual object.
4. $\boldsymbol{I}_{NF}$ maps ***D*** to the set of total functions of the form `SetOfFiniteSets(ArgNames` × $\boldsymbol{D}_{ind}$) → ***D***.

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

   - If `d` ∈ $\boldsymbol{D}_{func}$ then $\boldsymbol{I}_{NF}$(`d`) must be a function `SetOfFiniteSets(ArgNames` × $\boldsymbol{D}_{ind}$) → $\boldsymbol{D}_{ind}$.
   - This is analogous to the interpretation of positional terms with two differences:
     - Each pair <`s`,`v`> ∈ `ArgNames` × $\boldsymbol{D}_{ind}$ 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. $\boldsymbol{I}_{frame}$ maps $\boldsymbol{D}_{ind}$ to total functions of the form `SetOfFiniteBags`($\boldsymbol{D}_{ind}$ × $\boldsymbol{D}_{ind}$) → ***D***.

This mapping interprets frame terms. An argument, $d \in \boldsymbol{D}_{\text{ind}}$, to $\boldsymbol{I}_{\text{frame}}$ 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 $\boldsymbol{I}_{\text{frame}}$ 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: `o[a->b a->b]`. 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` and `?B`, `?D` with `b`. (We shall see later that `o[a->b a->b]` is equivalent to `o[a->b]`.)

6. $\boldsymbol{I}_{\text{sub}}$ gives meaning to the subclass relationship. It is a mapping of the form $\boldsymbol{D}_{\text{ind}} \times \boldsymbol{D}_{\text{ind}} \rightarrow \boldsymbol{D}$.

   The operator `##` is required to be transitive, i.e., `c1 ## c2` and `c2 ## c3` must imply `c1 ## c3`. This is ensured by a restriction in Section [Interpretation of Formulas](#).

7. $\boldsymbol{I}_{\text{isa}}$ gives meaning to class membership. It is a mapping of the form $\boldsymbol{D}_{\text{ind}} \times \boldsymbol{D}_{\text{ind}} \rightarrow \boldsymbol{D}$.

   The relationships `#` and `##` are required to have the usual property that all members of a subclass are also members of the superclass, i.e., `o # cl` and `cl ## scl` must imply `o # scl`. This is ensured by a restriction in Section [Interpretation of Formulas](#).

8. $\boldsymbol{I}_{=}$ is a mapping of the form $\boldsymbol{D}_{\text{ind}} \times \boldsymbol{D}_{\text{ind}} \rightarrow \boldsymbol{D}$.

   It gives meaning to the equality operator.

9. $\boldsymbol{I}_{\text{truth}}$ is a mapping of the form $\boldsymbol{D} \rightarrow \boldsymbol{TV}$.

   It is used to define truth valuation for formulas.

10. $\boldsymbol{I}_{\text{external}}$ is a mapping from the coherent set of schemas for externally defined functions to total functions $\boldsymbol{D}^* \rightarrow \boldsymbol{D}$. For each external schema $\sigma = (?X_1 \ \ldots \ ?X_n; \ \tau)$ in the [coherent set of external schemas](#) associated with the [language](#), $\boldsymbol{I}_{\text{external}}(\sigma)$ is a function of the form $\boldsymbol{D}^n \rightarrow \boldsymbol{D}$.

    For every external schema, σ, associated with the language, $\boldsymbol{I}_{\text{external}}(\sigma)$ 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, $\boldsymbol{I}_{\text{external}}(\sigma)$ is specified in [[RIF-DTB](#)] so that:

    - If σ is a schema of a built-in function then $\boldsymbol{I}_{\text{external}}(\sigma)$ must be the function defined in the aforesaid document.

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

For convenience, we also define the following mapping $\boldsymbol{I}$ from terms to $\boldsymbol{D}$:

- $\boldsymbol{I}(\mathtt{k}) = \boldsymbol{I}_{C}(\mathtt{k})$, if `k` is a symbol in `Const`
- $\boldsymbol{I}(\mathtt{?v}) = \boldsymbol{I}_{V}(\mathtt{?v})$, if `?v` is a variable in `Var`
- $\boldsymbol{I}(\mathtt{f(t_1\ ...\ t_n)}) = \boldsymbol{I}_{F}(\boldsymbol{I}(\mathtt{f}))(\boldsymbol{I}(\mathtt{t_1}),...,\boldsymbol{I}(\mathtt{t_n}))$
- $\boldsymbol{I}(\mathtt{f(s_1\text{->}v_1\ ...\ s_n\text{->}v_n)}) = \boldsymbol{I}_{NF}(\boldsymbol{I}(\mathtt{f}))(\{<\mathtt{s_1},\boldsymbol{I}(\mathtt{v_1})>,...,<\mathtt{s_n},\boldsymbol{I}(\mathtt{v_n})>\})$

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

- $\boldsymbol{I}(\mathtt{o[a_1\text{->}v_1\ ...\ a_k\text{->}v_k]}) = \boldsymbol{I}_{\mathrm{frame}}(\boldsymbol{I}(\mathtt{o}))(\{<\boldsymbol{I}(\mathtt{a_1}),\boldsymbol{I}(\mathtt{v_1})>,\ ...,\ <\boldsymbol{I}(\mathtt{a_n}),\boldsymbol{I}(\mathtt{v_n})>\})$

  Here {...} denotes a bag of attribute/value pairs. Jumping ahead, we note that duplicate elements in such a bag do not affect the value of $\boldsymbol{I}_{\mathrm{frame}}(\boldsymbol{I}(\mathtt{o}))$ -- see Section Interpretation of Non-document Formulas. For instance, $\boldsymbol{I}(\mathtt{o[a\text{->}b\ a\text{->}b]}) = \boldsymbol{I}(\mathtt{o[a\text{->}b]})$.

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

  Note that, by definition, `External(t)` is well-formed only if `t` is an instance of an external schema. Furthermore, by the definition of coherent sets of external schemas, `t` can be an instance of at most one such schema, so $\boldsymbol{I}(\mathtt{External(t)})$ is well-defined.

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

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

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

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

RIF-BLD does not impose restrictions on $\mathbf{I}_C$ 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.

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

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` ∈ ***D***, if TVal_I(`c1 ## c2`) = TVal_I(`c2 ## c3`) = **t** then TVal_I(`c1 ## c3`) = **t**.

5. *Membership*: TVal_I(`o # cl`) = $\boldsymbol{I}_{\mathrm{truth}}$(***I***(`o # cl`)).

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

   - For all `o`, `cl`, `scl` ∈ ***D***, if TVal_I(`o # cl`) = TVal_I(`cl ## scl`) = **t** then TVal_I(`o # scl`) = **t**.

6. *Frame*: TVal_I(`o[`a_1`->`v_1 ... a_k`->`v_k`]`) = $\boldsymbol{I}_{\mathrm{truth}}$(***I***(`o[`$a_1$`->`$v_1$ ... a_k`->`v_k`]`)).

   Since the bag of attribute/value pairs represents the conjunctions of all the 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)`) = $\boldsymbol{I}_{\mathrm{truth}}$($\boldsymbol{I}_{\mathrm{external}}$(σ)(***I***($s_1$), ..., ***I***($s_n$))), if `t` is an atomic formula that is an instance 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 instance of an external schema. Furthermore, by the definition of coherent sets of external schemas, `t` can be an instance 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 $\mathbf{I}^*$ is a semantic structure of the form $\langle \mathbf{TV}, \mathbf{DTS}, \mathbf{D}, \mathbf{D}_{\text{ind}}, \mathbf{D}_{\text{func}}, \mathbf{I}_C, \mathbf{I}^*_V, \mathbf{I}_F, \mathbf{I}_{\text{frame}}, \mathbf{I}_{NF}, \mathbf{I}_{\text{sub}}, \mathbf{I}_{\text{isa}}, \mathbf{I}_=, \mathbf{I}_{\text{external}}, \mathbf{I}_{\text{truth}} \rangle$, which is exactly like $\mathbf{I}$, except that the mapping $\mathbf{I}^*_V$, is used instead of $\mathbf{I}_V$. $\mathbf{I}^*_V$ is defined to coincide with $\mathbf{I}_V$ on all variables except, possibly, on $?v_1,\ldots,?v_n$.

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

    - $TVal_I(\Gamma)$ = **t** if and only if $TVal_I(\varphi_1)$ = **t**, ..., $TVal_I(\varphi_n)$ = **t**.
    - $TVal_I(\Gamma)$ = **f** otherwise.

    This means that a group of rules is treated as a conjunction. ☐

3.5 Interpretation of Documents

Document formulas are interpreted using *semantic multi-structures*. Semantic multi-structures are essentially similar to regular semantic structures but, in addition, they allow to interpret `rif:local` symbols that belong to different documents differently.

Definition (Semantic multi-structure). A ***semantic multi-structure*** is a set $\{\mathbf{I}^{\varphi_1}, \ldots, \mathbf{I}^{\varphi_n}, \ldots\}$ of semantic structures ***adorned*** with *distinct* RIF-BLD formulas $\varphi_1, \ldots, \varphi_n$. These structures must be identical in all respects except that the mappings $\mathbf{I}_C^{\varphi_1}, \ldots, \mathbf{I}_C^{\varphi_n}, \ldots$ may differ on the constants in `Const` that belong to the `rif:local` symbol space. ☐

We can now define the semantics of RIF documents.

Definition (Truth valuation of document formulas). Let Δ be a document formula and let $\Delta_1, \ldots, \Delta_k$ be all the RIF-BLD document formulas that are *imported* (directly or indirectly, according to Definition Imported document) into Δ. Let $\Gamma, \Gamma_1, \ldots, \Gamma_k$ denote the respective group formulas associated with these documents. Let $\mathbf{I} = \{\mathbf{I}^{\Delta}, \mathbf{I}^{\Delta_1}, \ldots, \mathbf{I}^{\Delta_k}, \ldots\}$ be a semantic multi-structure that contains semantic structures adorned with at least the documents $\Delta, \Delta_1, \ldots, \Delta_k$. Then we define:

- $TVal_I(\Delta)$ = **t** if and only if $TVal_{I^{\Delta}}(\Gamma) = TVal_{I^{\Delta_1}}(\Gamma_1) = \ldots = TVal_{I^{\Delta_k}}(\Gamma_k)$ = **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 `a = a`, and every *TVal* function maps such a Γ_i to the truth value **t**.

For non-document formulas, we extend $TVal_I(\varphi)$ from regular semantic structures to multi-structures as follows: if ***I*** is a multi-structure that has a component structure $\boldsymbol{I}^{\varphi}$ adorned with φ then $TVal_I(\varphi) = TVal_{I^{\varphi}}(\varphi)$. Otherwise, $TVal_I(\varphi)$ is undefined.

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.

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 documents. Entailment of condition formulas provides formal underpinning to RIF-BLD queries.

Definition (Models). A multi-structure ***I*** is a ***model*** of a formula, φ, written as $\boldsymbol{I} \models \varphi$, iff $TVal_I(\varphi) = \mathbf{t}$. 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, ***I***, for which both $TVal_I(\varphi)$ and $TVal_I(\psi)$ are defined, $\boldsymbol{I} \models \varphi$ implies $\boldsymbol{I} \models \psi$. ☐

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 `Δ |= "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.

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 (Conformant BLD document in XML syntax). A ***conformant*** BLD document in the XML syntax is a valid BLD document in the 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 χ_{bld} defined in Section Mapping from the Presentation Syntax to the XML Syntax. ☐

The XML serialization for RIF-BLD is *alternating* or *fully striped* [ANF01]. A fully 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 fixed 'ordered' attr
- 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 fixed 'ordered' attribute,
- Equal      (prefix version of term equation '=')
- Expr       (expression formula, positional or with named arguments)
- left       (Equal left-hand side role)
- right      (Equal right-hand side role)
- Const      (individual, function, or predicate symbol, with optional 'type
- 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 `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 `<Const type="&xs;dateTime">2007-11-23T03:55:44-02:30</Const>`. RIF-BLD also uses the `ordered` attribute to indicate that the children of `args` and `slot` elements are ordered.

Example 5 (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:

```
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 6 (A RIF condition with named arguments and its XML serialization).

This example illustrates XML serialization of RIF conditions that involve terms with named arguments. As in Example 5, we assume the following prefix directives:

```
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. While there is a RIF-BLD element tag for the `Import` directive, there are none for the `Prefix` and `Base` directives: they are handled as discussed in Section Mapping of the RIF-BLD Rule Language.

```
- 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 IRICONST)
- 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.

Example 7 (Serializing a RIF-BLD document containing an annotated group).

This example shows a serialization for the document from Example 4. For convenience, the document is reproduced at the top and then is followed by its serialization.

```
Presentation syntax:

Document(
  Prefix(ppl  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(ppl:John ?item) :-
            And(cpt:perishable(?item)
                cpt:delivered(?item ?deliverydate ppl: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(ppl:Fred ?item) :- cpt:unsolicited(?item)
    )
  )
)


XML syntax:

<!DOCTYPE Document [
  <!ENTITY ppl  "http://example.com/people#">
  <!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
    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">&ppl;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">&ppl;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">&ppl;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 $\chi_{bld}(\textbf{\textit{metavar}})$ occurs in an XML pattern in the right column of a translation table, it should be understood as a recursive application of $\chi_{\mathtt{bld}}$ to the presentation syntax represented by the metavariable. The XML syntax result of such an application is substituted for the expression $\chi_{\mathtt{bld}}$(***metavar***). A sequence of terms containing metavariables with subscripts is indicated by an ellipsis. 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 $\chi_{\mathtt{bld}}$ 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 $\chi_{\mathtt{bld}}$(`Presentation`) = `XML`. Since the presentation syntax of RIF-BLD is context sensitive, the mapping must differentiate between the terms that occur in the position of the 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 the 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 ***argument***$_i$ is assumed to be instantiated to values unequal to the instantiations of named arguments ***unicodestring***$_j$ `->` ***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 (
  conjunct$_1$
  . . .
  conjunct$_n$
)
	<And>
	  <formula>$\chi_{\mathtt{bld}}$(***conjunct***$_1$)</formula>
	  . . .
	  <formula>$\chi_{\mathtt{bld}}$(***conjunct***$_n$)</formula>
	</And>

Or (
  disjunct$_1$
  . . .
  disjunct$_n$
)
	<Or>
	  <formula>$\chi_{\mathtt{bld}}$(***disjunct***$_1$)</formula>
	  . . .
	  <formula>$\chi_{\mathtt{bld}}$(***disjunct***$_n$)</formula>
	</Or>

Exists
  variable$_1$
  . . .
  variable$_n$ (
	<Exists>
	  <declare>$\chi_{\mathtt{bld}}$(***variable***$_1$)</declare>
	  . . .
	  <declare>$\chi_{\mathtt{bld}}$(***variable***$_n$)</declare>

W3C Editor's Draft

premise)	<formula>χ_{bld}(***premise***)</formula> </Exists>
External (***atomframexpr***)	<External> <content>χ_{bld}(***atomframexpr***)</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>
pred (***unicodestring***$_1$ -> ***filler***$_1$ . . . ***unicodestring***$_n$ -> ***filler***$_n$)	<Atom> <op>χ_{bld}(***pred***)</op> <slot ordered="yes"> <Name>***unicodestring***$_1$</Name> χ_{bld}(***filler***$_1$) </slot> . . . <slot ordered="yes"> <Name>***unicodestring***$_n$</Name> χ_{bld}(***filler***$_n$) </slot> </Atom>
func (***unicodestring***$_1$ -> ***filler***$_1$ . . . ***unicodestring***$_n$ -> ***filler***$_n$)	<Expr> <op>χ_{bld}(***func***)</op> <slot ordered="yes"> <Name>***unicodestring***$_1$</Name> χ_{bld}(***filler***$_1$)

```
  </slot>
   . . .
  <slot ordered="yes">
    <Name>unicodestring_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>
  <sub>χ_bld(sub)</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</C
```

```
?unicodestring
```

```
<Var>unicodestring</Var>
```

4.3.2 Mapping of the Rule Language

The `χbld` 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 Translation of RIF-BLD 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 `χbld` 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 7. 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`.

<table>
<tr><th>Presentation Syntax</th><th>XML Syntax</th></tr>
<tr><td>

```
Document(
  Import(loc1 prfl1?)
   . . .
  Import(locn prfln?)
  group
        )
```

</td><td>

```
<Document>
  <directive>
    <Import>
      <location>χbld(loc1)</location>
      <profile>χbld(prfl1)</profile>?
    </Import>
  </directive>
   . . .
  <directive>
    <Import>
      <location>χbld(locn)</location>
      <profile>χbld(prfln)</profile>?
    </Import>
  </directive>
  <payload>χbld(group)</payload>
</Document>
```

</td></tr>
<tr><td>

```
Group(
  clause1
   . . .
```

</td><td>

```
<Group>
  <sentence>χbld(clause1)</sentence>
   . . .
```

</td></tr>
</table>

clause$_n$)	`<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 Translation of RIF-BLD Condition Language and Translation of RIF-BLD 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 ***unicodestring***$_j$ `->` ***filler***$_j$.

Presentation Syntax	

`(*` ***iriconst? frameconj?*** `*)`
Typetag `(` ***e***$_1$ . . . ***e***$_n$ `)`
	`<`***Typetag***`>`
	`<id>`χ_{bld}(***iriconst***
	`<meta>`χ_{bld}(***frame***
	e$_1$' . . . ***e***$_n$'
	`</`***Typetag***`>`
	
	`where` ***e***$_1$', . . .,
	χ_{bld}(***Typetag***(***e***$_1$ . .

`(*` ***iriconst? frameconj?*** `*)` ***Quantifier variable***$_1$ . . . ***variable***$_n$ `(` ***premise*** `)`	`<`***Quantifier***`>` `<id>`χ_{bld}(***iriconst***

```
  <meta>χbld(framec
  <declare>χbld(var
  . . .
  <declare>χbld(var
  <formula>χbld(pre
</Quantifier>
```

```
(* iriconst? frameconj? *)
pred (
  argument1
  . . .
  argumentn
     )
```

```
<Atom>
  <id>χbld(iriconst
  <meta>χbld(framec
  <op>χbld(pred)</c
  <args ordered="y
    χbld(argument1)
    . . .
    χbld(argumentn)
  </args>
</Atom>
```

```
(* iriconst? frameconj? *)
func (
  argument1
  . . .
  argumentn
     )
```

```
<Expr>
  <id>χbld(iriconst
  <meta>χbld(framec
  <op>χbld(func)</c
  <args ordered="y
    χbld(argument1)
    . . .
    χbld(argumentn)
  </args>
</Expr>
```

```
(* iriconst? frameconj? *)
pred (
  unicodestring1 -> filler1
  . . .
  unicodestringn -> fillern
     )
```

```
<Atom>
  <id>χbld(iriconst
  <meta>χbld(framec
  <op>χbld(pred)</c
  <slot ordered="y
    <Name>unicodes
    χbld(filler1)
  </slot>
   . . .
  <slot ordered="y
    <Name>unicodes
    χbld(fillern)
  </slot>
</Atom>
```

```
(* iriconst? frameconj? *)
func (
  unicodestring_1 -> filler_1
  . . .
  unicodestring_n -> filler_n
     )
```

```
<Expr>
  <id>χ_bld(iriconst
  <meta>χ_bld(framec
  <op>χ_bld(func)</c
  <slot ordered="y
    <Name>unicodes
    χ_bld(filler_1)
  </slot>
   . . .
  <slot ordered="y
    <Name>unicodes
    χ_bld(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?`***` *)`
`?`***`unicodestring`***
	`<Var>`
	`<id>`χ_{bld}`(`***`iriconst`***
	`<meta>`χ_{bld}`(`***`framec`***
	`unicodestring`
	`</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 syntax and the XML syntax of RIF-BLD.

Let T be a set of datatypes that includes the datatypes specified in [RIF-DTB], and suppose E is a set of external terms 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 the datatypes used in φ are in T, and
- all the externally defined terms used in φ are in 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 $\nu(\varphi) \models_{BLD} \nu(\psi)$.

A ***conformant 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 Conformant 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 φ $|=_{BLD}$ ψ, 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 RIF-BLD consumer*** is a conformant $BLD_{T,E}$ consumer in which T consists only of the datatypes and E consists only of the externally defined terms (functions and predicates) that are required by RIF-BLD. These datatypes and externally defined terms (called built-ins) are specified in [RIF-DTB]. A conformant RIF-BLD consumer must reject all inputs that do not match the syntax of BLD. If it implements extensions, it may do so under user control -- having a "strict BLD" mode and a "run-with-extensions" mode.
- 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.

Feature At Risk #3: Strictness Requirement

Note: This feature is "at risk" and may be removed from this specification based on feedback. Please send feedback to public-rif-comments@w3.org.

The two preceding clauses are features **AT RISK**. In particular, the "strictness" requirement is under discussion.

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 the systems that do not support some of the syntax directly to still support it through syntactic transformations. For instance, disjunctions in the 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 [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. *Alphabet*.

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

2. *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. For every integer `n ≥ 0`, there are signatures
      - f_n`{(individual ... individual) ⇒ individual}` -- for n-ary function symbols,
      - p_n`{(individual ... individual) ⇒ atomic}` -- for n-ary predicates.
      - ef_n`{(individual ... individual) ⇒ individual}` -- for n-ary *external* function symbols,

  - $\mathtt{ep_{n}}$`{(individual ... individual) ⇒ atomic}` -- for n-ary *external* predicates.

  These represent function and predicate symbols of arity `n` (each of the above cases has `n individual`s as arguments inside the parentheses).

c. For every set of symbols `s1`,...,`sk` ∈ `ArgNames`, there are signatures
  - $\mathtt{f_{s1...sk}}$`{(s1->individual ... sk->individual) ⇒ individual}`
  - $\mathtt{p_{s1...sk}}$`{(s1->individual ... sk->individual) ⇒ atomic}`.
  - $\mathtt{ef_{s1...sk}}$`{(s1->individual ... sk->individual) ⇒ individual}`
  - $\mathtt{ep_{s1...sk}}$`{(s1->individual ... sk->individual) ⇒ atomic}`.

  These are signatures for terms and predicates with arguments named `s1`, ..., `sk`, respectively. The signatures $\mathtt{ef_{s1...sk}}$ and $\mathtt{ep_{s1...sk}}$ are for external symbols. In this specialization of RIF-FLD, the argument names `s1`, ..., `sk` must be pairwise distinct.

d. A symbol in `Const` can have exactly one signature, `individual`, $\mathtt{f_{n}}$, $\mathtt{p_{n}}$, $\mathtt{ef_{n}}$, $\mathtt{ep_{n}}$, where $\mathtt{n} \geq 0$, or $\mathtt{f_{s1...sk}}$, $\mathtt{p_{s1...sk}}$, $\mathtt{ef_{s1...sk}}$, or $\mathtt{ep_{s1...sk}}$, for some `s1`,...,`sk` ∈ `ArgNames`. It cannot have the signature `atomic`, since only complex terms can have such signatures. Thus, by itself a symbol cannot be a proposition in RIF-BLD, but a term of the form `p()` can be.

  Accordingly, in RIF-BLD each constant symbol can be either an individual, a function of one particular arity, a predicate of one particular arity, an externally defined function symbol of one particular arity, or an externally defined predicate symbol of one particular arity -- it is not possible for the same symbol to play more than one role.

e. The constant symbols that belong to the primitive RIF datatypes (XML Schema datatypes, `rdf:XMLLiteral`, `rdf:text`, 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`, $\mathtt{f_{n}}$, $\mathtt{p_{n}}$, $\mathtt{ef_{n}}$, or $\mathtt{ep_{n}}$, for `n = 0,1,...`; or $\mathtt{f_{s1...sk}}$, $\mathtt{p_{s1...sk}}$, $\mathtt{ef_{s1...sk}}$, or $\mathtt{ep_{s1...sk}}$, for some argument names `s1`,...,`sk` ∈ `ArgNames`.
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` and `atomic` are not reserved keywords.

3. *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. external

- Compared to RIF-FLD, terms (both positional and with named arguments) have significant restrictions in order to keep BLD relatively simple.
    - 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.
    - Likewise, a symbol cannot be an atomic formula by itself. That is, if `p` ∈ `Const` then `p` is not a well-formed atomic formula. However, `p()` can be an atomic formula.
    - Signatures permit only constant symbols to occur in the context of function or predicate names. 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.
    - In an externally defined term, `External(t)`, `t` can be only a positional, named-argument, or a frame term. Compared to RIF-FLD, this restricts `t` so that it cannot be a constant.

      Combined with the fact that in a well-formed term of the form `External(t)` the subterm `t` must be an instance of an external schema (by the definition of well-formed external terms in RIF-FLD), it follows that a predicate or a function symbol, `p`, that occurs in an external term `External(p(...))` cannot also occur as a non-external symbol.
    - If a term, `t`, is an instance of an externally defined schema from 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.

4. *Required symbol spaces*.

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

5. *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, a conjunctive or disjunctive combination of atomic formulas, or an external atomic formula. All these can optionally have 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.

      Note: This feature (Equality in the rule conclusion) is "at risk". See feature at risk #2

    - 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` directive and the `Import(loc)` directive (with one argument) can import RIF-BLD documents only. There are no BLD-specific restrictions on the two-argument directive `Import`.

Recall that negation (classical or default) is not supported by RIF-BLD in either the rule conclusion or the premise.

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 just two values, **t** and **f** such that **f** $<_t$ **t**. The order $<_t$ is total.

- *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 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 E a set of externally defined functions and predicates, then the general definition of conformance in RIF-FLD yields the notion of conformant $\mathrm{BLD}_{T,E}$ producers and consumers.

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

Note: This feature (Strictness requirement) is "at risk". See feature at risk #3

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, David Hirtle, Stella Mitchell, Leora Morgenstern, Igor Mozetic, 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 (REWERSE), Axel Polleres (DERI), Chris Welty (IBM), Christian de Sainte Marie (ILOG), Dave Reynolds (HP), Gary Hallmark (ORACLE), Harold Boley (NRC), Hassan Aït-Kaci (ILOG), Igor Mozetic (JSI), John Hall (OMG), 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 two past members of the working group, Allen Ginsberg 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 http://www.isi.edu/in-notes/rfc3066.txt.

[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-DTB]
RIF Datatypes and Built-Ins 1.0 Axel Polleres, Harold Boley, Michael Kifer, eds. W3C Working Draft, 18 December 2008, http://www.w3.org/TR/2008/WD-rif-dtb-20081218/. Latest version available at http://www.w3.org/TR/rif-dtb/.

[RIF-FLD]
RIF Framework for Logic Dialects Harold Boley, Michael Kifer, eds. W3C Working Draft, 30 July 2008, http://www.w3.org/TR/2008/WD-rif-fld-20080730/. Latest version available at http://www.w3.org/TR/rif-fld/.

[RIF-PRD]
RIF Production Rule Dialect Christian de Sainte Marie, Adrian Paschke, Gary Hallmark, eds. W3C Working Draft, 18 December 2008, http://www.w3.org/TR/2008/WD-rif-prd-20081218/. Latest version available at http://www.w3.org/TR/rif-prd/.

[RIF-RDF+OWL]
RIF RDF and OWL Compatibility Jos de Bruijn, editor. W3C Working Draft, 30 July 2008, http://www.w3.org/TR/2008/WD-rif-rdf-owl-20080730/. Latest version available at http://www.w3.org/TR/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-SCHEMA2]
XML Schema Part 2: Datatypes, W3C Recommendation, World Wide Web Consortium, 2 May 2001. This version is http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/. The latest version is available at http://www.w3.org/TR/xmlschema-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 Working Draft 2 April 2008. 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 Working Draft, 18 December 2008, http://www.w3.org/TR/2008/WD-rif-ucr-20081218/. Latest version available at http://www.w3.org/TR/rif-ucr/.

[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.

[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 here 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="http://www.w3.org/2007/rif#"
 targetNamespace="http://www.w3.org/2007/rif#"
 elementFormDefault="qualified"
 version="Id: BLDCond.xsd, v. 1.0, 2008-07-20, dhirtle/hboley">

 <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 | Frame ')'
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 | 'External' '(' Expr ')'
Expr           ::= UNITERM
Const          ::= '"' UNICODESTRING '"^^' SYMSPACE | CONSTSHORT
Name           ::= UNICODESTRING
Var            ::= '?' 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 | Frame ')'
  -->
  <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 | Frame) 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 | Frame) context-->
  <xs:sequence>
    <xs:choice>
      <xs:element ref="Atom"/>
      <xs:element ref="Frame"/>
    </xs:choice>
  </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="0" 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 | 'External' '(' Expr ')'
  -->
    <xs:choice>
       <xs:element ref="Const"/>
       <xs:element ref="Var"/>
       <xs:element ref="Expr"/>
       <xs:element name="External" type="External-TERM.type"/>
    </xs:choice>
</xs:group>

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

```
<xs:element name="Name" type="xs:string">
  <!--
Name           ::= UNICODESTRING
  -->
</xs:element>

<xs:element name="Var">
  <!--
Var            ::= '?' UNICODESTRING
  -->
  <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"/>   <!-- type="&rif;i
    </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="http://www.w3.org/2007/rif#"
 targetNamespace="http://www.w3.org/2007/rif#"
 elementFormDefault="qualified"
 version="Id: BLDRule.xsd, v. 1.0, 2008-07-16, 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' '(' IRI ')'
 Prefix    ::= 'Prefix' '(' Name IRI ')'
 Import    ::= IRIMETA? 'Import' '(' IRICONST PROFILE? ')'
 Group     ::= IRIMETA? 'Group' '(' (RULE | Group)* ')'
 RULE      ::= (IRIMETA? 'Forall' Var+ '(' CLAUSE ')') | CLAUSE
 CLAUSE    ::= Implies | ATOMIC
 Implies   ::= IRIMETA? (ATOMIC | 'And' '(' ATOMIC* ')') ':-' FORMULA
 PROFILE   ::= TERM

   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' '(' IRICONST PROFILE? ')'
  -->
  <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">
   <xs:complexType>
     <xs:sequence>
       <xs:element name="Const" type="IRICONST.type"/>   <!-- type="&rif;i
     </xs:sequence>
   </xs:complexType>
 </xs:element>

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

 <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: RIF Media Type Registration

The anticipated RIF media type is "application/rif+xml". The draft registration for this media type (pending IETF discussion and approval by the IESG) follows.

```
Type name: application

Subtype name: rif+xml

Required parameters: none

Optional parameters: charset, as per RFC 3023 (XML Media Types)

Encoding considerations: same as RFC 3023 (XML Media Types)

Security considerations:

    Systems which consume RIF documents are potentially vulnerable
    to attack by malicious producers of RIF documents.  The
    vulnerabilities and forms of attack are similar to those of
    other Web-based formats with programming or scripting
    capabilities, such as HTML with embedded Javascript.
```

Excessive Resource Use / Denial of Service Attacks

Full and complete processing of a RIF document, even one conforming to the RIF-BLD dialect, may require unlimited CPU and memory resources. Through the use of "import", it may also require arbitrary URI dereferencing, which may consume all available network resources on the consuming system or other systems. RIF consuming systems SHOULD implement reasonable defenses against these attacks.

Exploiting Implementation Flaws

RIF is a relatively complex format, and rule engines can be extremely sophisticated, so it is likely that some RIF consuming systems will have bugs which allow specially constructed RIF documents to perform inappropriate operations. We urge RIF implementors to make systems which carefully anticipate and handle all possible inputs, including those which present syntactic or semantic errors.

External (Application) Functions

Because RIF may be extended with local, application defined datatypes and functions, arbitrary vulnerabilities may be introduced. Before being installed on systems which consume untrusted RIF documents, these external functions should be closely reviewed for their own vulnerabilities and for the vulnerabilities that may occur when they are used in unexpected combinations, like "cross-site scripting" attacks.

In addition, as this media type uses the "+xml" convention, it shares the same security considerations as other XML formats; see RFC 3023 (XML Media Types).

Interoperability considerations:

This media type is intended to be shared with other RIF dialects, to be specified in the future. Interoperation between the dialects is governed by the RIF specifications.

Published specification:

RIF Basic Logic Dialect
W3C Working Draft (Recommendation Track)
http://www.w3.org/TR/rif-bld/

```
      This media type is intended to be shared with other RIF
      dialects, to be specified in the future.

   Applications that use this media type:

      Unknown at the time of this draft.  Multiple applications are
      expected, however, before the specification reaches W3C
      Proposed Recommendation status.

   Additional information:

     Magic number(s):

         As with XML in general (See RFC 3023 (XML Media Types)),
         there is no magic number for this format.

         However, the XML namespace "http://www.w3.org/2007/rif#" will
         normally be present in the document.  It may theoretically
         be missing if the document uses XML entities in an
         obfuscatory manner.

         The hex form of that namespace will depend on the charset.
         For utf-8, the hex is: 68 74 74 70 3a 2f 2f 77 77 77 2e 77
         33 2e 6f 72.

     File extension(s):

         .rif (or .xml)

     Macintosh file type code(s):

         "TEXT" (like other XML)

   Person & email address to contact for further information:

      Sandro Hawke, sandro@w3.org.  Please send technical comments
      and questions about RIF to public-rif-comments@w3.org, a
      mailing list with a public archive at
      http://lists.w3.org/Archives/Public/public-rif-comments/

   Intended usage:

      COMMON

   Restrictions on usage:

      None
```

```
Author:

    The editor and contact for this media type registration is
    Sandro Hawke, sandro@w3.org.

Change controller:

    RIF is a product of the Rule Interchange Format (RIF) Working
    Group of the World Wide Web Consortium (W3C).  See
    http://www.w3.org/2005/rules/wg for information on the group.
    The W3C (currently acting through this working group) has
    change control over the RIF specification.



(Any other information that the author deems interesting may be added
below this line.)
```

11 Appendix: Changes Since the Last Call Version (2008-07-30)

This section summarizes the main changes to this document since the "last call" public snapshot 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.
- Numerous clarifications and explanations were added in response to the public comments.
- A number of typos were found and fixed.