Abstract

This document, developed by the Rule Interchange Format (RIF) Working Group, defines a general RIF Framework for Logic Dialects (RIF-FLD). The framework describes mechanisms for specifying the syntax and semantics of logic RIF dialects through a number of generic concepts such as signatures, symbol spaces, semantic structures, and so on. The actual dialects should specialize this framework to produce their syntaxes and semantics.

Status of this Document

May Be Superseded

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

Set of Documents

This document is being published as one of a set of 12 documents:

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

Summary of Changes

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

W3C Members Please Review By 8 January 2013

The W3C Director seeks review and feedback from W3C Advisory Committee representatives, via their review form by 8 January 2013. This will allow the Director to assess consensus and determine whether to issue this document as a W3C Edited Recommendation.

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

No Endorsement

Publication as a Proposed Edited Recommendation does not imply endorsement by the W3C Membership. This is a draft document and may be updated, replaced or obsoleted by other documents at any time. It is inappropriate to cite this document as other than work in progress.
1 Overview of RIF-FLD

The RIF Framework for Logic Dialects (RIF-FLD) is a formalism for specifying all logic dialects of RIF, including the RIF Basic Logic Dialect [RIF-BLD] and [RIF-Core] (albeit not [RIF-PRD], as the latter is not a logic-based RIF dialect). RIF-FLD is a formalism in which both syntax and semantics are described through a number of mechanisms that are commonly used for various logic languages, but are rarely brought all together. Amalgamation of several different mechanisms is required because the framework must be broad enough to accommodate several different types of logic languages and because various advanced mechanisms are needed to facilitate translation into a common framework. RIF-FLD gives precise definitions to these mechanisms, but allows well-defined aspects to vary. The design of RIF envisions that future standard logic dialects will be based on RIF-FLD. Therefore, for any RIF dialect to become a standard, its development should start as a specialization of FLD and extensions to (or, deviations from) FLD should be justified.

The framework described in this document is very general and captures most of the popular logic rule languages found in Databases, Logic Programming, and on the Semantic Web. However, it is anticipated that the needs of future dialects might stimulate further evolution of RIF-FLD. In particular, future extensions might include a logic rendering of actions as found in production and reactive rule languages. This would support designers of future RIF dialects. All logic RIF dialects should be derived from RIF-FLD by specialization, as explained in Sections Syntax of a RIF Dialect as a Specialization of RIF-FLD and Semantics of a RIF Dialect as a Specialization of RIF-FLD. In addition to specialization, to lower the barrier of entry for their intended audiences, a dialect designer may choose to also specify the syntax and semantics in a direct, but equivalent, way, which does not require familiarity with RIF-FLD. For instance, the RIF Basic Logic Dialect [RIF-BLD] is specified by specialization from RIF-FLD and also directly, without relying on the framework. Thus, the reader who is only interested in RIF-BLD can proceed directly to that document.

RIF-FLD has the following main components:

- **Syntactic framework.** This framework defines the mechanisms for specifying the formal presentation syntax of RIF logic dialects by specializing the presentation syntax of the framework. The presentation syntax is used in RIF to define the semantics of the dialects and to illustrate the main ideas with examples. This syntax is not intended to be a concrete syntax for the dialects; it leaves out details such as the delimiters of the various syntactic components, parenthesizing, precedence of operators, and the like. Since RIF is an interchange format, it uses XML as its only concrete syntax.

- **Semantic framework.** This framework describes the mechanisms that are used for specifying the models of RIF logic dialects.

- **XML serialization framework.** This framework defines the general principles that logic dialects are to use in specifying their concrete XML-based syntaxes. For each dialect, its concrete XML syntax is a derivative of the dialect’s presentation syntax. It can be seen as a serialization of that syntax.
**Syntactic framework.** The syntactic framework defines eleven types of RIF terms:

- **Constants and variables.** These terms are common to most logic languages.
- **Positional terms.** These terms are commonly used in first-order logic. RIF-FLD defines positional terms in a slightly more general way in order to enable dialects with higher-order syntax, such as HiLog [CKW93] and ReIfun [RF99].
- **Terms with named arguments.** These are like positional terms except that each argument of a term is named and the order of the arguments is immaterial. Terms with named arguments generalize the notion of rows in relational tables, where column headings correspond to argument names.
- **Lists.** These terms correspond to lists in programming, and are used in the Basic Logic Dialect. Restricted versions of these terms are used in the Core Dialect and the Production Rules Dialect.
- **Frames.** A frame term represents an assertion about an object and its properties. These terms correspond to molecules of F-logic [KLW95]. There is syntactic similarity between terms with named arguments and frames, since properties (or attributes) of an object resemble named arguments. However, the semantics of these terms are different (see Sects. Semantic Structures).
- **Classification.** These terms are used to define the subclass and class membership relationships. There are two kinds of classification terms: membership terms and subclass terms. Like frames, these terms were borrowed from F-logic [KLW95].
- **Equality.** These terms are used to equate other terms.

It should be noted that [RIF-DTB] introduces a number of built-in equality predicates for the various data types (for instance, `pred:numeric-equal` or `pred:boolean-equal`). These predicates have fixed interpretations, which coincide with the interpretation of the equality terms defined in this document when the latter are evaluated over data types. However, outside of the data types, the interpretation of the equality terms may vary and is determined by the contents of RIF documents. General use of equality terms is supported in systems such as FLORA-2 [FL2], and special cases are also allowed in ReIfun [RF99].

- **Formula terms.** These are the terms for which truth values are defined by the RIF semantic framework. Most dialects would treat such terms in a special way and will impose various restrictions on the contexts in which such terms will be allowed to occur. Some advanced dialects, however, will have fewer such restrictions, which will make it possible to rely formulas and manipulate them as objects.
- **Extensional.** These terms are used to represent built-ins and external data sources that are treated as “black boxes.”
- **Remote.** These are the terms that are used to represent aggregation functions over sets.
- **Restriction.** These terms are used to represent queries to RIF documents that are not part of the RIF document that contains these terms.

Terms are then used to define several types of RIF-FLD formulas. RIF dialects can choose to permit all or some of the aforesaid categories of terms. In addition, RIF-FLD introduces extension points, one of which allows the introduction of new kinds of terms. An extension point is a keyword that is not a syntactic construct per se, but a placeholder that is supposed to be replaced by specific syntactic constructs of an appropriate kind. RIF-FLD defines several types of extension points: symbols (NEWSYMBOL), connectives (NEWCONNECTIVE), quantifiers (NEWQUANTIFIER), aggregate functions (NEWAGGRFUNCTION), and terms (NEWTERM).

The syntactic framework also defines the following specialization mechanisms:

- **Symbol spaces.** Symbol spaces partition the set of non-logical symbols that correspond to individual constants, predicates, and functions, and each partition is then given its own semantics. A symbol space has an identifier and a lexical space, which defines the “shape” of the symbols in that symbol space. Some symbol spaces in RIF are used to identify Web entities and their lexical space consists of strings that syntactically look like internationalized resource identifiers [RFC-3987], or IRIs (e.g., `http://www.w3.org/2007/nif#in`). Other symbol spaces are used to represent the datatypes required by RIF (for example, `http://www.w3.org/2001/XMLSchema#integer`).

- **Signatures.** Signatures determine which terms and formulas are well-formed. They constitute a generalization of the notion of sorts in classical first-order logic [Enderton95]. Each nonlogical symbol (and some logical symbols, like =) has an associated signature. A signature defines, in a precise way, the syntactic contexts in which the symbol is allowed to occur.

For instance, the signature associated with a symbol `p` might allow `p` to appear in a term of the form `f(p)`, but disallow it to occur in a term in the form `p(a,b)`. The signature for `f`, on the other hand, might allow that symbol to appear in `f(p)` and `f(p,q)`, but disallow `f(p,q,r)` and `f(f)`. In this way, it is possible to control which symbols are used for predicates and which for functions, where variables can occur, and so on.

Depending on their needs, dialects can decide which symbols have which signatures.

- **Restriction.** A dialect might impose further restrictions on the form of a particular kind of term or formula. For example, variables or aggregate terms might not be allowed in certain places.

- **Extension points.** RIF dialects are required to replace extension points with zero or more specific syntactic constructs of an appropriate kind. Note that in this way extension becomes part of specialization.

**Semantic framework.** This framework defines the notion of a semantic structure (also known as interpretation in the literature [Enderton01, Mendelsohn97]). Semantic structures are used to interpret formulas and to define logical entailment. As with the syntax, this framework includes a number of mechanisms that RIF logic dialects can specialize to suit their needs. These mechanisms include:

- **Set of truth values.** RIF-FLD is designed to accommodate dialects that support reasoning with inconsistent and uncertain information. Most of the logics that are designed to deal with these situations are multi-valued. Consequently, RIF-FLD postulates that there is a set of truth values, `TV`, which includes the values `t` (true) and `f` (false) and possibly others. For example, the RIF Basic Logic Dialect (RIF-BLD) is two-valued, but other dialects can have additional truth values.

- **Semantic structures.** Semantic structures determine how the different symbols in the alphabet of a dialect are interpreted and how truth values are assigned to formulas.

- **Datatypes.** Some symbol spaces that are part of the RIF syntactic framework have fixed interpretations. For instance, symbols in the symbol space `http://www.w3.org/2001/XMLSchema#string` are always interpreted as sequences of Unicode characters, and `a ≠ b` for any pair of distinct symbols. A symbol space whose symbols have a fixed interpretation in any semantic structure is called a `datatype`.

- **Entailment.** This notion is fundamental to logic-based dialects. Given a set of formulas (e.g., facts and rules) `G`, entailment determines which other formulas necessarily follow from `G`. Entailment is the main mechanism underlying query answering in Databases, Logic Programming, and the various reasoning tasks in Description Logics.

A set of formulas `G` logically entails another formula `γ` if for every semantic structure `I` in some set `S`, if `G` is true in `I` then `γ` is also true in `I`. Almost all logics define entailment this way. The difference lies in which set `S` they use. For instance, logics that are based on the classical first-order predicate calculus, such as most Description Logics, assume that `S` is the set of all semantic structures. In contrast, most Logic
Programming languages use default negation. Accordingly, the set \( S \) contains only the so-called minimal Herbrand models \([\text{Lloyd87}]\) of \( G \) and, furthermore, only the minimal models of a special kind. See \([\text{Shoham87}]\) for a more detailed exposition of this subject.

**XML serialization framework.** This framework defines the general principles for mapping the presentation syntax of RIF-FLD to the concrete XML interchange format. This includes:

- A specification of the XML syntax for RIF-FLD, including the associated XML Schema document.
- A specification of a one-to-one mapping from the presentation syntax of RIF-FLD to its XML syntax. This mapping must map any well-formed formula of RIF-FLD to an XML instance document that is valid with respect to the aforesaid XML Schema document.

This specification is the latest draft of the RIF-FLD definition. Each RIF dialect that is derived from RIF-FLD will be described in its own document. The first such dialect, the RIF Basic Logic Dialect, is described in \([\text{RIF-BLD}]\). A core dialect, which is defined by further specializing RIF-BLD, is specified in \([\text{RIF-Core}]\).

## 2 Syntactic Framework

The next subsection explains how to derive the presentation syntax of a RIF dialect from the presentation syntax of the RIF framework. The actual syntax of the RIF framework is given in subsequent subsections.

In the (normative) subsections 2 to 9, the presentation syntax is defined using "mathematical English," a special form of English for communicating mathematical definitions, examples, etc. In the non-normative final subsection **EBNF Grammar for the Presentation Syntax of RIF-FLD**, a grammar for a superset of the presentation syntax is given using Extended Backus–Naur Form (EBNF).

### 2.1 Syntax of a RIF Dialect as a Specialization of RIF-FLD

The **presentation syntax for a RIF dialect** can be obtained from the general syntactic framework of RIF by specializing the following parameters, which are defined later in this document:

1. The alphabet of RIF-FLD can be restricted by omitting symbols; it can also be expanded by actualizing the extension points \text{NEWSYMBOL}, \text{NEWCONNECTIVE}, \text{NEWQUANTIFIER}, and \text{NEWAGGRFUNC}, i.e., by replacing them with zero or more actual symbols of the appropriate kind.
2. An assignment of signatures to each constant and variable symbol. Signatures determine which terms in the dialect are well-formed and which are not. The exact way signatures are assigned depends on the dialect. An assignment can be explicit or implicit (for instance, derived from the context in which each symbol is used).
3. The choice of the **types of terms** supported by the dialect.

   The RIF logic framework introduces the following types of terms:
   - constant
   - variable
   - positional
   - with named arguments
   - lists
   - equality
   - frame
   - class membership
   - subclass
   - aggregates
   - remote term reference
   - external
   - formulas

   A dialect might support all of these terms or just a subset. For instance, some dialects might not support terms with named arguments or frame terms or certain forms of external terms (e.g., external frames). A dialect might even support additional kinds of terms that are not listed above (for instance, typing terms of F-logic \([\text{KLW95}]\)). This is done by actualizing the extension point \text{NEWTERM}, i.e., by replacing it with zero or more new kinds of terms.

4. The choice of **symbol spaces** supported by the dialect.

   Symbol spaces determine the syntax of the constant symbols that are allowed in the dialect. All RIF dialects are expected to support certain symbol spaces (see the section **Symbol Spaces**). Dialects can also introduce additional symbol spaces, such as a symbol space to represent Skolem constants and functions.

5. The choice of the **formulas** supported by the dialect.

   RIF-FLD offers the following kinds of formula terms "out of the box":
   - Atomic
   - Conjunction
   - Disjunction
   - Symmetric negation (classical, explicit, or strong)
   - Default negation (as in logic programming)
   - Rule (as in logic programming as opposed to the classical material implication)
   - Quantification (universal and existential)
   - Remote (for querying remote RIF documents)
   - External (built-in predicates and external black-box sources of information)

   A dialect might support all of these formulas or it might impose various restrictions. For instance, the formulas allowed in the conclusion and/or premises of implications might be restricted (e.g., \([\text{RIF-BLD}]\) essentially allows Horn rules only), certain types of quantification might be...
prohibited (e.g., [RIF-BLD] disallows existential quantification in the rule head), symmetric or default negation (or both) might not be allowed (as in RIF-BLD), etc. The Core subdialect of RIF-BLD disallows equality formulas in the conclusions of rules.

More interestingly, dialects can introduce additional types of formulas by adding new connectives (e.g., classical implication or bi-implication) and quantifiers through actualizing the extension points NEWCONNECTIVE and NEWQUANTIFIER.

Note that although the presentation syntax of a RIF logic dialect is normative, since semantics is defined in terms of that syntax, the presentation syntax is not intended as a concrete syntax, and conformant systems are not required to implement it.

### 2.2 Alphabet

**Definition (Alphabet).** The alphabet of the presentation syntax of RIF-FLD consists of the following disjoint subsets of symbols:

- A countably infinite set of constant symbols Const.
- A countably infinite set of variable symbols Var.
- A countably infinite set of connective symbols.
- A countably infinite set of quantifiers.
- A finite set of aggregate symbols, which includes And, Or, Naf, Neg, ⊥, and NEWCONNECTIVE.

NEWCONNECTIVE is not an actual symbol in the alphabet, but rather a RIF-FLD extension point, which must be actualized. Dialects are expected to specialize the connectives by

- Replacing NEWCONNECTIVE with zero or more new connective symbols. Dialects cannot keep the extension point.
- Dropping zero or more of the predefined connective symbols listed above. Dialects cannot redefine the semantics of the predefined connectives, however.

A countably infinite set of quantifiers, which consists of the symbols Exists ?X<sub>1</sub>, ..., ?X<sub>n</sub> and Forall ?X<sub>1</sub>, ..., ?X<sub>n</sub>, where ?X<sub>1</sub>, ..., ?X<sub>n</sub>, n ≥ 0, are distinct variable symbols; plus the extension point, NEWQUANTIFIER, which must be actualized. Dialects are supposed to specialize this repertoire of quantifier symbols by

- Replacing NEWQUANTIFIER with zero or more new quantifier symbols. Dialects cannot keep the extension point.
- Dropping zero or more of the predefined quantifier symbols listed above. However, dialects cannot redefine the semantics of the predefined quantifiers.

In the actual presentation syntax, we will be linearizing the predefined quantifier symbols and write them as Exists ?X<sub>1</sub>, ..., ?X<sub>n</sub> and Forall ?X<sub>1</sub>, ..., ?X<sub>n</sub> instead of Exists ?X<sub>1</sub>, ..., ?X<sub>n</sub> and Forall ?X<sub>1</sub>, ..., ?X<sub>n</sub>.

Every quantifier symbol has an associated list of variables that are bound by that quantifier. For the standard quantifiers Exists ?X<sub>1</sub>, ..., ?X<sub>n</sub> and Forall ?X<sub>1</sub>, ..., ?X<sub>n</sub>, the associated list of variables is ?X<sub>1</sub>, ..., ?X<sub>n</sub>.

- The symbols =, ≠, ⊆, ⊇, External, Dialect, Base, Prefix, Import, and Module.
- The symbols for representing lists: List and OpenList.
- The symbols Group and Document.
- A countable set of aggregate symbols of the form sym ?V<sub>1</sub>, ..., ?V<sub>n</sub>, where n ≥ 0, sym is a symbol that denotes an aggregate function, and ?V<sub>1</sub>, ..., ?V<sub>n</sub> are variable symbols. The symbol ?V<sub>n</sub> is called the comprehension variable of the aggregate symbol and ?V<sub>1</sub>, ..., ?V<sub>n</sub> are grouping variables.

RIF-FLD reserves the following symbols for standard aggregate functions: Min, Max, Count, Avg, Sum, Prod, Set, and Bag. Aggregate functions also have an extension point, NEMAGGFUNC, which must be actualized. Dialects can specialize the aforesaid set of aggregate functions by

- Replacing NEMAGGFUNC with zero or more new symbols for aggregate functions. Dialects cannot keep the extension point.
- Dropping zero or more of the predefined aggregate functions listed above. However, dialects cannot redefine the semantics of the predefined aggregate functions.
- Auxiliary symbols { }, [ ], ( ), <, >, |, ?, @, and ^.
- An extension point NEWSYMBOL.

As with other extension points, this is not an actual symbol in the alphabet, but a placeholder that dialects are supposed to replace with zero or more actual new alphabet symbols.

The symbol Naf represents default negation, which is used in rule languages with logic programming and deductive database semantics. Examples of default negation include Clark's negation-as-failure [Clark97], the well-founded negation [GRS91], and stable-model negation [GL88]. The name of the symbol Naf used here comes from negation-as-failure but in RIF-FLD this can refer to any kind of default negation.

The symbol Neg represents symmetric negation (as opposed to default negation, which is asymmetric because completely different inference rules are used to derive p and Naf p). Examples of symmetric negation include classical first-order negation, explicit negation, and strong negation [APP96].

The symbols =, ≠, and # are used in formulas that define equality, class membership, and subclass relationships, respectively. The symbol -> is used in terms that have named arguments and in frame terms. The symbol External indicates that an atomic formula or a function term is defined externally (e.g., a built-in). Dialect is a directive used to indicate the dialect of a RIF document (for those dialects that require this), the symbols Base and Prefix enable abridged representations of IRIs, and the symbol Import is an import directive. The Module directive is used to connect remote terms with the actual remote RIF documents.

Finally, the symbol Document is used for specifying RIF-FLD documents and the symbol Group is used to organize RIF-FLD formulas into collections.
2.3 Symbol Spaces

Throughout this document, we will be using the following abbreviations:

- xs: stands for the XML Schema URI http://www.w3.org/2001/XMLSchema#
- rdf: stands for http://www.w3.org/2000/02/22-rdf-syntax-ns#
- pred: stands for http://www.w3.org/2007/rdf-builtin-predicates#
- rif: stands for the URI of RIF, http://www.w3.org/2007/rif#

These and other abbreviations will be used as prefixes in the compact URI-like notation (CURIE), a notation for succinct representation of IRIs [RFC-3987]. The precise meaning of this notation in RIF is defined in [RIF-DB].

The set of all constant symbols in a RIF dialect is partitioned into a number of subsets, called symbol spaces, which are used to represent XML Schema datatypes, datatypes defined in other W3C specifications, such as rdf:XMLLiteral, and to distinguish other sets of constants. All constant symbols have a syntax (and sometimes also semantics) imposed by the symbol space to which they belong.

Definition (Symbol space). A symbol space is a named subset of the set of all constants, Const. The semantic aspects of symbol spaces will be described in Section Semantic Framework. Each symbol in Const belongs to exactly one symbol space.

Each symbol space has an associated lexical space and a unique identifier. More precisely,

- The lexical space of a symbol space is a non-empty set of Unicode character strings.
- The identifier of a symbol space is a sequence of Unicode characters that form an absolute IRI [RFC-3987].
- Different symbol spaces cannot share the same identifier.

The identifiers for symbol spaces are not themselves constant symbols in RIF.

To simplify the language, we will often use symbol space identifiers to refer to the actual symbol spaces (for instance, we may use “symbol space xs:string” instead of “symbol space identified by xs:string”).

To refer to a constant in a particular RIF symbol space, we use the following presentation syntax:

```
"literal"^^symspace
```

where literal is called the lexical part of the symbol, and symspace is the identifier of the symbol space. Here literal is a sequence of Unicode characters that must be an element in the lexical space of the symbol space symspace. For instance, "1.2"^^xs:decimal and "1"^^xs:decimal are syntactically valid constants because 1.2 and 1 are members of the lexical space of the XML Schema datatype xs:decimal. On the other hand, "a+2"^^xs:decimal is not a syntactically valid symbol, since a+2 is not part of the lexical space of xs:decimal.

The set of all symbol spaces that partition Const is considered to be part of the logical language of RIF-FLD.

RIF requires that all dialects include the symbol spaces listed and described in Section Constants and Symbol Spaces of [RIF-DB] as part of their language. These symbol spaces include constants that belong to several important XML Schema datatypes, certain RDF datatypes, and constant symbols specific to RIF. The latter include the symbol spaces rif:iri and rif:local, which are used to represent internationalized resource identifiers (IRIs [RFC-3987]) and constant symbols that are not visible outside of the RIF document in which they occur, respectively. Documents that are exchanged through RIF can use additional symbol spaces (for instance, a symbol space to represent Skolem constants and functions).

We will often refer to constant symbols that come from a particular symbol space, X, as X constants. For instance, the constants in the symbol space rif:iri will be referred to as IRI constants or rif:iri constants and the constants found in the symbol space rif:local as local constants or rif:local constants.

2.4 Terms

The most basic construct of a logic language is a term. RIF-FLD supports many kinds of terms: constants, variables, the regular positional terms, plus terms with named arguments, equality, classification terms, frames, and more. The word “term” will be used to refer to any kind of term.

Definition (Term). A term can have one of the following forms:

1. Constants and variables. If t ∈ Const or t ∈ Var then t is a simple term.
2. Positional terms. If t and t₁, ..., tₙ are terms then t(t₁ ... tₙ) is a positional term.

Positional terms in RIF-FLD generalize the regular notion of a term used in first-order logic. For instance, the above definition allows variables everywhere, as in ?X(?Y ?Z(?V "12"^^xs:integer)), where ?X, ?Y, ?Z, and ?V are variables. Even ?X("abc"^^xs:string "7?" ?Z(?V "3"^^xs:integer)) is a positional term (as in HiLog [CKW93]).

3. Terms with named arguments. A term with named arguments is of the form t(s₁->v₁ ... sₙ->vₙ), where t, v₁, ..., vₙ are terms, and s₁, ..., sₙ are (not necessarily distinct) symbols from the set ArgNames.

The term t here represents a predicate or a function; s₁, ..., sₙ represent argument names; and v₁, ..., vₙ represent argument values. Terms with named arguments are like regular positional terms except that the arguments names are named and their order is immaterial. Note that a term with no arguments, like f(), is, trivially, both a positional term and a term with named arguments.


4. List terms. There are two kinds of list terms: open and closed.
A closed list has the form List(t₁ ... tₙ), where m≥0 and t₁, ..., tₙ are terms.

An open list (or a list with a tail) has the form OpenList(t₁ ... tₙ t), where m=n=0 and t₁, ..., tₙ, t are terms. Open lists are written in the presentation syntax as follows: List(t₁ ... tₙ | t).

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

A closed list of the form List() (i.e., a list in which m=0) is called the empty list.

5. Equality terms. An equality term has the form t = s, where t and s are terms.

6. Classification terms. There are two kinds of classification terms: class membership terms (or just membership terms) and subclass terms. Let t be a term.

- t#S is a membership term if t and S are terms.
- t#S#t is a subclass term if t and S are terms.

Classification terms are used to describe class hierarchies.

7. Frame terms. t[p₁?v₁ ... pₙ?vₙ] is a frame term (or simply a frame) if t, p₁, ..., pₙ, v₁, ..., vₙ, n ≥ 0, are terms.

Frame terms are used to describe properties of objects. As in the case of the terms with named arguments, the order of the properties p₁?v₁ in a frame is immaterial.

8. Externally defined terms. If t is a constant, positional term, a term with named arguments, an equality, a classification, or a frame term then External(t loc) is an externally defined term.

Such terms are 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. The loc part in an external term is intended to play the role of a locator of the source that defines the external term t. It must uniquely identify the external source. The exact form of the locator loc, the protocol that associates locators with external sources, and the type of the imported documents is left to dialects to specify. However, all dialects must support the form <IRI>, where IRI is a sequence of Unicode characters that forms an IRI.

This syntax enables very flexible representations for externally defined information sources: not only predicates and functions, but also frames, classification, and equality terms can be used. In this way, external sources can be modeled in an object-oriented way. For instance, External(“http://example.com/foobar”^^rif:iri) could be a representation for an external method “http://example.com/mycompany/president”^^rif:iri in an external object identified by the IRI “http://example.com/acme”.

Since, in most cases, external terms are expected to be based on predicates, RIF-FLD also permits a shorthand notation: if t is a positional or a named-argument term of the form p(...), then External(t) is considered to be a shorthand for External(t p*=>), where p* is the IRI corresponding to p (for instance, if p is “http://example.com/fooobar”^^rif:iri then p* is “http://example.com/fooobar”).

9. Formula term. If S is a connective or a quantifier symbol and t₁, ..., tₙ are terms then S(t₁ ... tₙ) is a formula term.

Formula terms correspond to compound formulas in logic, i.e., formulas that are constructed from atomic formulas by combining them with connectives and quantifiers. For better visual appeal, some connectives (e.g., rule implication, ¬, and default negation, Naf) may be written in infix or prefix form (e.g., a :- b and Naf a), but the above function application form is considered to be canonical.

Let φ be a formula term of the form S(t₁ ... tₙ), where S is a quantifier, and let τ₁, ..., τₙ be a list of variables bound by S. We say that all occurrences of these variables are bound in the formula term φ. In general, if t is a term and φ a formula term that occurs in t then all occurrences of the variables that are bound in φ are also said to be bound in t. The occurrences of variables in a term that are not bound are said to be free. A term that has no free occurrences of variables is closed.

10. Aggregate term. An aggregate term has the form sym ν[τ₁ ... τₙ](t), where sym ν[τ₁ ... τₙ] is an aggregate symbol, n≥0, and t is a term. For readability, we will usually write aggregate terms as sym[τ ν[τ₁ ... τₙ]](t). If n=0, we will omit the [...] part. Note that aggregates can be nested, i.e., τ can contain aggregate terms.

In addition, it is required that the variables ν, τ₁, ..., τₙ have free occurrences in t, and all occurrences of other variables in t are bound.

The comprehension variable ν and the grouping variables τ₁, ..., τₙ of the symbol sym ν[τ₁ ... τₙ] are also said to be the comprehension and grouping variables of the above aggregate term. The comprehension variable ν is considered bound by the aggregation term, but the grouping variables τ₁, ..., τₙ remain free.

As a practical convenience, dialects may allow more general terms in place of the comprehension variable, similarly to Prolog’s findall/3 built-in. In this case, sym[Term [τ₁ ... τₙ]](t) is treated as a shorthand for sym[ν](τ ν[τ₁ ... τₙ] | And(ν=Term)).

11. Remote term reference. A remote term reference (also called remote term) is a term of the form φ[r] where φ is a term; r can be a constant, variable, a positional, or a named-argument term.

Remote terms are used to query remote RIF documents, called remote modules. Here φ is the actual query and r is a reference used to identify the remote module. Remote terms should be contrasted with external terms, which are used to query external sources that are not RIF documents. Since remote terms refer to remote RIF documents, their semantics is defined by RIF-FLD. In contrast, external terms are used to query external opaque sources, which are not RIF documents. So, their semantics is opaque in RIF.

12. NewTerm. This is not a specific kind of term, but an extension point; dialects are supposed to replace it with zero or more new types of terms.

The above definitions are very general. They make no distinction between constant symbols that represent individuals, predicates, and function symbols. The same symbol can occur in multiple contexts at the same time. For instance, if p, a, and b are symbols then p(p(a) p(a p c)) is a term. Even variables and general terms are allowed to occur in the position of predicates and function symbols, so p(a) ?(v a c p) is also a term.

Furthermore, the extendible set of quantifiers and connectives allows dialects to introduce additional features, which could include modal operators, bounded quantification, rule labels, and so on. For instance, to add labels to formulas, as required by some rule languages, a dialect could introduce a new connective, Label, and formulas of the form Label(t φ). Note that RIF-FLD also supports a very general form of annotations, which can be used to assign identifiers to rules. However, annotations do not affect the semantics of RIF.
dialects, so they cannot be used to label rules in dialects where rule labels do affect the semantics. It is in those cases that RIF dialect designers might choose to introduce a special connective, like Label above.)

Frame, classification, and other terms can be freely nested, as exemplified by p(?X q[r[p(1,2)->s](d->f->g)]. Some language environments, like FLORa-2 [FLO], OO [DREO [OODE]], NuXtRE [NuXRE], and CycL [CycL] support fairly large (partially overlapping) subsets of RIF-FLD terms, but most languages support much smaller subsets. RIF dialects are expected to carve out the appropriate subsets of RIF-FLD terms, and the general form of the RIF logic framework allows a considerable degree of freedom.

Observe that the argument names of frame terms, p₁; ..., pₙ, are terms and, 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 integral part of many inference rules underlying first-order logic. We are not aware of any rule language where terms with named arguments use anything more general than what is defined here.

Dialects can restrict the contexts in which the various terms are allowed by using the mechanism of signatures. The RIF-FLD language associates a signature with each variable (both constant and variable symbols) and uses signatures to define well-formed terms. Each RIF dialect is expected to select appropriate signatures for the symbols in its alphabet, and only the terms that are well-formed according to the selected signatures are allowed in that particular dialect.

Example 1 (Terms)

- Positional term: "http://example.com/ex1"^^rif:iri(1 "http://example.com/ex2"^^rif:iri(7X 5) "abc")
- Term with named arguments: "http://example.com/Person"^^rif:iri(1 "http://example.com/John"
- Frame term: "http://example.com/Date"^^rif:iri(1 "http://example.com/spouse"^^rif:iri(7Y)
- Empty list: List()
- Closed list with variable inside: List("a"^^rif:local 7Y "c"^^rif:local)
- Open list with variables: List("a"^^rif:local 7Y "c"^^xs:string | ?Z)
- Equality term with lists inside: List(7Head | 7Tail) = List("a"^^rif:local 7Y "c"^^xs:string)
- Nested list: List("a"^^rif:local List(7X "b"^^rif:local) "c"^^rif:local)
- Classification terms
  - Membership: 7X # 7Y
  - Subclass: 7X # "http://example.com/ex1"
- External term: External(7X, 7Y) # "http://example.com/Person"
- Subclass: "http://example.com/Student"
- External term: External(7X, 7Y) # "http://example.com/Person"
- External term: External(7X, 7Y) # "http://example.com/Person"
- External term: External(7X, 7Y) # "http://example.com/Person"
- Formula terms
  - (p"^^rif:local(7X)"^^rif:local(7X) =: 7X("q"^^rif:local))
  - Forall7X,7Y(Exists7Z(7p"^^rif:local(7X)"^^rif:local(7X) ?Z))
- DROID term: "http://example.com/to-be"^^rif:iri(7X) Neg("http://example.com/to-be"^^rif:iri(7X))
- Aggregate term: avg(7Sal [7Dept]) Exists7Emp "http://example.com/salary"^^rif:local(7Emp [7Dept 7Sal])
- Remote term: 7Q[7N -> "http://example.com/salary"^^rif:iri(7I) -> 7S[7T"http://acme.foo"^^xs:anyURI

2.5 Schemas for Externally Defined Terms

This section introduces the notion of external schemas, which serve as templates for externally defined terms. These schemas determine whether externally defined terms are acceptable in a RIF dialect. Externally defined terms are RIF built-ins, which are specified in [RIF-DB], but are more general. They are designed to accommodate the ideas of procedural attachments and querying of external data sources. Because of the need to accommodate many different possibilities, the RIF logical framework supports a very general notion of an externally defined term. Such a term is not necessarily a function or a predicate – it can be a frame, a classification term, and so on.

Definition (Schema for external term). An external schema has the form (7X₁ ... 7Xₙ; τ; loc) where

- loc is the locator for an external source.
- τ is a term of one of these kinds: constant, positional, named-argument, equality, classification, frame.
- 7X₁ ... 7Xₙ is a list of all distinct variables that occur in τ

The names of the variables in an external schema are immaterial, but their order is important. For instance, (7X 7Y; ?X["foo"^^xs:string->7Y]; loc) and (7Y 7W; ?Y["foo"^^xs:string->7W]; loc) are considered to be indistinguishable, but (7X 7Y; ?X["foo"^^xs:string->7Y]; loc) and (7Y 7X; ?Y["foo"^^xs:string->7X]; loc) are viewed as different schemas.

An external term External(τ loc1) is an instantiation of an external schema (7X₁ ... 7Xₙ; τ; loc) iff loc1=loc and τ can be obtained from τ by a simultaneous substitution 7X₁/s₁ ... 7Xₙ/sₙ of the variables 7X₁ ... 7Xₙ with terms s₁ ... sₙ, respectively. Some of the terms s_i can be variables themselves. For example, External(7Z["foo"^^xs:string->f("a"^^rif:local ?P)]; loc) is an instantiation of (7X 7Y; ?X["foo"^^xs:string->7Y]; loc) by the substitution 7X/7Z 7Y/f("a"^^rif:local ?P).

Observe that a variable cannot be an instantiation of an external term, since τ in the above definition cannot be a variable. It will be seen later that this implies that a term of the form External(7X loc) is not well-formed in RIF.

The intuition behind the notion of an external schema, such as (7X 7Y; ?X["foo"^^xs:string->7Y]) <http://example.com/acme> and (7Y; pred:isTime(7Y) pred:isTime), is that ?X["foo"^^xs:string->7Y] or pred:isTime(7Y) are invocation patterns for querying external sources, and instantiations of those schemas correspond to concrete invocations. Thus, External("http://foo.bar.com"^^rif:iri["foo"^^xs:string->123"^^xs:integer]) <http://example.com/acme> and External(pred:isTime("22:33:44"^^xs:time) pred:isTime) are examples of invocations of external terms – one querying the external source identified by the IRI http://example.com/acme and the other invoking the built-in identified by the IRI pred:isTime.

Recall that one-argument externals, such as External(7) are shortcuts for two-argument externals. So, we define a one-argument external term to be an instantiation of an external schema iff its corresponding two-argument form is an instantiation of that schema.

Definition (Coherent set of external schemas). A set E of external schemas is coherent if there is no term, τ, that is an instantiation of two distinct schemas in E.

http://www.w3.org/TR/2012/PER-rif-fld-20121211/
The intuition behind this notion is to ensure that any use of an external term is associated with at most one external schema. This assumption is relied upon in the definition of the semantics of externally defined terms. Note that the coherence condition is easy to verify syntactically and that it implies that schemas like \{?X ?Y; ?X["foo"^^xsd:string->?Y]; loc\} and \{?Y ?X; ?X["foo"^^xsd:string->?Y]; loc\}, which differ only in the order of their variables, cannot be in the same coherent set.

It is important to keep in mind that external schemas are not part of the logic language in RIF, since they do not appear anywhere in RIF expressions. Instead, like signatures, which are defined below, they are best thought of as part of the grammar of the language. In particular, they will be used to determine which external terms, i.e., the terms of the form $\text{External}(t \ loc)$ are well-formed.

### 2.6 Signatures

In this section we introduce the concept of a signature, which is a key mechanism that allows RIF-FLD to control the context in which the various symbols are allowed to occur. For instance, a symbol $f$ with signature \{term\} => term, \{term\} => term can occur in terms like $f(a\ b)$, $f(f(a\ b)\ a)$, $f(f(a))$, etc., if $a$ and $b$ have signature term. But $f$ is not allowed to appear in the context $f(a\ b\ a)$ because there is no \=>-expression in the signature of $f$ to support such a context.

The above example provides intuition behind the use of signatures in RIF-FLD. Much of the development, below, is inspired by [CK95]. It should be kept in mind that signatures are not part of the logic language in RIF, since they do not appear anywhere in RIF-FLD formulas. Instead they are part of the grammar: they are used to determine which sequences of tokens are in the language and which are not. The actual way by which signatures are assigned to the symbols of the language may vary from dialect to dialect. In some dialects (for example [RIF-BLD]), this assignment is derived from the context in which each symbol occurs and no separate language for signatures is used. Other dialects may choose to assign signatures explicitly. In that case, they would require a concrete language for signatures (which would be separate from the language for specifying the logic formulas of the dialect).

**Definition (Signature name).** Let $\Sigma$ be a non-empty, partially-ordered finite or countably infinite set of symbols, called signature names. Since signatures are not part of the logic language, their names do not have to be disjoint from $\text{Const}$, $\text{Var}$, and $\text{ArgNames}$. We require that this set includes at least the following **reserved signature names**:

- $\text{atomic}$ -- used to represent the syntactic context where atomic formulas are allowed to appear.
- $\text{formula}$ -- represents the context where formulas (atomic or composite) may appear.
- $\text{=}$$\text{-connective}$-- the signature for the connectives, such as $\text{And}$ and $\text{Or}$, that can take any number of arguments.
- $\text{2-connective}$ -- the signature for the connectives, such as the rule implication connective $\text{:-}$, that take exactly two arguments.
- $\text{1-connective}$ -- the signature for the connectives that take exactly one argument. In our case, this signature will be used for the negation connectives and the quantifiers $\text{Forall}$ and $\text{Exists}$.
- $\text{=}$$\text{-}$ used for representing contexts where equality terms can appear.
- $\theta$ -- a signature name reserved for membership terms.
- $\text{##}$ -- a signature reserved for subclass terms.
- $\text{->}$ -- a signature reserved for frame terms.
- $\text{aggregate}$ -- a signature reserved for aggregate functions.
- $\text{remote}$ -- a signature reserved for the symbol $\theta$ that is used to build remote terms.
- $\text{list}$ -- a signature reserved for the symbol $\text{List}$ that is used to represent closed lists.
- $\text{openlist}$ -- a signature reserved for the symbol $\text{OpenList}$ that is used to represent open lists.

Dialects may introduce additional signature names. For instance, RIF Basic Logic Dialect [RIF-BLD] introduces the signature name individual. The partial order on $\Sigma$ is dialect-specific; it is used in the definition of well-formed terms below.

We use the symbol $<$ to represent the partial order on $\Sigma$. Informally, $\alpha < \beta$ means that terms with signature $\alpha$ can be used wherever terms with signature $\beta$ are allowed. We will write $\alpha \leq \beta$ if either $\alpha = \beta$ or $\alpha < \beta$.

**Definition (Signature).** A signature has the form $\eta(e_1, \ldots, e_n, \ldots)$ where $\eta \in \Sigma$ is the name of the signature and $(e_1, \ldots, e_n, \ldots)$ is a countable set of arrow expressions. Such a set can thus be infinite, finite, or even empty. In RIF-BLD, signatures can have at most one arrow expression. Other dialects (such as one for HiLog [CKW93] and Refun [RF99], for example) may require polymorphic symbols and thus allow signatures with more than one arrow expression in them.

An arrow expression is defined as follows:

- If $\kappa_1, \ldots, \kappa_n \in \Sigma$, na0, are signature names then $(\kappa_1 \ldots \kappa_n) \Rightarrow \kappa$ is a **positional arrow expression**.

For instance, $(\text{individual}) \Rightarrow \text{individual}$ is an arrow expression, if individual is a signature name.

- If $\kappa_1, \ldots, \kappa_n \in \Sigma$, na0, are signature names and $p_1, \ldots, p_n \in \text{ArgNames}$ are argument names then $(p_1 \Rightarrow \kappa_1 \ldots p_n \Rightarrow \kappa_n) \Rightarrow \kappa$ is an **arrow expression with named arguments**.

For instance, $(\text{arg1} \Rightarrow \text{individual}, \text{arg2} \Rightarrow \text{individual}) \Rightarrow \text{individual}$ is an arrow expression with named arguments. The order of the arguments in arrow expressions with named arguments is immaterial, so any permutation of arguments yields the same expression.

RIF dialects are always associated with sets of coherent signatures, defined next. The overall idea is that a coherent set of signatures must include all the predefined signatures (such as signatures for equality and classification terms) and the signatures included in a coherent set must not conflict with each other. For instance, two different signatures should not have identical names and if one signature is said to extend another then the arrow expressions of the supersignature should be included among the arrow expressions of the subsignature (a kind of an arrow expression "inheritance").

**Definition (Coherent signature set).** A set $\Sigma$ of signatures is coherent iff

1. $\Sigma$ contains the special signatures $\text{atomic}$ and $\text{formula}$, which represent the context of atomic formulas and more generally, composite formulas, respectively. Furthermore, it is required that $\text{atomic} < \text{formula}$.
2. $\Sigma$ contains the special signature $\text{=}$$\text{-connective}(e_1, \ldots, e_n, \ldots)$, where each $e_n$ has the form $\text{formula} \Rightarrow \text{formula}$ (the left-hand side of this signature is a sequence of $\text{symbols} \Rightarrow \text{formula}$). This signature is assigned to the connectives $\text{And}$ and $\text{Or}$.
3. $\Sigma$ contains the special signature $\text{2-connective}((\text{formula} \Rightarrow \text{formula}) \Rightarrow \text{formula})$. This signature is assigned to the rule implication connective.
4. $\Sigma$ contains the signature $\text{1-connective}((\text{formula} \Rightarrow \text{formula}) \Rightarrow \text{formula})$. This signature is assigned to the negation connectives $\text{Not}$ and $\text{Neg}$, and to the reserved quantifiers of RIF-FLD, $\text{Forall}\text{?X}_1, \ldots, \text{?X}_n$, and $\text{Exists}\text{?X}_1, \ldots, \text{?X}_n$, for all variable sequences $\text{?X}_1, \ldots, \text{?X}_n$ and $n \geq 0$.
5. $\Sigma$ contains the signature $(e_1, \ldots, e_n, \ldots)$ for the equality symbol.
All arrow expressions $e_i$ here have the form $(\kappa \kappa) \rightarrow \gamma$ (the arguments in an equation must be compatible) and at least one of these expressions must have the form $(\kappa \kappa) \rightarrow \text{atomic}$ (i.e., equation terms are also atomic formulas). Dialects may further specialize this signature.

6. $\Sigma$ contains the signature $\#(e_1, \ldots, e_n)$ for membership terms.

Here all arrow expressions $e_i$ are binary (have two arguments) and at least one has the form $(\kappa \gamma) \rightarrow \text{atomic}$. Dialects may further specialize this signature.

7. $\Sigma$ contains the signature $\#(e_1, \ldots, e_n)$ for subclass terms.

Here all arrow expressions $e_i$ have the form $(\kappa \kappa) \rightarrow \gamma$ (the arguments must be compatible) and at least one of these arrow expressions has the form $(\kappa \kappa) \rightarrow \text{atomic}$. Dialects may further specialize this signature.

8. $\Sigma$ contains the signature $\rightarrow(e_1, \ldots, e_n)$ for frames.

Here all arrow expressions $e_i$ are ternary (have three arguments) and at least one of them is of the form $(\kappa_1 \kappa_2 \kappa_3) \rightarrow \text{atomic}$.

Dialects may further specialize this signature.

9. $\Sigma$ contains the signatures list and openlist for representing list terms.

- The signature list, for closed lists, has arrow expressions of the form $(\cdot \kappa) \rightarrow \kappa, (\kappa \kappa) \rightarrow \kappa, (\kappa \kappa \kappa) \rightarrow \kappa$, and so on, where $\kappa$ is a signature.
- The signature openlist, for open lists, has arrow expressions of the form $(\kappa \kappa) \rightarrow \kappa, (\kappa \kappa \kappa) \rightarrow \kappa$, and so on, where $\kappa$ is a signature.

10. $\Sigma$ contains the signature $\text{aggregate}(e_1, e_2, \ldots)$ for aggregate terms.

Here each arrow expression $e_i$ has the form $(\text{formula}) \rightarrow \kappa_i$, for some signatures $\kappa_1, \kappa_2, \ldots$.

11. $\Sigma$ contains the signature $\text{remote}(e_1, e_2, \ldots)$, where at least one of the $e_i$ is an arrow expression of the form $(\text{formula} \kappa) \rightarrow \text{formula}$ for some signature $\kappa$.

This signature is assigned to the remote term symbol $\@$.

12. $\Sigma$ has at most one signature for any given signature name.

13. Whenever $\Sigma$ contains a pair of signatures, $\eta A$ and $\eta B$, such that $\eta A \subseteq \eta B$.

Here $\eta A$ denotes a signature with the name $\eta$ and the associated set of arrow expressions $A$; similarly $\eta B$ is a signature named $\eta$ with the set of expressions $B$. The requirement that $\eta B \supseteq \eta A$ ensures that symbols that have signature $\eta$ can be used wherever the symbols with signature $\kappa$ are allowed.

The requirement that coherent sets of signatures must include the signatures for $=, \neq, \rightarrow$, and so on is just a technicality that simplifies definitions. Some of these signatures may go "unused" in a dialect even though, technically speaking, they must be present in the signature set associated with that dialect. If a dialect disallows equality, classification terms, or frames in its syntax then the corresponding signatures will remain unused. Such restrictions can be imposed by specializing RIF-FLD -- see Section Syntax of a RIF Dialect as a Specialization of RIF-FLD.

An incoherent set of signatures would be exemplified by one that includes signatures $\text{mysig}(\cdot) \rightarrow \text{atomic}$ and $\text{mysig}(\text{atomic}) \rightarrow \text{atomic}$ because it has two different signatures with the same name. Likewise, if a set contains $\text{mysig}(\cdot) \rightarrow \text{atomic}$ and $\text{mysig}(\text{atomic}) \rightarrow \text{atomic}$ and $\text{mysig}_1 < \text{mysig}_2$ then it is incoherent because the set of arrow expressions of $\text{mysig}_1$ does not contain the set of arrow expressions of $\text{mysig}_2$.

2.7 Presentation Syntax of a RIF Dialect

The presentation syntax of a RIF dialect is a set of well-formed formulas, as defined in the next section. The language of the dialect is determined by the following parameters (see Syntax of a RIF Dialect as a Specialization of RIF-FLD):

- An alphabet.
- A set of symbol spaces.
- An assignment of signatures from a coherent set of signatures to the symbols in Var, Const, connectives, and quantifiers:

  Each variable symbol is associated with exactly one signature from a coherent set of signatures. A constant symbol can have one or more signatures, and different symbols can be associated with the same signature. (Variables are not allowed to have multiple signatures because then well-formed terms would not be closed under substitutions. For instance, a term like $f(?X, ?X)$ could be well-formed, but $f(a, a)$ could be ill-formed.)

- Restrictions on the classes of terms allowed in the language of the dialect.
- Restrictions on the classes of formulas allowed in the language of the dialect.
- A coherent set of external schemas.

We have already seen how the alphabet and the symbol spaces are used to define RIF terms. The next section shows how signatures and external schemas are used to further specialize this notion to define well-formed RIF-FLD terms.

Note that the signatures for RIF-FLD connectives are fixed. Therefore, when defining a dialect, there is no need to repeat the definitions of 1-connecive, 2-connecive, etc. However, since the same symbol can be given several signatures, the syntactic context of connectives can be expanded by assigning more signatures to them. For instance, if a dialect allows rules to be reified and treated as objects, one could add another signature to $\rightarrow$ with the arrow expression $(\text{formula} \text{formula}) \rightarrow \text{individual}$, assuming that individual is a suitably chosen signature (e.g., one that is assigned to variables or a subsignature of the variable’s signature).

In contrast, the signatures for equality, frames, etc., are not completely fixed and have to be explicitly specialized by the dialects. For instance, in the arrow expression $(\kappa \kappa) \rightarrow \text{atomic}$ for the equality symbol, one has to tell what $\kappa$ exactly is. Since this kind of signatures must be explicitly defined by the dialects anyway, the dialect designer is allowed to add additional arrow expressions to them. If, for instance, a dialect designer would like to allow reification of the equality statements, the signature for $=$ could be defined as $=[[\text{individual} \text{individual}] \rightarrow \text{atomic}, \text{(individual individual)} \rightarrow \text{individual}$.
2.8 Well-formed Terms and Formulas

Since signature names uniquely identify signatures in coherent signature sets, we will often refer to signatures simply by their names. For instance, if one of f’s signatures is $\text{atomic}()$, we may simply say that symbol $f$ has signature atomic.

Definition (Well-formed term).
1. A constant or variable symbol with signature $\eta$ is a well-formed term with signature $\eta$.
2. A positional term $t(t_1 \ldots t_n)$, $\sigma n$, is well-formed and has a signature $\sigma$ iff
   - $t$ is a well-formed term that has a signature that contains an arrow expression of the form $(a_1 \ldots a_n) \Rightarrow \sigma$; and
   - Each $t_i$ is a well-formed term whose signature is $\gamma_i$ such that $\gamma_i \subseteq \sigma_i$.
   As a special case, when $n=0$ we obtain that $t()$ is a well-formed term with signature $\sigma$, if $t$’s signature contains the arrow expression $(\cdot) \Rightarrow \sigma$.
3. A term with named arguments $t(p_1\rightarrow t_1 \ldots p_n\rightarrow t_n)$, $\sigma n$, is well-formed and has a signature $\sigma$ iff
   - $t$ is a well-formed term that has a signature that contains an arrow expression with named arguments of the form $(p_1\rightarrow a_1 \ldots p_n\rightarrow a_n) \Rightarrow \sigma$; and
   - Each $t_i$ is a well-formed term whose signature is $\gamma_i$, such that $\gamma_i \subseteq \sigma_i$.
   As a special case, when $n=0$ we obtain that $t()$ is a well-formed term with signature $\sigma$, if $t$’s signature contains the arrow expression $(\cdot) \Rightarrow \sigma$.
4. An equality term of the form $t_1=t_2$ is well-formed and has a signature $\kappa$ iff
   - The signature $\kappa$ has an arrow expression $(\sigma \sigma) \Rightarrow \kappa$.
   - $t_1$ and $t_2$ are well-formed terms with signatures $\gamma_1$ and $\gamma_2$, respectively, such that $\gamma_1 \subseteq \sigma, i=1,2$.
5. A membership term of the form $t_1\# t_2$ is well-formed and has a signature $\kappa$ iff
   - The signature $\# \kappa$ has an arrow expression $(\sigma \sigma) \Rightarrow \kappa$.
   - $t_1$ and $t_2$ are well-formed terms with signatures $\gamma_1$ and $\gamma_2$, respectively, such that $\gamma_1 \subseteq \sigma, i=1,2$.
6. A subclass term of the form $t_1\#\# t_2$ is well-formed and has a signature $\kappa$ iff
   - The signature $\#\# \kappa$ has an arrow expression $(\sigma \sigma) \Rightarrow \kappa$.
   - $t_1$ and $t_2$ are well-formed terms with signatures $\gamma_1$ and $\gamma_2$, respectively, such that $\gamma_1 \subseteq \sigma, i=1,2$.
7. A frame term of the form $t[s_1\rightarrow v_1 \ldots s_n\rightarrow v_n]$ is well-formed and has a signature $\kappa$ iff
   - The signature $\rightarrow \kappa$ has arrow expressions $(\sigma \sigma) \Rightarrow \kappa$.
   - $t$, $s_i$, and $v_j$ are well-formed terms with signatures $\gamma$, $\gamma_1$, and $\gamma_2$, respectively, such that $\gamma \subseteq \sigma$ and $\gamma_1 \subseteq \gamma_2$.
   - $s_1, \ldots, s_n$ are well-formed terms with signatures $\gamma_1$ and $\gamma_2$, respectively, such that $\gamma_1 \subseteq \gamma_2, i=1,2$.
8. An externally defined term, $\text{External}(t \text{ loc})$, is well-formed and has signature $\kappa$ iff
   - $t$ is well-formed and has signature $\kappa$.
   - $\text{External}(t \text{ loc})$ is an instantiation of an external schema that belongs to a coherent set of external schemas of the language.

Note that, according to the definition of coherent sets of schemas, a term can be an instantiation of at most one external schema.

9. A formula term of the form $S(t_1 \ldots t_n)$, $\sigma n$, is well-formed if $S$ is a connective or a quantifier whose signature has an arrow expression $(\sigma_1 \ldots \sigma_n) \Rightarrow \text{formula}$ and each $t_i$ is a well-formed term whose signature is $\sigma_i$.
   In the special case of our reserved connectives and quantifiers, $t_1, \ldots, t_n$ must have signatures that are below formula, i.e., $\subseteq \text{formula}$. Also, if $S$ is $\land$ then $n$ must be equal 2 and if $S$ is $\neg$, $\forall$, or $\exists$, then $n=1$.
10. An aggregate term of the form $\text{sym}(T \{v_1 \ldots v_n\} \mid t)\kappa$ is well-formed if the aggregate symbol $\text{sym}$ $\{v_1 \ldots v_n\} \Rightarrow \kappa$ is assigned signature aggregate and the term $\text{sym}(v_1 \{v_1 \ldots v_n\} \mid t)$ is well-formed (as a positional term).
   This implies that $t$ must have the signature formula or $< \text{formula}$. Unless a dialect introduces additional signatures, this also means that $t$ must be a formula term (i.e., a compound formula) or an atomic formula (see below).
11. A remote term of the form $\varphi \text{@} r$ is well-formed if the positional term $(\varphi \mid r)$ is well-formed. This implies that $\varphi$ must be well-formed and have the signature formula, that $r$ must be a well-formed term, and that the term $\varphi \text{@} r$ itself has the signature formula (and, possibly, others).

Note that, like the constant symbols, well-formed terms can have more than one signature. Also note that, according to the above definition, $f()$ and $f$ are distinct terms.

Definition (Well-formed formula). A well-formed atomic formula is a well-formed term one of whose signatures is atomic or $< \text{atomic}$ (less than atomic in the order $\subseteq$). Note that equality, membership, subclass, and frame terms are atomic formulas, since atomic is one of their signatures. A well-formed formula is
- A well-formed term whose signature is formula or $< \text{formula}$; or
- A group formula; or
- A document formula.

Group and document formulas are defined below. For clarity, we will also give explicit definitions of conjunctive, disjunctive, rule, and other formulas even though they were already defined as special cases of the definition of well-formed formula terms (the first of the above bullets). Recall that all terms have a canonical function application form, but some are also written in a more familiar infix or prefix forms. For instance, rule implication, $\Rightarrow$, has the canonical form $\Rightarrow (a \cdot b)$ and the canonical form for negation, $\neg$, $p$ and $\neg$ $p$, is $\neg$ $p$ and $\neg$ $p$.

1. Atomic: If $\varphi$ is a well-formed atomic formula then it is also a well-formed formula.
2. Remote: A well-formed remote term $\varphi \text{@} r$ is also a well-formed formula.
3. Conjunction: If $\varphi_1, \ldots, \varphi_n, n \geq 0$, are well-formed formula terms then so is $\text{And}(\varphi_1 \ldots \varphi_n)$.
   As a special case, $\text{And}(\cdot)$ is allowed and is treated as a tautology, i.e., a formula that is always true.
4. Disjunction: If $\varphi_1, \ldots, \varphi_n, n \geq 0$, are well-formed formula terms then so is $\text{Or}(\varphi_1 \ldots \varphi_n)$.
   As a special case, $\text{Or}(\cdot)$ is treated as a contradiction, i.e., a formula that is always false.
5. Symmetric negation: If $\varphi$ is a well-formed formula term then so is $\neg$ $\varphi$. 

http://www.w3.org/TR/2012/PER-rif-fof-20121211/
6. **Default negation**: If \( φ \) is a well-formed formula term then so is \( Naf \ φ \).
7. **Rule implication**: If \( φ \) and \( γ \) are well-formed formula terms then so is \( φ \rightarrow γ \).
8. **Constraint**: If \( φ \) is a well-formed formula term then so is \( ∼ φ \).

This type of formula is also known as an error-producing constraint. The intent is that the constraint formula is satisfied if the premise \( φ \) is false. Constraints can also be viewed as rule implications whose conclusion is false.

9. **Universal and existential quantification**: If \( φ \) is a well-formed formula term then
   - \( \forall \gamma \: \γ_1 \ldots \gamma_n (φ) \)
   - \( \exists \gamma \: \γ_1 \ldots \gamma_n (φ) \)

are well-formed formula terms. Recall that \( \forall γ \: \γ_1 \ldots \gamma_n \) and \( \exists \gamma \: \gamma_1 \ldots \gamma_n \) are the reserved universal and existential quantifiers, respectively. The notation \( \forall \gamma \: \gamma_1 \ldots \gamma_n (φ) \) is an alternative for \( \forall γ_1 \ldots \gamma_n (φ) \), and similarly for \( \exists \).

10. **Group**: If \( φ_1, \ldots, φ_n \) are well-formed formula terms or group-formulas then \( Group(φ_1 \ldots φ_n) \) is a well-formed group formula. As a special case, the empty group formula, \( Group() \), is well-formed and is treated as a tautology, i.e., a well-formed formula that is always true.

Non-empty group formulas are intended to represent sets of formulas. Note that some of the \( φ_i \)'s can themselves be group formulas, which means that groups can be nested.

11. **Document**: An expression of the form \( Document(\text{directive}_1 \ldots \text{directive}_r \: Γ) \) is a well-formed document formula, if

   - \( Γ \) is an optional well-formed group formula; it is called the group formula associated with the document.
   - \( \text{directive}_1, \ldots, \text{directive}_r \) is an optional sequence of directives. A directive can be a dialect directive, a base directive, a prefix directive, an import directive, or a remote module directive.
     - **A dialect directive** has the form \( \text{Dialect}(D) \), where \( D \) is a Unicode string that specifies the name of a dialect. This directive specifies the dialect of a RIF document. Some dialects may require this directive in all of its documents, while others (notably, RIF-BLD) may not allow it and instead may entirely rely on other syntax. (Purely syntactic identification may not always be possible for dialects that are syntactically identical but semantically different, such as deductive databases with stable model semantics [GL88] and with well-founded semantics [GB95].) These two dialects are examples where the Dialect directive might be necessary.
     - **A base directive** has the form \( \text{Base}<\text{iri}> \), where \( \text{iri} \) is a Unicode string in the form of an absolute IRI.
       
       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]. This applies to relative IRIs that appear anywhere, including as constants, symbol spaces, location, and profile.
     - **A prefix directive** has the form \( \text{Prefix}(p <v>) \), 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 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: \( \text{Import}(loc) \) or \( \text{Import}(loc \: p) \).
       
       Here \( loc \) is a locator that uniquely identifies some other document, which is to be imported. The exact form of the locator \( loc \), the protocol that associates locators with documents, and the type of the imported documents is left to dialects to specify. However, all dialects must support the form \(<\!\!\text{iri}!\!>\), where \( \text{iri} \) is a sequence of Unicode characters that forms an IRI.
       
       The second argument to Import, \( p \), is a sequence of Unicode characters called the profile of import.
       
       RIF-FLD gives a semantics only to the one-argument directive \( \text{Import}(loc) \). The two-argument directive \( \text{Import}(loc \: p) \) is reserved for RIF dialects, which can use it to import non-RIF logical entities, such as RDFS data and OWL ontologies [RIF-RDF+OWL]. The profile can specify what kind of entity is being imported and under what semantics. For instance, the various RDF entailment regimes are specified in [RIF-RDF+OWL] as profiles that have the form of Unicode strings that form IRIs.
     - **A remote module directive** has the form \( \text{Module}(n \: loc) \). Here \( n \) is a variable-free term that represents the internal name of the remote module linked to the document – it is the name under which the module is referenced in the document. The second argument, \( loc \), is a locator for the document that contains the rules and the data of the module.
       
       As with Import, RIF-FLD does not restrict \( n \) and \( loc \) syntactically any further. However, we shall see that it does impose semantic restrictions on \( n \), and \( loc \) is required to uniquely identify an existing RIF document. The exact protocol that is used to associate \( loc \) with documents and the type of those documents is left to dialects.
       
       Note that although Base, Prefix, and Import all make use of symbols of the form \(<\!\!\text{iri}!\!>\) to indicate the connection of these symbols to IRIs, these symbols are not rif:iri constants, as semantically they are interpreted in a way that is quite different from constants.
       
       A document formula can contain at most one Dialect and at most one Base directive. The Dialect directive, if present, must be first, followed by an optional Base directive, followed by any number of Prefix directives, followed by any number of Import directives, followed by any number of Module directives.
       
       In the definition of a formula, the component formulas \( φ, φ_1, φ_2, \) and \( Γ \) are said to be subformulas of the respective formulas (conjunction, disjunction, negation, implication, group, etc.) that are built using these components.

Observe that the restrictions in (1) – (8) above imply that groups and documents cannot be nested inside formula terms and documents cannot be nested inside groups.

**Example 2** (Signatures, well-formed terms and formulas).

We illustrate the above definitions with the following examples. In addition to atomic, let there be another signature, term\{ \}, which is intended here to represent the context of the arguments to positional function or atomic formulas.

Consider the term \( p(p(a) \: p(a \: b \: c)) \). If \( p \) has the (polymorphic) signature \( \text{mysig}([\text{individual} = \text{individual}, \text{individual} \: \text{individual} = \text{individual}, \text{individual} \: \text{individual} \: \text{individual} = \text{individual}]) \) and \( a, b, c \) each has the signature \( \text{individual} \), then \( p(p(a) \: p(a \: b \: c)) \) is a well-formed term with signature \( \text{individual} \). If instead \( p \) had the signature \( \text{mysig2}([\text{individual} \: \text{individual} \: \text{individual}]) \), then...
individual}=individual, (individual individual individual)=individual then p(p(a) p(a b c)) would not be a well-formed term since then p(a) would not be well-formed (in this case, p would have no arrow expression which allows p to take just one argument).

For a more complex example, let r have the signature mysig3((individual)=atomic, (atomic individual)=individual, (individual individual individual)=individual). Then r(r(a) r(a b c)) is well-formed. The interesting twist here is that r(a) is an atomic formula that occurs as an argument to a function symbol. However, this is allowed by the arrow expression (atomic individual)=individual, which is part of r’s signature. If r’s signature were mysig4((individual)=atomic, (atomic individual)=atomic, (individual individual individual)=individual) instead, then r(r(a) r(a b c)) would be not only a well-formed term, but also a well-formed atomic formula.

An even more interesting example arises when the right-hand side of an arrow expression is something other than individual or atomic. For instance, let John, Mary, NewYork, and Boston have signatures individual{ }; flight and parent have signature hsig2(individual individual)=atomic; and closure has signature hh2((h2)=p2), where p2 is the name of the signature p2(individual individual)=atomic.

Then flight(NewYork Boston), closure(flight(NewYork Boston), parent(John Mary), and closure(parent)(John Mary) would be well-formed formulas. Such formulas are allowed in languages like HiLog [CKW93], which support predicate constructors like closure in the above example.

2.9 Annotations in the Presentation Syntax

RIF-FLD 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 constant and φ is a RIF formula that is not a document-formula. 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 rest of the annotation. RIF-FLD does not impose any restrictions on φ apart from what is stated above. This means that φ may include variables, function symbols, rif:local constants, and so on.

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

A convention is used to avoid a syntactic ambiguity in the above definition. For instance, in (* id φ *) t[w -> v] the annotation can be attributed to the term t or to the entire frame t[w -> v]. Similarly, for an annotated HiLog-like term of the form (* id φ *) f(a)(b,c), the annotation can be attributed to the entire term f(a)(b,c) or to just f(a). The convention adopted in RIF-FLD is that any annotation is syntactically associated with the largest RIF-FLD term or formula that appears to the right of that annotation. Therefore, in our examples the annotation (* id φ *) is considered to be attached to the entire frame t[w -> v] and to the entire term f(a)(b,c). 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(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 f"] *) 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:creator, dc:date, and foaf:maker.

Example 3 (A RIF-FLD document with nested groups and annotations)

We illustrate formulas, including documents and groups, with the following complete example (with apologies to Shakespeare for the imperfect rendering of the intended meaning in logic). For better readability, we use the notation defined in [RIF-DBE], which provides shortcuts for writing IRIs.

The first shortcut notation lets one write long rif:iri constants in the form prefix:name, where prefix is a short name that expands into an IRI according to a suitable Prefix directive. For instance, ex:man would expand into the rif:iri constant "http://example.org/ontology#man";

Example 3 (A RIF-FLD document with nested groups and annotations)

We illustrate formulas, including documents and groups, with the following complete example (with apologies to Shakespeare for the imperfect rendering of the intended meaning in logic). For better readability, we use the notation defined in [RIF-OFBE], which provides shortcuts for writing IRIs. The first shortcut notation lets one write long rif:iri constants in the form prefix:name, where prefix is a short name that expands into an IRI according to a suitable Prefix directive. For instance, ex:man would expand into the rif:iri constant "http://example.org/ontology#man";

Example 3 (A RIF-FLD document with nested groups and annotations)

We illustrate formulas, including documents and groups, with the following complete example (with apologies to Shakespeare for the imperfect rendering of the intended meaning in logic). For better readability, we use the notation defined in [RIF-OFBE], which provides shortcuts for writing IRIs. The first shortcut notation lets one write long rif:iri constants in the form prefix:name, where prefix is a short name that expands into an IRI according to a suitable Prefix directive. For instance, ex:man would expand into the rif:iri constant "http://example.org/ontology#man";

Example 3 (A RIF-FLD document with nested groups and annotations)

We illustrate formulas, including documents and groups, with the following complete example (with apologies to Shakespeare for the imperfect rendering of the intended meaning in logic). For better readability, we use the notation defined in [RIF-OFBE], which provides shortcuts for writing IRIs. The first shortcut notation lets one write long rif:iri constants in the form prefix:name, where prefix is a short name that expands into an IRI according to a suitable Prefix directive. For instance, ex:man would expand into the rif:iri constant "http://example.org/ontology#man";

Example 3 (A RIF-FLD document with nested groups and annotations)

We illustrate formulas, including documents and groups, with the following complete example (with apologies to Shakespeare for the imperfect rendering of the intended meaning in logic). For better readability, we use the notation defined in [RIF-OFBE], which provides shortcuts for writing IRIs. The first shortcut notation lets one write long rif:iri constants in the form prefix:name, where prefix is a short name that expands into an IRI according to a suitable Prefix directive. For instance, ex:man would expand into the rif:iri constant "http://example.org/ontology#man";

Example 3 (A RIF-FLD document with nested groups and annotations)

We illustrate formulas, including documents and groups, with the following complete example (with apologies to Shakespeare for the imperfect rendering of the intended meaning in logic). For better readability, we use the notation defined in [RIF-OFBE], which provides shortcuts for writing IRIs. The first shortcut notation lets one write long rif:iri constants in the form prefix:name, where prefix is a short name that expands into an IRI according to a suitable Prefix directive. For instance, ex:man would expand into the rif:iri constant "http://example.org/ontology#man";
The syntax of RIF-FLD relies on the signature mechanism and is not context-free, so EBNF does not capture this syntax precisely. As a result:

```plaintext
Dialect ::= 'Dialect' '(\' Name \')' 
Base ::= 'Base' '(:NCTERM #rif:dbflavor 'rif-ld')' 
Prefix ::= 'Prefix' '(:NCName #rif:dbflavor 'rif-locale')' 
Import ::= IRIMETA? 'Import' '(\' LOCATOR PROFILE \')' 
Module ::= IRIMETA? 'Module' '(: (Const | Expr) LOCATOR )' 
Group ::= IRIMETA? 'Group' '(\' (FORMULA | Group) \')' 
```

RIF defines a concrete syntax only for exchanging rules, and that syntax is XML-based, obtained as a refinement and serialization of the EBNF grammar. Keeping the above in mind, the EBNF grammar can be seen as just an intermediary between the mathematical English and the XML. However, it also gives a succinct view of the syntax of RIF-FLD and as such can be useful for dialect designers and users alike.

For all predicates/frames that supply the requisite information:

```plaintext
    pred:numeric-less-than(500 count(?Sem(?Crs))) \{Exists ?Stud\}) 
Forall ?Prof ?Crs ?Semester ?U (u:teaches(?Prof ?Crs ?Semester) :- 
    pred:numeric-less-than(500 count(?Prof(?Crs))) \{Exists ?Sem\}) 
Forall ?Crs (u:popular_course(?Crs) :- 
    pred:numerical-less-than(500 count(?Crs)->?Crs) \{Exists ?Crs\})) 
```

In this example, the main document contains three rules, which define the predicates u:takes, u:teaches, and u:popular_course. The information for the first two predicates is obtained by querying the remote modules corresponding to Universities 1 and 2. The rule that defines the first predicate says that if the remote university knowledge base says that a student takes a course in a certain semester then takes(s c s) is true in the main document. The second rule makes a similar statement about professors teaching courses in various semesters. Inside the main document, the external modules are referred to via the terms _univ(1) and _univ(2). The Module directives tie these references to the actual locations. The underscore in front of univ signifies that this is a rif:local symbol and is a shortcut for univ"^^rif:local, as defined in RIF-DTB.

Section Constants and Symbol Spaces. Note that the remote modules use frames to represent the enrollment information and predicates to represent course offerings. The rules in the main document convert both of these representations to predicates. The third rule illustrates a use of aggregation. The comprehension variable here is ?Stud and ?Crs is a grouping variable. Note that these are the only free variables in the formula over which aggregation is computed. For each course, the aggregate counts the number of students in that course over all semesters, and if the number exceeds 500 then the course is declared popular. Note also that the comprehension variable ?Stud is bound by the aggregate, so it is not quantified in the Forall-prefix of the rule.

The imported document has only one rule, which defines a new concept, u:student0f (a student is a student0f of a certain professor if that student takes a course from that professor). Since the main document imports the second document, it can answer queries about u:student0f as if this concept were defined directly within the main document.

2.10 EBNF Grammar for the Presentation Syntax of RIF-FLD

Until now, to specify the syntax of RIF-FLD we relied on "mathematical English," a special form of English for communicating mathematical definitions, examples, etc. We will now specify the syntax using the familiar EBNF notation. The following points about the EBNF notation should be kept in mind:

- The syntax of RIF-FLD relies on the signature mechanism and is not context-free, so EBNF does not capture this syntax precisely. As a result, the EBNF grammar defines a strict superset of RIF-FLD (not all formulas that are derivable using the EBNF grammar are well-formed).
- The EBNF syntax is not a concrete syntax: it does not address the 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 "" abstracts from one or more blanks, tabs, newlines, etc. This is done intentionally since RIF's presentation syntax is used as a tool for specifying the semantics and for illustration of the main RIF concepts through examples.
- RIF defines a concrete syntax only for exchanging rules, and that syntax is XML-based, obtained as a refinement and serialization of the EBNF syntax via the presentation-syntax-to-XML mapping for RIF-FLD.
The RIF-FLD presentation syntax does not commit to any particular vocabulary and permits arbitrary sequences of Unicode characters in constant symbols, argument names, and variables. Such sequences are denoted with UNICODESTRING in the above syntax. Constant symbols have this form: ':UNICODESTRING'"^^"SYMSPACE, where SYMSPACE is a ANGLEBRACKIRI or CURIE that represents the identifier of the symbol space of the constant. UNICODESTRING, ANGLEBRACKIRI, and CURIE are defined in Section Shortcuts for Constants in RIF's Presentation Syntax of [RIF-DTB]. Constant symbols can also have several shortcut forms, which are represented by the non-terminal NEWSYMBOL. These shortcuts are also defined in the same section of [RIF-DTB]. One of them is the CURIE shortcut, which is used in the examples in this document. Names are Unicode character sequences that form valid XML NCNames [XML-Names]. Variables are composed of Names prefixed with a ?-sign.

LOCATOR, which is used in several places in the grammar, is a non-terminal whose definition is left to the dialects. It is intended to specify the protocol by which external sources, remote modules, and imported RIF documents are located. This must include the basic form <IRI>, where IRI is a Unicode string in the form of an absolute IRI.

The symbols NEWCONNECTIVE, NEWQUANTIFIER, NEWAGGRFUNC, and NEWTERM are RIF-FLD extension points. They are not actual symbols in the alphabet. Instead, dialects are supposed to replace NEWCONNECTIVE, NEWQUANTIFIER, and NEWAGGRFUNC, by zero or more actual new symbols, while NEWTERM is to be replaced by zero or more new kinds of terms. Note that the extension point NEWSYMBOL is not shown in the EBNF grammar, since the grammar completely avoids mentioning the alphabet of the language (which is infinite).

Each RIF-FLD formula and term can be prefixed with one optional annotation, IRIMETA, for identification and metadata. IRIMETA is represented using (*\ldots*)-brackets that contain an optional rif:iri constant as identifier followed by an optional Frame or conjunction of Frames as metadata. 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].

Note that the RIF-FLD presentation syntax (as reflected in the above EBNF grammar) strives to have a more familiar look by avoiding some of the formal parts of the syntax defined in Sections Alphabet and Terms. For instance, as mentioned in those sections, the quantifier symbols Exists\ldots , Forall\ldots , and Forall\ldots exist as symbols. Likewise, the symbol OpenList is not used. Instead, open lists are written using the more familiar form LIST(Head|Tail). Also, some connectives, such as :-, are written in infix form.

Other connectives, such as Neg and Naf, are written in prefix form without parentheses.

3 Semantic Framework

Recall that the presentation syntax of RIF-FLD allows the use of shorthand notation, which is specified via the Prefix and Base directives, and various shortcuts for integer, real, and 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 Semantics of a RIF Dialect as a Specialization of RIF-FLD

The RIF-FLD semantic framework defines the notions of semantic structures and of models for RIF-FLD formulas. The semantics of a dialect is derived from these notions by specializing the following parameters.

1. The effect of the syntax.
   - The syntax of a dialect may limit the kinds of terms that are allowed.
     For instance, if a dialect’s syntax excludes frames or terms with named arguments then the parts of the semantic structures whose purpose is to interpret those types of terms become redundant.
   - The dialect might introduce additional terms and their interpretation by semantic structures.
   - The dialect might introduce additional connectives and quantifiers with their interpretation.

2. Truth values.

   The RIF-FLD semantic framework allows formulas to have truth values from an arbitrary partially ordered set of truth values, TV. A concrete dialect must select a concrete partially or totally ordered set of truth values.

3. Datatypes.

   A datatype is a symbol space whose symbols have a fixed interpretation in any semantic structure. RIF-FLD defines a set of core datatypes that each dialect is required to include, in addition to their syntax and semantics. However, RIF-FLD does not limit dialects to just the core types; they can introduce additional datatypes, and each dialect must define the exact set of datatypes that it includes.

4. Directives, connectives, extension points.

   Specialization of the definitions of RIF-BLD directives and logical connectives. Semantics of the syntactic components corresponding to the RIF-FLD extension points.

5. Logical entailment.

   Logical entailment in RIF-FLD is defined with respect to an unspecified set of intended semantic structures (sometimes also known as preferred semantic structures). This notion of entailment is very general and is known to subsume most of the known logical semantics for rule-based systems.

   A RIF dialect must define which semantic structures should considered intended. The actual choice depends on the desired semantics and is typically done by a trained logician. Many "off-the-shelf" semantics -- each suitable for different purposes -- have been defined in the literature. For instance, one dialect might specify that all semantic structures are intended (which leads to classical first-order entailment), another may consider only the minimal models as intended structures, while a third dialect might only use stable or well-founded models.

   These notions are defined in the remainder of this specification.

3.2 Truth Values

Definition (Set of truth values). Each RIF dialect must define the set of truth values, denoted by TV. This set must have a partial order, called the truth order, denoted ≤. In some dialects, < can be a total order. We write a ≤ b if either a < b or a and b are the same element of TV. In addition,

- TV must be a complete lattice with respect to ≤, i.e., the least upper bound (lub) and the greatest lower bound (glb) must exist for any subset of TV.
- TV is required to have two distinguished elements, f and t, such that f ≤ t for every t ∈ TV.
- TV has an operator of negation, ~: TV → TV, such that
  - ~ is a self-inverse function: applying ~ twice gives the identity mapping.
  - ~ is anti-monotonic: if f ≤ t implies ~f ≤ ~t.
  - ~t = f (and thus ~f = t).

The last condition follows from the earlier ones and is listed for didactic purposes only.

RIF dialects can have additional truth values. For instance, the semantics of some versions of NAF, such as well-founded negation, requires three truth values: t, f, and u (undefined), where f < t, u ≤ t. Handling of contradictions and uncertainty usually requires at least four truth values: t, u, f, and l (inconsistent). In this case, the truth order is partial: f < t, u < t, f < l, l < t. The negation operator ~ is then defined to be the identity on the new truth values u and l.

3.3 Datatypes

Definition (Datatype). A datatype is a symbol space that has

- an associated set, called the value space, and
- a mapping from the lexical space of the symbol space to the value space, called lexical-to-value-space mapping.

Semantic structures are always defined with respect to a particular set of datatypes, denoted by DTS. In a concrete dialect, DTS always includes the datatypes supported by that dialect. All RIF dialects must support the datatypes that are listed in Section Datatypes of (RIF-DTB). Their value spaces and the lexical-to-value-space mappings for these datatypes are described in the same section.

Although the lexical and the value spaces might sometimes look similar, one should not confuse them. Lexical spaces define the syntax of the constant symbols in the RIF language. Value spaces define the meaning of the constants. The lexical and the value spaces are often not even
isomorphic. For example, 1.2~"xs:decimal and 1.20~"xs:decimal are two legal -- and distinct -- constants in RIF because 1.2 and 1.20 belong to the lexical space of xs:decimal. However, these two constants are interpreted by the same element of the value space of the xs:decimal type. Therefore, 1.2~"xs:decimal = 1.20~"xs:decimal is a RIF tautology. Likewise, RIF semantics for datatypes implies certain inequalities. For instance, abc~"xs:string ≠ abcd~"xs:string is a tautology, since the lexical-to-value-space mapping of the xs:string type maps these two constants into distinct elements in the value space of xs:string.

3.4 Semantic Structures

The central step in specifying a model-theoretic semantics for a logic-based language is defining the notion of a semantic structure. Semantic structures are used to assign truth values to RIF-FLD formulas.

Definition (Semantic structure). A semantic structure, I, is a tuple of the form <TV, DTS, Dc, Ic, Iv, Ir, If, Ilist, Itail, Iframe, Isub, Ilea, Iext, Icon, Ivar, Istr>. Here D is a non-empty set of elements called the domain of I. We will continue to use Con to refer to the set of all constant symbols and Var to refer to 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 datatypes.

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

1. IC maps Con to elements of D.
   This mapping interprets constant symbols.

2. IV maps Var to elements of D.
   This mapping interprets variable symbols.

3. IL maps D to total functions D* → D (here D* is a set of all finite sequences over the domain D).
   This mapping interprets positional terms.

4. IN maps terms with named arguments. It is a total mapping from D to the set of total functions of the form SetOfFiniteBags(ArgNames × D) → D.
   This is analogous to the interpretation of positional terms with two differences:
   - Each pair <s, v> ∈ ArgNames × D represents an argument/value pair instead of just a value in the case of a positional term.
   - The argument to a term with named arguments is a finite bag of argument/value pairs rather than a finite ordered sequence of simple elements.
   - Bags are used here because the order of the argument/value pairs in a term with named arguments is immaterial and the pairs may repeat: p(a->b a->b). (However, p(a->b a->b) is not equivalent to p(a->b), as we shall see later.)
   - To see why such repetition can occur, note that argument names may repeat: p(a->b a->b). This can be understood as treating a as a bag-valued argument. Identical argument/value pairs can then arise as a result of a substitution. For instance, p(a->?a a->?b) becomes p(a->?a a->?b) if the variables ?a and ?b are both instantiated with the symbol b.

5. Ilist and Itail are used to interpret lists. They are mappings of the following form:
   - Ilist : D* → D
   - Itail : D* × D → D
   In addition, these mappings are required to satisfy the following conditions:
   - The function Ilist is injective (one-to-one).
   - The set Ilist(D*), henceforth denoted Dlist, is disjoint from the value spaces of all data types in DTS.
   - Itail(d1, ..., dk) = Ilist(d1, ..., dk) for d1, ..., dk ∈ D.

6. Iname is a total mapping from D to total functions of the form SetOfFiniteBags(D × D) → D.
   This mapping interprets frame terms. An argument, d ∈ D, to Iframe represents an object and a finite bag (a1, v1, ... , ak, vk) represents a bag (multiset) of attribute-value pairs for d. We will see shortly how Iname is used to determine the truth valuation of frame terms.

   Bags are employed here because the order of the attribute/value pairs in a frame is immaterial and the pairs may repeat. For instance, 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 a and b, respectively. (We shall see later that o[a->b a->b] is equivalent to o[a->b].)

7. Isub gives meaning to the subclass relationship. It is a total function D × D → 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.

8. Ieq gives meaning to class membership. It is a total function D × D → 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 # c1 and c1 == c1 must imply o # c1. This is ensured by a restriction in Section Interpretation of Formulas.

9. I = is a total function D × D → D.
   It gives meaning to the equality operator.
10. \( I_{\text{truth}} \) is a total mapping \( D \rightarrow TV \).

It is used to define truth valuation for formulas.

11. \( I_{\text{external}} \) is a mapping from the coherent set of schemas for externally defined terms to total functions \( D^* \rightarrow D \). For each external schema \( \sigma = (\forall \chi_1 \ldots \chi_n; \tau; \text{loc}) \) in the coherent set of such schemas associated with the language, \( I_{\text{external}}(\sigma) \) is a function of the form \( D^* \rightarrow D \).

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

- If \( \sigma \) is a schema of a built-in function then \( I_{\text{external}}(\sigma) \) must be the function defined in the aforesaid document.
- If \( \sigma \) is a schema of a built-in predicate then \( I_{\text{truth}} \circ I_{\text{external}}(\sigma) \) (the composition of \( I_{\text{truth}} \) and \( I_{\text{external}}(\sigma) \), a truth-valued function) must be as specified in \( \text{RIF-DB} \).

12. \( I_{\text{connective}} \) is a mapping that assigns every connective, quantifier, or aggregate symbol a function \( D^* \rightarrow D \).

Further restrictions on the interaction of this function with \( I_{\text{truth}} \) will be imposed in order to ensure the intended semantics for each connective and quantifier. For aggregates, \( I_{\text{connective}} \) maps them to functions \( D \rightarrow D \) and additional restrictions are imposed on the mapping \( I \) defined below.

We also define the following term-interpreting mapping on well-formed terms, which we denote using the same symbol \( I \) that is used for the semantic structure itself. This overloading is convenient and does not lead to ambiguity.

1. \( I(k) = I_c(k) \), if \( k \) is a symbol in Const
2. \( I(\text{?v}) = I_e(\text{?v}) \), if \( ?v \) is a variable in Var
3. \( I(f(t_1 \ldots t_n)) = I_e(f)((I(t_1)) \ldots (I(t_n))) \)
4. \( I(f(s_1 \ldots s_n)) = I_e(f)(\{s_1, s_2, \ldots, s_n\}) \)

Here we use \( (...) \) to denote a bag of argument/value pairs.

5. For list terms, the mapping is defined as follows:
   - \( I(\text{List}(\text{)}) = I_{\text{list}}(\text{}) \)
   - \( I(\text{List}(\{t_1, \ldots, t_n\}) = I_{\text{list}}(I(t_1), \ldots, I(t_n)), \text{if } n \geq 0 \)
   - \( I(\text{List}(\{t_1, \ldots, t_n \mid t\}) = I_{\text{list}}(I(t_1), \ldots, I(t_n), I(t)), \text{if } n > 0 \)

6. \( I(\text{o[a->b a->b]} = I_{\text{frame}}(I(o))\{\langle \text{o(a)}, \text{o(b)}\rangle, \ldots, \langle \text{o(a)}, \text{o(b)}\rangle\}) \)

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

7. \( I(\text{c1##c2}) = I_{\text{class}}(I(c_1), I(c_2)) \)
8. \( I(\text{c#c}) = I_{\text{class}}(I(c), I(c)) \)
9. \( I(\text{x=y}) = I_{\text{equiv}}(I(x), I(y)) \)
10. \( I(\text{External}(t \text{ loc})) = I_{\text{external}}(I)(I(s_1), \ldots, I(s_n)), \text{if } \text{External}(t \text{ loc}) \text{ is an instantiation of the external schema } \sigma = (\forall \chi_1 \ldots \chi_n; \tau; \text{loc}) \text{ by substitution } \forall \chi_1/s_1 \ldots \forall \chi_n/s_n. \)

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

11. If \( S \) is a connective, a quantifier, or an aggregate and \( S(t_1 \ldots t_n) \) is a well-formed formula term (for an aggregate, \( n = 1 \)) then \( I(S(t_1 \ldots t_n)) = I_{\text{connective}}(S)(I(t_1) \ldots I(t_n)) \)

12. For standard aggregates, the mapping \( I \) is defined as follows.

Let \( \text{aggr} \{\forall \chi_1 \ldots \chi_n; \tau; \text{loc}\} \) be an aggregate and let \( S \) be the following set:

\( S = \{ I_{\text{aggr}}(I_{\text{?v}}(\forall \chi_1 \ldots \chi_n; \tau; \text{loc})), \ldots, I_{\text{aggr}}(I_{\text{?v}}(\forall \chi_1 \ldots \chi_n; \tau; \text{loc})) \} \) for all semantic structures \( I' \) such that \( I'(\tau) = t \) and \( I' \) is exactly like \( I \) except that \( I_{\text{?v}}(\forall \chi_1 \ldots \chi_n; \tau; \text{loc}) \) is different from \( I_{\text{?v}}(\forall \chi_1 \ldots \chi_n; \tau; \text{loc}) \).

In addition, let \( S_{\text{set}} \) denote the set of all elements \( x \) such that \((x, x_1, \ldots, x_n) \in S \) and \( S_{\text{bag}} \) denote the bag of all such elements \( x \) (i.e., \( S_{\text{bag}} \) can have repeated occurrences of the same element).

a. Set aggregate:
   - \( I(\text{setof}(\forall \chi_1 \ldots \chi_n; \tau; \text{loc})), I(t) \) = \( I_{\text{list}}(L) \)
   - where \( L \) is a sorted list of the elements in \( S_{\text{set}} \). Since sorting requires an ordering, the above is well-defined only for semantic structures with totally ordered domains. If \( L \) is infinite then the value of the aggregate in \( I \) is indeterminate (i.e., it can be any element of the domain \( D \)).

The requirement that the list \( L \) must be sorted comes from the fact that there can be many ways to represent \( S_{\text{set}} \) as a list, while \( I(\text{setof}(\forall \chi_1 \ldots \chi_n; \tau; \text{loc})), I(t) \) must be defined as one concrete element of the domain \( D \). Sorting a set is a standard way of providing the requisite unique representation.

b. Bag aggregate:
   - \( I(\text{bagof}(\forall \chi_1 \ldots \chi_n; \tau; \text{loc})), I(t) \) = \( I_{\text{list}}(L) \)
   - where \( L \) is a sorted list of the elements in \( S_{\text{bag}} \). This is well-defined only for semantic structures with totally ordered domains. If \( L \) is infinite then the value of the aggregate in \( I \) is indeterminate (i.e., it can be any element of the domain \( D \)).

The reason for sorting \( L \) is the same as in the case of the set aggregate.

c. Min aggregate:
If the signature

\[ t(a\rightarrow 1 \ b\rightarrow 2 \ c\rightarrow 3) = \text{min}(S_{\text{bag}}), \]

if the function \( \text{min} \) is defined for \( S_{\text{bag}} \) in the dialect. If not, the value of the aggregate in \( I \) is indeterminate. The bag \( S_{\text{bag}} \) must have a well-defined total order and \( \text{min} \) must compute the minimum elements of finite totally ordered bags.

d. Max aggregate:

\[ t(\text{max}(\{ t | t \}) = \text{max}(S_{\text{bag}}), \]

if the function \( \text{max} \) is defined for \( S_{\text{bag}} \) in the dialect. If not, the value of the aggregate in \( I \) is indeterminate. The bag \( S_{\text{bag}} \) must have a well-defined total order and \( \text{max} \) must compute the maximum elements of finite totally ordered bags.

e. Count aggregate:

\[ I(\text{count}(\{ t | t \}) = \text{count}(S_{\text{bag}}), \]

if the function \( \text{count} \) is defined for \( S_{\text{bag}} \) in the dialect. If not, the value of the aggregate in \( I \) is indeterminate. The function \( \text{count} \) must compute the cardinality of finite bags.

f. Sum aggregate:

\[ I(\text{sum}(\{ t | t \}) = \text{sum}(S_{\text{bag}}), \]

if the function \( \text{sum} \) is defined for \( S_{\text{bag}} \) in the dialect. If not, the value of the aggregate in \( I \) is indeterminate. The function \( \text{sum} \) must compute summations of the elements of finite bags. (For decimals, integers, floats, etc., summation must coincide with the usual notion. However, this function might also be defined for other domains in some dialects.)

g. Prod aggregate:

\[ I(\text{prod}(\{ t | t \}) = \text{prod}(S_{\text{bag}}), \]

if the function \( \text{prod} \) is defined for \( S_{\text{bag}} \) in the dialect. If not, the value of the aggregate in \( I \) is indeterminate. The function \( \text{prod} \) must compute products of the elements of finite bags. (For decimals, integers, floats, etc., product must coincide with the usual notion. However, this function might also be defined for other domains.)

h. Avg aggregate:

\[ I(\text{avg}(\{ t | t \}) = \text{avg}(S_{\text{bag}}), \]

if the function \( \text{avg} \) is defined for \( S_{\text{bag}} \) in the dialect. If not, the value of the aggregate in \( I \) is indeterminate. The function \( \text{avg} \) must compute averages (arithmetic means) of the elements of finite bags. (For decimals, integers, floats, etc., average must coincide with the usual notion. However, this function might also be defined for other domains.)

13. For remote terms of the form \( \varphi[r] \), the mapping \( I \) is defined in Section Interpretation of Documents.

Note that the definitions of \( h_{\text{RIF}}, h_{\text{name}} \), and of \( I(x=y) \) imply that the terms with named arguments that differ only in the order of their arguments are mapped by \( I \) to the same argument. Similarly, frame terms that differ only in the order of their attribute/value pairs (or in the number of repetitions of the same attribute/value pair) are mapped to the same domain element. This implies that the equalities like \( t(a\rightarrow 1 \ b\rightarrow 2 \ c\rightarrow 3) = t(c\rightarrow 3 \ a\rightarrow 1 \ b\rightarrow 2) \) and \( \text{ex:} \{ \text{ex:a}\rightarrow 1 \ \text{ex:b}\rightarrow ^{\text{abc}} \ \text{ex:a}\rightarrow 1 \} = \text{ex:} \{ \text{ex:b}\rightarrow ^{\text{abc}} \ \text{ex:a}\rightarrow 1 \} \) are tautologies in RIF-FDL.

The effect of signatures. For every signature, \( sg \), supported by a dialect, there is a subset \( D_{sg} \subseteq D \), called the domain of the signature. Terms that have a given signature, \( sg \), must be mapped by \( I \) to \( D_{sg} \), and if a term has more than one signature it must be mapped into the intersection of the corresponding signature domains. To ensure this, the following is required:

1. If \( sg < sg' \) then \( D_{sg} = D_{sg'} \).
2. If \( k \) is a constant that has signature \( sg \) then \( I(k) \in D_{sg} \).
3. If \( ?v \) is a variable that has signature \( sg \) then \( I(?v) \in D_{sg} \).
4. If \( sg \) has an arrow expression of the form \( (s_1 ... s_n) \rightarrow \) then, for every \( d \in D_{sg} \), \( I(d) \) must map \( D_{sg} \times ... \times D_{sg} \) to \( D \).
5. If \( sg \) has an arrow expression of the form \( (p_1 \rightarrow 1 \ ... \ p_n \rightarrow s_n) \rightarrow \) then, for every \( d \in D_{sg} \), \( I(d) \) must map the set \{ \( <p_1,d_1>, ... <p_n,d_n> \) \} to \( D \).
6. If the signature \( -> \) has arrow expressions \( (sg_1,s_1), ... (sg_n,s_n) \rightarrow \), then, for every \( d \in D_{sg} \), \( I(d) \) must map \( D_{sg} \times D \rightarrow D \).
7. If the signature \( * \) has an arrow expression \( (s -> k) \rightarrow \) then \( I(s) \) must map \( D_{sg} \times D \rightarrow D \).
8. If the signature \( # \) has an arrow expression \( (s \rightarrow k) \rightarrow \) then \( I(s) \) must map \( D_{sg} \times D \rightarrow D \).
9. If the signature \( = \) has an arrow expression \( (s \rightarrow k) \rightarrow \) then \( I(s) \) must map \( D \rightarrow D \).

The effect of datatypes. The datatype identifiers in \( DTS \) impose the following restrictions. If \( dt \in DTS \), let \( LS_{dt} \) denote the lexical space of \( dt \), \( VS_{dt} \) denote its value space, and \( L_{dt}: LS_{dt} \rightarrow VS_{dt} \) the lexical-to-value-space mapping. Then the following must hold:

- \( VS_{dt} \subseteq D \) and \( LS_{dt} \).
- For each constant \( "\text{lit}" \rightarrow \text{dt} \) such that \( \text{lit} \in LS_{dt} \), \( I("\text{lit}" \rightarrow \text{dt}) = L_{dt}(\text{lit}) \).

That is, \( I \) must map the constants of a datatype \( dt \) in accordance with \( L_{dt} \).

RIF-FDL does not impose special requirements on \( I \) for constants in the symbol spaces that do not correspond to the identifiers of the datatypes in \( DTS \). Dialects may have such requirements, however. An example of such a restriction could be a requirement that no constant in a particular symbol space (such as \( \text{rif:local} \)) can be mapped to \( VS_{dt} \) of a datatype \( dt \).

3.5 Annotations and the Formal Semantics

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

Note that although annotations associated with RIF-FDL formulas are ignored by the semantics, they can be extracted by XML tools. Since annotations are represented by frame terms, they can be reasoned with the rules. The frame terms used to represent metadata can then be fed to other formulas, thus enabling reasoning about metadata. However, RIF does not define any concrete semantics for metadata.

3.6 Interpretation of Non-document Formulas

This section defines how a semantic structure \( I \) determines the truth value \( TV(I(\varphi)) \) of a RIF-FDL formula, \( \varphi \), where \( \varphi \) is any formula other than a document formula or a remote formula. Truth valuation of document formulas is defined in the next section.

To this end, we define a mapping, \( TV \), from the set of all non-document formulas to \( TV \). Note that the definition implies that \( TV(I(\varphi)) \) is defined only if the set \( DTS \) of the datatypes in \( I \) includes all the datatypes mentioned in \( \varphi \).
Definition (Truth valuation). Truth valuation for well-formed formulas in RIF-FLD is determined using the following function, denoted \( TVal \):

1. **Constants:** \( TVal(k) = I_{truth}(I(k)) \), if \( k \in \text{Const} \).
2. **Variables:** \( TVal(?v) = I_{truth}(I(?v)) \), if \( ?v \in \text{Var} \).
3. **Positional atomic formulas:** \( TVal(r(t_1 \ldots t_n)) = I_{truth}(I(r(t_1 \ldots t_n))) \).
4. **Atomic formulas with named arguments:** \( TVal(p(s_1\ldots v_1 \ldots s_k\ldots v_k)) = I_{truth}(I(p(s_1 \ldots v_1 \ldots s_k\ldots v_k))) \).
5. **Equality:** \( TVal(x = y) = I_{truth}(I(x = y)) \).

To ensure that equality has precisely the expected properties, it is required that

- \( I_{truth}(I(x = y)) = t \) if \( x = y \) and \( I_{truth}(I(x = y)) = f \) otherwise.

6. **Subclass:** \( TVal(sc \# cl) = I_{truth}(I(sc \# cl)) \).

To ensure that the operator \# is transitive, i.e., \( c_1 \# c_2 \) and \( c_2 \# c_3 \) imply \( c_1 \# c_3 \), the following is required:

- For all well-formed terms \( c_1, c_2, c_3 \) : \( \text{gb}(TVal(c_1 \# c_2), TVal(c_2 \# c_3)) \leq TVal(c_1 \# c_3) \).

Note that this is a restriction on \( I_{truth} \) and the mapping \( I \), which is expressed in a more succinct form using \( TVal \).

7. **Membership:** \( TVal(o \# cl) = I_{truth}(I(o \# cl)) \).

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

- For all well-formed terms \( o, cl, scl \) : \( \text{gb}(TVal(o \# cl), TVal(cl \# scl)) \leq TVal(o \# scl) \).

Note that this is a restriction on \( I_{truth} \) and the mapping \( I \), which is expressed in a more succinct form using \( TVal \).

8. **Frame:** \( TVal(o(a_1\ldots v_1 \ldots a_k\ldots v_k)) = I_{truth}(I(o[a_1\ldots v_1 \ldots a_k\ldots v_k]) \).

Since the bag of attribute/value pairs represents the conjunction of all the pairs, the following is required:

- \( TVal(o(a_1\ldots v_1 \ldots a_k\ldots v_k)) = \text{gb}(TVal(o[a_1\ldots v_1]), \ldots, TVal(o[a_k\ldots v_k])) \).

Observe that this is a restriction on \( I_{truth} \) and the mapping \( I \). For brevity, it is expressed in a more succinct form using \( TVal \).

9. **Externally defined atomic formula:** \( TVal(\text{External}(t \# loc)) = I_{truth}(I_{\text{External}}(I(t), \ldots, I(loc))) \).

Note that, by definition, \( \text{External}(t \# loc) \) is well-defined only if it is an instantiation of an external schema. Furthermore, by the definition of coherent sets of external schemas, it can be an instantiation of at most one external schema, so \( I_{\text{External}}(t \# loc) \) is well-defined.

10. **Connectives and quantifiers:** if \( S \) is a connective or a quantifier and \( S(t_1 \ldots t_n) \) is a well-formed formula term then \( TVal(S(t_1 \ldots t_n)) = I_{truth}(I(S(t_1 \ldots t_n))) \).

To ensure the intended semantics for the RIF-FLD reserved connectives and quantifiers, the following restrictions are imposed (observe that all these are restrictions on \( I_{truth} \) and the mapping \( I \), which are expressed via \( TVal \), for brevity):

a. **Conjunction:** \( TVal(\text{And}(c_1 \ldots c_n)) = \text{gb}(TVal(c_1), \ldots, TVal(c_n)) \).

   The empty conjunction is treated as a tautology, so \( TVal(\text{And}()) = t \).

b. **Disjunction:** \( TVal(\text{Or}(c_1 \ldots c_n)) = \text{lub}(TVal(c_1), \ldots, TVal(c_n)) \).

   The empty disjunction is treated as a contradiction, so \( TVal(\text{Or}()) = f \).

c. **Negation:** \( TVal(\text{Neg} \ ?v) = TVal(?v) \) and \( TVal(\text{Naf} \ ?v) = \neg TVal(?v) \).

   The symbol \( \neg \) is the self-inverse operator of negation on \( TVal \) introduced in Section Truth Values.

   The symmetric negation, \( \text{Neg} \), is sufficiently general to capture many different kinds of such negation. For instance, classical negation would, in addition, require \( TVal(\text{Neg} \ ?v) = \neg TVal(?v) \); strong negation (analogous to the one in [APPS9]) can be characterized by \( TVal(\text{Neg} \ ?v) \leq \neg TVal(?v) \); and explicit negation (analogous to [APPS9]) would require no additional constraints.

   Note that both classical and default negation are interpreted the same way in any concrete semantic structure. The difference between the two kinds of negation comes into play when logical entailment is defined.

d. **Quantification:**

   - \( TVal(\exists ?v \ldots ?v_n \ ?v) = \text{lub}(TVal(?v)) \).
   - \( TVal(\forall ?v \ldots ?v_n \ ?v) = \text{gb}(TVal(?v)) \).

   Here \( \text{lub} \) (respectively, \( \text{gb} \)) is taken over all interpretations \( I \) of the form \( <TV, DTS, D, IC, P, \text{Ir}, \text{lr}, \text{Iat}, \text{Iall}, \text{Iname}, \text{Iab}, \text{Iref}, \text{Iv}, \text{Iexternal}, \text{Iconnective}, \text{Itruth}> \), which are exactly like \( I \), except that the mapping \( P, \text{Ir} \), is used instead of \( I, \text{Ir} \). \( P, \text{Ir} \) is defined to coincide with \( I \) on all variables except, possibly, on \( ?v_1, \ldots, ?v_n \).

e. **Rule implication:**

   - \( TVal(\text{head} :: \text{body}) = t \), if \( TVal(\text{head}) \geq TVal(\text{body}) \).
   - \( TVal(\text{head} :: \text{body}) \leq t \) otherwise.

f. **Constraint:**

   - \( TVal(\text{head} :: \text{body}) = t \), if \( TVal(\text{body}) = f \).
   - \( TVal(\text{head} :: \text{body}) \leq t \) otherwise.

g. **Dialects that introduce additional connectives and quantifiers should define appropriate restrictions on \( TVal \) to give these new elements desired semantics.**

11. **Groups of formulas:**
If $\Gamma$ is a group formula of the form $\text{Group}(\varphi_1 \ldots \varphi_n)$ then

\[
TVal(\Gamma) = \text{glb}(TVal(\varphi_1), \ldots, TVal(\varphi_n)).
\]

This means that a group of formulas is treated as a conjunction. In particular, the empty group is treated as a tautology, so $TVal(\text{Group}()) = t$. 

Note that rule implications and equality formulas are always two-valued, even if $TV$ has more than two values.

### 3.7 Interpretation of Documents

Document formulas are interpreted using semantic multi-structures, which are sets of semantic structures. Their purpose is to provide a semantics to RIF multi-documents, i.e., RIF documents that import other RIF documents and/or contain references to other RIF documents via remote module reference formulas.

**Definition (Imported document).** Let $\Delta$ be a document formula and $\text{Import}(\text{loc})$ be one of its import directives, where $\text{loc}$ is a locator of another document formula, $\Delta'$. In this case, we say that $\Delta'$ is **directly imported** into $\Delta$.

A document formula $\Delta'$ is said to be imported into $\Delta$ if it is either directly imported into $\Delta$ or it is imported (directly or not) into another document, which itself is directly imported into $\Delta$. 

The above definition deals only with one-argument import directives, since two-argument directives are expected to be defined on a case-by-case basis by other specifications that might be integrated with RIF. For instance, [RIF-RDF+OWL](#) defines the semantics of the 2-argument import directive for importing RDF and OWL documents into RIF-BLD.

**Definition (Renaming apart of local constants).** A **renaming mapping**, $\rho$, is a function that maps document formulas to document formulas subject to the following restriction:

- If $\rho(\Delta) = \Delta'$ then $\Delta'$ is exactly as $\Delta$ except that all occurrences of some $\text{rif:local}$ constants in $\Delta$ may be consistently renamed into other $\text{rif:local}$ constants.
- By consistent renaming here we mean that different occurrences of the same $\text{rif:local}$ constant in $\Delta$ are renamed identically.

**Definition (Semantic multi-structure).** A **semantic multi-structure**, $I$, is a triple $(I_{\text{ren}}, I_{\text{map}}, I_{\text{set}})$ where

- $I_{\text{ren}}$ is a renaming mapping;
- $I_{\text{map}}$ is a set of adorned semantic structures of the form $(f^1, f^2, \ldots)$, where the adornments $m_1, m_2, \ldots$ are elements in the interpretation domain (explained below);
- $I_{\text{set}}$ is a **modularization mapping**. It is a function from the set of all documents in the language to the powerset of $I_{\text{set}}$. That is, if $\Delta$ is a document then $I_{\text{map}}(\Delta) \subseteq I_{\text{set}}(I_{\text{set}}(\Delta) \text{ can be empty})$.

All these mappings are subject to the following restrictions:

- All semantic structures in $I_{\text{set}}$ have the same domain and the mappings $f^1, f^2, \ldots$ are all the same. We will denote this mapping by $I_C$ and the common domain will be called the domain of $I$.
- Note that this implies that if $t$ is a ground (i.e., variable-free) term then $f^1(t) = f^2(t) = \ldots$, and so we can write $I(t)$ without ambiguity.
- The adornments $m_1, m_2, \ldots$ are elements in the domain of $I$ and $I_{\text{set}}$ can have at most one semantic structure with a given adornment. Intuitively, the adornments represent module names and thus $f^1$ is to be understood as the semantic structure for interpreting the documents associated with module $m_1$.
- If one document, $\Delta$, imports another document, $\Delta'$, either directly or indirectly then $I_{\text{map}}(\Delta) \subseteq I_{\text{map}}(\Delta')$.
- That is, if a super-document is associated with some module then its imported documents are also associated with that module. However, the imported document can be associated with other modules as well.
- If Module$(\text{loc})$ appears in some document, where $\text{loc}$ is a locator for another document, $\Delta$, then $f^1(\text{loc}) \in I_{\text{map}}(\Delta) \subseteq I_{\text{set}}$.
- Thus, in this case, $f^1(\text{loc})$ must adorn one (and only one) semantic structure in $I_{\text{set}}$.

Intuitively, these conditions say that if some document associates $\Delta$ with a particular module then $I_{\text{map}}$ must respect that. Note that different directives Module$(\text{loc}_1), \ldots, \text{Module}(\text{loc}_k)$ (with different locators but the same module name) have the effect that multiple documents become associated with the same document and the same semantic structure. This association happens through the Module directives that occur inside the same or different documents. The semantic effect is that all these associated document formulas are true in that module.

**Definition (Remote module).** Let $\Delta$ be a document formula and let Module$(\text{loc})$ be one of its remote module directives, where $\text{loc}$ is a locator for another document formula, $\Delta'$. In this case, we say that $\Delta'$ is a **directly linked remote module** of $\Delta$.

A document formula $\Delta'$ is said to be a **linked remote module** for $\Delta$ if it is either directly linked to $\Delta$ or it is linked (directly or not) to some other document that is either imported into or directly linked to $\Delta$.

In the definition of term-interpreting mappings, we postponed defining remote term references till now. The next definitions fill in this gap. The reason we postponed this definition was that this could not be defined solely in reference to a single semantic structure $I$: a multi-structure context is required in addition to $I$. This leads us to $TVal_{\varphi}(\varphi)$, the notion of truth valuation in the context of a semantic multi-structure $I$.

**Definition (Term-interpreting mapping for remote term references).** Let $\varphi$ be a non-document formula, $I$ a semantic multi-structure, and $f \in I_{\text{set}}$.

First we define $I(\varphi)$, the term-interpreting function for $f$ in the context of a multi-structure, $I$. For terms that are not remote references, the definition is exactly as for term-interpreting mappings for ordinary semantic structures with the difference that $I(\varphi)$ is used everywhere instead of just $f$. The definition of $TVal_{\varphi}(\varphi)$, the truth valuation in the context of $I$, is the same as the definition of $TVal_{\varphi}$ for ordinary semantic structures in this case.

However, if $\varphi$ is a remote reference $\Psi@r$ then $I(\varphi)$ is defined as follows:
Let \( I(r) = d \). If \( I_{doc} \) has no semantic structure adorned with \( d \), the value of \( I(j) \) is indeterminate (i.e., it can be any element of the domain of \( I \)). Otherwise, let \( j \in I_{doc} \) be the structure adorned with \( d \) (it is then unique, by definition). In that case, we define:

- \( I(q) \) to be \( j(q) \)
- \( TVal(I(q)) = I_{truth}(I(q)) \)

We now define how truth of document formulas is determined in semantic multi-structures.

**Definition (Truth valuation of formulas in multi-document structures).** Let \( \Delta \) be a document formula and let \( \Delta_1, \ldots, \Delta_n \) be all the RIF-FLD document formulas that are imported into \( \Delta \) or linked to it (directly or indirectly). Let furthermore \( \Gamma_1, \ldots, \Gamma_m \) be the respective group formulas associated with these documents. Let \( I = (I_{doc}, I_{map}, I_{set}) \) be a semantic multi-structure where \( I_{set} = \{ f^1, f^2, \ldots \} \).

We define the truth valuation for \( \Delta \) as follows.

- \( TVal(\Delta) = glb\{ TVal(g(I_{doc})) | J \in I_{map}(\Delta) \}, \ TVal(g(I_{doc})) \ | J \in I_{map}(\Delta_2) \}, \ldots, \ TVal(g(I_{doc})) | J \in I_{map}(\Delta_n) \} \).

Observe that, before computing truth values, documents’ rif:local constants may be renamed. □

It is instructive to see how remote terms are interpreted by a semantic multi-structure \( I \). Suppose \( \phi \) is such a term that occurs somewhere in a document, \( \Delta \). If \( J \in I_{map}(\Delta) \) is an ordinary semantic structure associated to \( \Delta \) then \( f(J) \) determines the semantic structure in which \( \phi \) is to be evaluated: it is the structure in \( I_{set} \) adorned with \( J(r) \), say \( M \). This is what \( f(J(r)) \) is all about. Note that \( \psi \) may also contain remote term references (as \( \phi \) may be a compound formula). In this case, since \( \psi \) is evaluated using \( M \), the same principle applies.

**Definition (Models).** Let \( I \) be a semantic structure or a multi-structure. We say that \( I \) is a model of a formula, \( \phi \), written as \( I \models \phi \), iff \( TVal(I(\phi)) = t \).

Here \( \phi \) is a document formula, if \( I \) is a multi-structure, and a non-document formula, if \( I \) is an ordinary semantic structure. □

### 3.8 Intended Semantic Structures

The semantics of a set of formulas, \( \Gamma \), is the set of its intended (or preferred) semantic multi-structures. Intended multi-structures are used to define the notion of logical entailment in RIF dialects. RIF-FLD does not fix what these intended multi-structures should be, leaving the choice to RIF dialects. Different logic dialects may use different criteria for what is to be considered an intended semantic multi-structure, and the freedom to set these criteria lets RIF-FLD cover a wide range of possible logical semantics. The actual choice of intended models for this or that logic dialect is a prerogative of the dialect designer and should be attempted only by a trained logician.

For instance, to model the classical first-order notion of entailment, every semantic structure would be intended. For [RIF-BLD](http://www.w3.org/TR/2012/PER-rif-bl-20121211/), which is based on Horn rules, intended multi-structures are defined only for sets of rules: an intended semantic multi-structure of a RIF-BLD set of formulas, \( \Gamma \), is the unique minimal Henbrand model ([Lloyd87]) of \( \Gamma \). For dialects in which rule bodies may contain literals negated with the default negation connective \( \text{Naf} \), only some of the minimal Henbrand models of \( \Gamma \) are intended. Each logic dialect of RIF must define the set of intended semantic multi-structures precisely. The two most common theories for default negation use the well-founded models ([GRS91]) and stable models ([GL88]) as their intended models.

The following example illustrates the notion of intended semantic structures. Suppose \( \Gamma \) consists of a single rule formula \( p : \text{Naf} \ q \). If \( \text{Naf} \ q \) were interpreted as classical negation, then this rule would be simply equivalent to \( \text{Or}(p, q) \), and so it would have two kinds of models: those where \( p \) is true and those where \( q \) is true. In contrast to first-order logic, most rule-based systems do not consider \( p \) and \( q \) symmetrically. Instead, they view the rule \( p : \text{Naf} \ q \) as a statement that \( p \) must be true if it is not possible to establish the truth of \( q \). Since it is, indeed, impossible to establish the truth of \( q \), such theories would derive \( p \) even though it does not logically follow from \( \text{Or}(p, q) \). The logic underlying rule-based systems also assumes that only the minimal Henbrand models are intended (minimality here is with respect to the set of true facts). Furthermore, although our example has two minimal Henbrand models — one where \( p \) is true and \( q \) is false, and the other where \( p \) is false, \( q \) is true, only the first model is considered to be intended.

The above concept of intended semantic multi-structures and the corresponding notion of logical entailment, below, is due to [Shoham87] (where these structures were called preferred).

### 3.9 Logical Entailment

We will now define what it means for one RIF-FLD formula to entail another. This notion is typically used for defining queries to knowledge bases and for other tasks, such as testing subsumption of concepts (e.g., in OWL). We assume that each set of formulas has an associated set of intended semantic structures (which depend on RIF dialects).

**Definition (Logical entailment).** Let \( \phi \) and \( \psi \) be RIF-FLD formulas. We say that \( \phi \) entails \( \psi \), written as \( \phi \models \psi \), if and only if the following holds:

- \( \phi \) and \( \psi \) are both document formulas:
  - for every intended semantic multi-structure \( I \) it is the case that \( TVal(I(\phi)) \preceq TVal(I(\psi)) \).
- \( \phi \) and \( \psi \) are both non-document formulas:
  - for every intended semantic multi-structure \( I \) and for every \( J \in I \) it is the case that \( TVal(I(\phi)) \preceq TVal(I(\psi)) \).

Intuitively, this means that \( \phi \) must entail \( \psi \) in any module the two formulas might happen to be. □

This general notion of entailment covers both first-order logic and the non-monotonic logics that underlie many rule-based languages; it extends the notion of entailment defined in [Shoham87] to the case of multi-valued logics.
4.1 XML for the RIF-FLD Language

RIF-FLD uses [XML1.0] for its XML syntax. The XML serialization for RIF-FLD 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 EBNF of the presentation syntax, such as FORMULA, become XML Schema groups in Appendix XML Schema for FLD. 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.

The RIF serialization framework for the syntax of Section EBNF Grammar for the Presentation Syntax of RIF-FLD uses the following XML tags. While there is a RIF-FLD element tag for the Import directive and an attribute for the Dialect directive, there are none for the Base and Prefix directives: they are handled as discussed in Section Mapping from the RIF-FLD Presentation Syntax to the XML Syntax.

- Document (document, with optional 'dialect' attribute, containing optional directive and payload roles)
- directive (directive role, containing Import)
- payload (payload role, containing Group)
- Import (importation, containing location and optional profile)
- Module (remote module, associating internal name with location)
- location (location role, containing ANYPUNCTINST)
- internal (internal role, containing variable-free term as remote module name)
- profile (profile role, containing PROFILE)
- Group (nested collection of sentences)
- sentence (sentence role, containing FORMULA or Group)
- Forall (quantified formula for 'Forall', containing declare and formula roles)
- Exists (quantified formula for 'Exists', containing declare and formula roles)
- declare (declare role, containing a Var)
The name of a prefix is not associated with an XML element, since it is handled via preprocessing as discussed in Section Mapping of the Non-annotated RIF-FLD Language.

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

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

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>.

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

RIF-FLD also uses the ordered attribute to indicate that the children of args and slot elements are ordered.

Example 5 (Serialization of a nested RIF-FLD group with annotations).

This example shows an XML serialization for the formulas in Example 3. For convenience of reference, the original formulas are included at the top. For better readability, we again use the shortcut syntax defined in [RIF-DTB].

Presentation syntax:

```xml
Document(
  Dialect(FOL)
  Base(<http://www.shakespeare-literature.com/Hamlet/>)
  Prefix(dc <http://purl.org/dc/terms/>)
  Prefix(ex <http://example.org/ontology#>)

  (* <assertions> 
    <assertions>[dc:title->"Hamlet" dc:creator->"Shakespeare"] *)
  Group(
    Exists ?X (And(?X # ex:RottenThing ex:partof(?X <http://www.denmark.dk>))
    Forall ?X (Or(<tobe>(?X) Naf <tobe>(?X)))
    Forall ?X (And(Exists ?B (And(ex:has(?X ?B) ?B # ex:business)))
      Exists ?D (And(ex:has(?X ?D) ?D # ex:desire)))
    :- ?X # ex:man)

  (* <facts> *)
  Group(
    <Yorick> # ex:poor
```
<Hamlet> ex:prince
</Hamlet>

XML serialization:

```xml
<!DOCTYPE Document [
<!ENTITY dc "http://purl.org/dc/terms/">
<!ENTITY ex "http://example.org/ontology#">
<!ENTITY rif "http://www.w3.org/2007/rif#">
<!ENTITY xs "http://www.w3.org/2001/XMLSchema#">
]>

<Document
dialect="FOL">
<payload>
<Group>
<meta>
<Frame>
<object>
<Const type="&rif;iri">assertions</Const>
</object>
<slot ordered="yes">
<Const type="&rif;iri">&dc;title</Const>
<Const type="&xs;string">Hamlet</Const>
</slot>
<slot ordered="yes">
<Const type="&rif;iri">&dc;creator</Const>
<Const type="&xs;string">Shakespeare</Const>
</slot>
</Frame>
</meta>
<sentence>
<Exists>
<declare><Var>X</Var></declare>
<formula>
<And>
<Member>
<instance><Var>X</Var></instance>
<class><Const type="&rif;iri">ex:RottenThing</Const></class>
</Member>
</formula>
<Atom>
<op><Const type="&rif;iri">ex:partof</Const></op>
<args ordered="yes">
<Var>X</Var>
<Const type="&rif;iri">http://www.denmark.dk</Const>
</args>
</Atom>
</formula>
</ Exists>
</sentence>
<sentence>
<Forall>
<declare><Var>X</Var></declare>
<formula>
<Or>
<Atom>
<op><Const type="&rif;iri">tobe</Const></op>
<args ordered="yes">
<Var>X</Var>
</args>
</Atom>
</formula>
<Formula>
<Forall>
<declare><Var>X</Var></declare>
<formula>
<Or>
<Atom>
<op><Const type="&rif;iri">tobe</Const></op>
<args ordered="yes">
<Var>X</Var>
</args>
</Atom>
</formula>
</Forall>
</Formula>
</sentence>
</sentence>
</Forall>
</sentence>
</sentence>
</payload>
</Group>
</Document>
```

RIF Framework for Logic Dialects

W3C Proposed Edited Recommendation 11 December 2012

http://www.w3.org/TR/2012/PER-rif-fld-20121211/
\[ \forall X \left( \text{Member}(X, \text{ex:man}) \implies \right. \]
\[ \left. \begin{align*}
& \exists B \left( \text{Atom} \left( \text{ex:has}(X, B) \land \text{Member}(B, \text{ex:business}) \right) \right) \land \\
& \exists D \left( \text{Atom} \left( \text{ex:has}(X, D) \land \text{Member}(D, \text{ex:desire}) \right) \right) \right) \right) \]
\]
4.2 Mapping from the RIF-FLD Presentation Syntax to the XML Syntax

This section defines a normative mapping, $\text{rif14}^{\text{xml}}$, from the presentation syntax of Section 4.2 Mapping from the RIF-FLD Presentation Syntax to the XML Syntax of RIF-FLD to the XML syntax of RIF-FLD. 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 $\text{rif14}^{\text{xml}}$ to these metavariables. When an expression $\text{rif14}^{\text{xml}}(\text{metavar})$ occurs in an XML pattern in the right column of a translation table, it should be understood as a recursive application of $\text{rif14}^{\text{xml}}$ to the presentation syntax represented by the metavariable. The XML syntax result of such an application is substituted for the expression $\text{rif14}^{\text{xml}}(\text{metavar})$. 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, $?$, to its right.

### 4.2.1 Mapping of the Non-annotated RIF-FLD Language

The $\text{rif14}^{\text{xml}}$ mapping from the presentation syntax to the XML syntax of the non-annotated RIF-FLD Language is given by the table below. Each row indicates a translation $\text{rif14}^{\text{xml}}(\text{Presentation}) = \text{XML}$. The function remove-outer-quotes used in the translation removes enclosing double quotes from a string and leaves unquoted strings untouched. Since the presentation syntax of RIF-FLD is context sensitive, the mapping must differentiate between the terms that occur in the position of the atomic formulas and the terms that occur as atomic formulas. To this end, in the translation table, the positional and named-argument terms that occur in the context of atomic formulas are denoted by expressions of the form $\text{pred}(\ldots)$ and the terms that occur as individuals are denoted by expressions of the form $\text{func}(\ldots)$. In the table, each metavariable for an (unnamed) positional argument is assumed to be instantiated to values unequal to the instantiations of named arguments $\text{namej} \rightarrow \text{filler}$. Regarding the last but first row, we assume that shortcuts for constants [RIF-UTO] have already been expanded to their full form (“...”*sym espacio*). The $\text{AGGRFUNC}$ metavariable stands for any of the aggregation functions Min, Max, Count, Avg, Sum, Prod, Set, Bag, or $\text{NEWAGGRFUNC}$.

Thus, the mapping of the extension point for aggregate functions ($\text{NEWAGGRFUNC}$) is handled by the $\text{AGGRFUNC}$ metavariable, along with the mapping of the specific aggregate functions (Min etc.). The mapping of the extension points for quantifiers ($\text{NEWQUANTIFIER}$) and connectives ($\text{NEWCONNECTIVE}$) generalizes the mapping for the specific quantifiers (Forall, Exists) and connectives (And, Or), respectively. The mapping of the extension point for terms ($\text{NEWTERM}$) keeps $\text{NEWTERM}$ entirely unconstrained in the presentation syntax and uses a wildcard content model (indicated by ellipses) in the XML syntax. This is because the content of $\text{NEWTERM}$ is left entirely up to RIF dialects. Recall that the extension point for symbols ($\text{NEWSYMBOL}$) is part of the alphabet and is not dealt with in the EBNF and XML grammars.

Also recall that $\text{OpenList}(t_1 \ldots t_m, \text{mloc}_1 \ldots \text{mloc}_k)$ is just an alternative form for $\text{List}(t_1 \ldots t_m | t_0)$, so its mapping is not represented separately.

Note that the Import and Directive directives are handled by the presentation-to-XML syntax mapping, using an XML attribute for dialect names (values: FLD, BLD, Core, etc.). On the other hand, the Prefix and Base directives are not handled by this mapping but 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 $\text{rif14}^{\text{xml}}$ applies only to such expanded documents. RIF-FLD 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 5. It replaces prefix names with definitions of XML entities as follows. Each Prefix declaration becomes an ENTITY declaration [XML11] within a DOCTYPE DTD attached to the RIF-FLD 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>
<thead>
<tr>
<th>Presentation Syntax</th>
<th>XML Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Document(</td>
<td>&amp;document;</td>
</tr>
<tr>
<td>Element(name)?</td>
<td>dialect=&quot;name&quot;;</td>
</tr>
<tr>
<td>Import(iiloc; prfl?)</td>
<td>&lt;directive;</td>
</tr>
<tr>
<td>...</td>
<td>&lt;import;</td>
</tr>
<tr>
<td>Module(name; mloc?)</td>
<td>&lt;location&gt;\text{rif14}(iiloc)&lt;/location&gt;;</td>
</tr>
<tr>
<td>Group</td>
<td>&lt;profile&gt;\text{rif14}(prfl?)&lt;/profile&gt;;</td>
</tr>
<tr>
<td></td>
<td>&lt;/import;</td>
</tr>
<tr>
<td></td>
<td>&lt;/directive;</td>
</tr>
<tr>
<td></td>
<td>&lt;directive;</td>
</tr>
<tr>
<td></td>
<td>&lt;import;</td>
</tr>
<tr>
<td></td>
<td>&lt;location&gt;\text{rif14}(iiloc)&lt;/location&gt;;</td>
</tr>
<tr>
<td></td>
<td>&lt;profile&gt;\text{rif14}(prfl?)&lt;/profile&gt;;</td>
</tr>
<tr>
<td></td>
<td>&lt;/import;</td>
</tr>
<tr>
<td></td>
<td>&lt;/directive;</td>
</tr>
<tr>
<td></td>
<td>&lt;directive;</td>
</tr>
<tr>
<td></td>
<td>&lt;import;</td>
</tr>
<tr>
<td></td>
<td>&lt;location&gt;\text{rif14}(iiloc)&lt;/location&gt;;</td>
</tr>
<tr>
<td></td>
<td>&lt;profile&gt;\text{rif14}(prfl?)&lt;/profile&gt;;</td>
</tr>
<tr>
<td></td>
<td>&lt;/import;</td>
</tr>
<tr>
<td></td>
<td>&lt;/directive;</td>
</tr>
<tr>
<td></td>
<td>&lt;directive;</td>
</tr>
<tr>
<td></td>
<td>&lt;import;</td>
</tr>
<tr>
<td></td>
<td>&lt;location&gt;\text{rif14}(mloc)&lt;/location&gt;;</td>
</tr>
<tr>
<td></td>
<td>&lt;profile&gt;\text{rif14}(mloc)&lt;/profile&gt;;</td>
</tr>
<tr>
<td></td>
<td>&lt;/import;</td>
</tr>
<tr>
<td></td>
<td>&lt;/directive;</td>
</tr>
<tr>
<td></td>
<td>&lt;directive;</td>
</tr>
<tr>
<td></td>
<td>&lt;import;</td>
</tr>
<tr>
<td></td>
<td>&lt;location&gt;\text{rif14}(name)&lt;/location&gt;;</td>
</tr>
<tr>
<td></td>
<td>&lt;profile&gt;\text{rif14}(name)&lt;/profile&gt;;</td>
</tr>
<tr>
<td></td>
<td>&lt;/import;</td>
</tr>
<tr>
<td></td>
<td>&lt;/directive;</td>
</tr>
<tr>
<td></td>
<td>&lt;directive;</td>
</tr>
<tr>
<td></td>
<td>&lt;import;</td>
</tr>
<tr>
<td></td>
<td>&lt;location&gt;\text{rif14}(name)&lt;/location&gt;;</td>
</tr>
<tr>
<td></td>
<td>&lt;profile&gt;\text{rif14}(name)&lt;/profile&gt;;</td>
</tr>
<tr>
<td></td>
<td>&lt;/import;</td>
</tr>
<tr>
<td></td>
<td>&lt;/directive;</td>
</tr>
<tr>
<td></td>
<td>&lt;directive;</td>
</tr>
<tr>
<td></td>
<td>&lt;import;</td>
</tr>
<tr>
<td></td>
<td>&lt;location&gt;\text{rif14}(name)&lt;/location&gt;;</td>
</tr>
<tr>
<td></td>
<td>&lt;profile&gt;\text{rif14}(name)&lt;/profile&gt;;</td>
</tr>
<tr>
<td></td>
<td>&lt;/import;</td>
</tr>
</tbody>
</table>
|                     |  }
<table>
<thead>
<tr>
<th><strong>RIF Framework for Logic Dialects</strong></th>
<th><strong>W3C Proposed Edited Recommendation 11 December 2012</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Group</strong></td>
<td><code>&lt;Group&gt;</code></td>
</tr>
<tr>
<td><em>clause1</em></td>
<td><code>&lt;sentence&gt;</code>χ fld(clause1)<code>&lt;/sentence&gt;</code></td>
</tr>
<tr>
<td><em>clause2</em></td>
<td><code>&lt;sentence&gt;</code>χ fld(clause2)<code>&lt;/sentence&gt;</code></td>
</tr>
<tr>
<td><em>clause2</em></td>
<td><code>&lt;/Group&gt;</code></td>
</tr>
<tr>
<td><strong>Forall</strong></td>
<td><code>&lt;Forall&gt;</code></td>
</tr>
<tr>
<td><em>variable1</em></td>
<td><code>&lt;declare&gt;</code>χ fld(variable1)<code>&lt;/declare&gt;</code></td>
</tr>
<tr>
<td><em>variable2</em></td>
<td><code>&lt;declare&gt;</code>χ fld(variable2)<code>&lt;/declare&gt;</code></td>
</tr>
<tr>
<td><em>variable3</em></td>
<td><code>&lt;declare&gt;</code>χ fld(variable3)<code>&lt;/declare&gt;</code></td>
</tr>
<tr>
<td><em>body</em></td>
<td><code>&lt;formula&gt;</code>χ fld(body)<code>&lt;/formula&gt;</code></td>
</tr>
<tr>
<td><code>n ≥ 0</code></td>
<td><code>&lt;/Forall&gt;</code></td>
</tr>
<tr>
<td><strong>Exists</strong></td>
<td><code>&lt;Exists&gt;</code></td>
</tr>
<tr>
<td><em>variable1</em></td>
<td><code>&lt;declare&gt;</code>χ fld(variable1)<code>&lt;/declare&gt;</code></td>
</tr>
<tr>
<td><em>variable2</em></td>
<td><code>&lt;declare&gt;</code>χ fld(variable2)<code>&lt;/declare&gt;</code></td>
</tr>
<tr>
<td><em>variable3</em></td>
<td><code>&lt;declare&gt;</code>χ fld(variable3)<code>&lt;/declare&gt;</code></td>
</tr>
<tr>
<td><em>body</em></td>
<td><code>&lt;formula&gt;</code>χ fld(body)<code>&lt;/formula&gt;</code></td>
</tr>
<tr>
<td><code>n ≥ 0</code></td>
<td><code>&lt;/Exists&gt;</code></td>
</tr>
<tr>
<td><strong>NEWQUANTIFIER</strong></td>
<td><code>&lt;NEWQUANTIFIER&gt;</code></td>
</tr>
<tr>
<td><em>variable1</em></td>
<td><code>&lt;declare&gt;</code>χ fld(variable1)<code>&lt;/declare&gt;</code></td>
</tr>
<tr>
<td><em>variable2</em></td>
<td><code>&lt;declare&gt;</code>χ fld(variable2)<code>&lt;/declare&gt;</code></td>
</tr>
<tr>
<td><em>variable3</em></td>
<td><code>&lt;declare&gt;</code>χ fld(variable3)<code>&lt;/declare&gt;</code></td>
</tr>
<tr>
<td><em>body</em></td>
<td><code>&lt;formula&gt;</code>χ fld(body)<code>&lt;/formula&gt;</code></td>
</tr>
<tr>
<td><code>n ≥ 0</code></td>
<td><code>&lt;/NEWQUANTIFIER&gt;</code></td>
</tr>
<tr>
<td><strong>conclusion</strong></td>
<td><code>&lt;Implies&gt;</code></td>
</tr>
<tr>
<td><em>condition</em></td>
<td><code>&lt;if&gt;</code>χ fld(condition)<code>&lt;/if&gt;</code></td>
</tr>
<tr>
<td><em>conclusion</em></td>
<td><code>&lt;then&gt;</code>χ fld(conclusion)<code>&lt;/then&gt;</code></td>
</tr>
<tr>
<td><code>&lt;/Implies&gt;</code></td>
<td></td>
</tr>
<tr>
<td><strong>And</strong></td>
<td><code>&lt;And&gt;</code></td>
</tr>
<tr>
<td><em>conjunct1</em></td>
<td><code>&lt;formula&gt;</code>χ fld(conjunct1)<code>&lt;/formula&gt;</code></td>
</tr>
<tr>
<td><em>conjunct2</em></td>
<td><code>&lt;formula&gt;</code>χ fld(conjunct2)<code>&lt;/formula&gt;</code></td>
</tr>
<tr>
<td><em>conjunct3</em></td>
<td><code>&lt;formula&gt;</code>χ fld(conjunct3)<code>&lt;/formula&gt;</code></td>
</tr>
<tr>
<td><code>n ≥ 0</code></td>
<td><code>&lt;/And&gt;</code></td>
</tr>
<tr>
<td><strong>Or</strong></td>
<td><code>&lt;Or&gt;</code></td>
</tr>
<tr>
<td><em>disjunct1</em></td>
<td><code>&lt;formula&gt;</code>χ fld(disjunct1)<code>&lt;/formula&gt;</code></td>
</tr>
<tr>
<td><em>disjunct2</em></td>
<td><code>&lt;formula&gt;</code>χ fld(disjunct2)<code>&lt;/formula&gt;</code></td>
</tr>
<tr>
<td><em>disjunct3</em></td>
<td><code>&lt;formula&gt;</code>χ fld(disjunct3)<code>&lt;/formula&gt;</code></td>
</tr>
<tr>
<td><code>n ≥ 0</code></td>
<td><code>&lt;/Or&gt;</code></td>
</tr>
<tr>
<td><strong>NEWCONNECTIVE</strong></td>
<td><code>&lt;NEWCONNECTIVE&gt;</code></td>
</tr>
<tr>
<td><em>argument1</em></td>
<td><code>&lt;formula&gt;</code>χ fld(argument1)<code>&lt;/formula&gt;</code></td>
</tr>
<tr>
<td><em>argument2</em></td>
<td><code>&lt;formula&gt;</code>χ fld(argument2)<code>&lt;/formula&gt;</code></td>
</tr>
<tr>
<td><em>argument3</em></td>
<td><code>&lt;formula&gt;</code>χ fld(argument3)<code>&lt;/formula&gt;</code></td>
</tr>
<tr>
<td><code>n ≥ 0</code></td>
<td><code>&lt;/NEWCONNECTIVE&gt;</code></td>
</tr>
<tr>
<td><strong>Neg</strong></td>
<td><code>&lt;Neg&gt;</code></td>
</tr>
<tr>
<td><em>form</em></td>
<td><code>&lt;formula&gt;</code>χ fld(form)<code>&lt;/formula&gt;</code></td>
</tr>
<tr>
<td><code>&lt;/Neg&gt;</code></td>
<td></td>
</tr>
<tr>
<td><strong>Naf</strong></td>
<td><code>&lt;Naf&gt;</code></td>
</tr>
<tr>
<td><em>form</em></td>
<td><code>&lt;formula&gt;</code>χ fld(form)<code>&lt;/formula&gt;</code></td>
</tr>
<tr>
<td><code>&lt;/Naf&gt;</code></td>
<td></td>
</tr>
<tr>
<td>query @ modref</td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>(&lt;Remote&gt;)</td>
<td>(\chi_{\text{id}}(\text{query})&lt;/\text{formula}&gt;)</td>
</tr>
<tr>
<td>(\text{&lt;internal&gt;\text{id}(modref)&lt;/internal&gt;})</td>
<td>(&lt;/Remote&gt;)</td>
</tr>
</tbody>
</table>

| External ( atomframexpr ) | 
|---|---|
| \(<\text{External}\) | \(\chi_{\text{id}(atomframexpr)}</\text{content}>\) |
| \(</\text{External}\>\) | \(</\text{External}\>\) |

| pred ( ) | 
|---|---|
| \(<\text{Atom}\) | \(\chi_{\text{id}(pred)}</\text{op}>\) |
| \(</\text{Atom}\>\) | \(</\text{Atom}\>\) |

| pred ( argument\_1, \ldots, argument\_m ) | 
|---|---|
| \(<\text{Atom}\) | \(\chi_{\text{id}(pred)}</\text{op}>\) |
| \(\text{<args ordered="yes">\chi_{\text{id}(argument\_1)}\ldots\chi_{\text{id}(argument\_m)}</args></\text{Atom}>\) | \(</\text{Atom}>\) |

| func ( ) | 
|---|---|
| \(<\text{Expr}\) | \(\chi_{\text{id}(func)}</\text{op}>\) |
| \(</\text{Expr}\>\) | \(</\text{Expr}\>\) |

| func ( argument\_1, \ldots, argument\_m ) | 
|---|---|
| \(<\text{Expr}\) | \(\chi_{\text{id}(func)}</\text{op}>\) |
| \(\text{<args ordered="yes">\chi_{\text{id}(argument\_1)}\ldots\chi_{\text{id}(argument\_m)}</args></\text{Expr}>\) | \(</\text{Expr}>\) |

| List ( element\_1, \ldots, element\_n ) | 
|---|---|
| \(<\text{List}\) | \(\chi_{\text{id}(element\_1)}\ldots\chi_{\text{id}(element\_n)}</\text{items}>) |
| \(</\text{List}>\) | \(</\text{List}>\) |

<table>
<thead>
<tr>
<th>List ( element_1, \ldots, element_n</th>
<th>remainder )</th>
</tr>
</thead>
<tbody>
<tr>
<td>(&lt;\text{List})</td>
<td>(\chi_{\text{id}(element_1)}\ldots\chi_{\text{id}(element_n)}&lt;/\text{items}&gt;)</td>
</tr>
<tr>
<td>(&lt;\text{rest}&gt;\chi_{\text{id}(\text{remainder})}&lt;/\text{rest}&gt;)</td>
<td>(&lt;/\text{List}&gt;)</td>
</tr>
</tbody>
</table>

| pred ( name\_1 \rightarrow filler\_1, \ldots, name\_n \rightarrow filler\_n ) | 
|---|---|
| \(<\text{Atom}\) | \(\chi_{\text{id}(pred)}</\text{op}>\) |
| \(<\text{slot ordered="yes">\chi_{\text{id}(name\_1)}\text{Name}\chi_{\text{id}(filler\_1)}\text{Name}</\text{slot}>\) | \(</\text{Atom}>\) |

| func ( name\_1 \rightarrow filler\_1, \ldots, name\_n \rightarrow filler\_n ) | 
|---|---|
| \(<\text{Expr}\) | \(\chi_{\text{id}(func)}</\text{op}>\) |
| \(<\text{slot ordered="yes">\chi_{\text{id}(name\_1)}\text{Name}\chi_{\text{id}(filler\_1)}\text{Name}</\text{slot}>\) | \(</\text{Expr}>\) |
### 4.2.2 Mapping of RIF-FLD Annotations

The $\chi$ld mapping from RIF-FLD annotations in the presentation syntax to the XML syntax is specified by the table below. It extends the translation table of Section Mapping of the Non-annotated RIF-FLD Language. The metavariable Typetag in the presentation and XML syntaxes stands for any of the class names And, Or, External, Document, or Group, Quantifier for Exists or Forall, and Negation for Neg or Naf. The dollar sign, $\$, stands for any of the binary infix operator names #, ##, =, :-, or @, while Binop stands for their respective class names Member, Subclass, Equal, Implies, or Remote. The metavariable attr? is used with Typetag to capture the optional dialect attribute (with its value) of Document. Again, each metavariable for an (unnamed) positional argument $name_i$ is assumed to be instantiated to values unequal to the instantiations of named arguments $name_j \rightarrow filler_j$.

<table>
<thead>
<tr>
<th>Presentation Syntax</th>
<th>XML Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>(* const? frameconj? *) Typetag ( $e_1$ . . . $e_n$ ) n ≥ 0</td>
<td>(&lt;Typetag attr?&gt; $id$=$\chi$ld(const)?$)&lt;/id&gt;&lt;meta&gt;$\chi$ld(frameconj)?$&lt;/meta&gt;$e_1$' . . . $e_n$'</td>
</tr>
</tbody>
</table>
where $attr, e_1', \ldots, e_n'$ are defined by the equation
\[
\chi_{\text{fld}}(\text{Typetag}(e_1' \ldots e_n')) = <\text{Typetag} attr? e_1' \ldots e_n'</Typetag>
\]
5 Conformance of RIF Processors with RIF Dialects

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

Let $\mathcal{T}$ be a set of datatypes and symbol spaces that includes the datatypes specified in [RIF-DTB] and the symbol spaces $\text{rif:iri}$ and $\text{rif:local}$. Let $\mathcal{E}$ be a coherent set of external schemas that includes the built-ins listed in [RIF-DTB]. Let $\mathcal{D}$ be a RIF dialect (e.g., [RIF-BLD]). We say that a formula $\varphi$ is a $\mathcal{D}_{\mathcal{T},\mathcal{E}}$ formula iff

- it is a formula in the dialect $\mathcal{D}$,
- all datatypes and symbol spaces used in $\varphi$ are in $\mathcal{T}$, and
- all externally defined terms used in $\varphi$ are instantiations of some external schemas in $\mathcal{E}$.

A RIF processor is a conformant $\mathcal{D}_{\mathcal{T},\mathcal{E}}$ consumer iff it implements a semantics-preserving mapping, $\mu$, from the set of all $\mathcal{D}_{\mathcal{T},\mathcal{E}}$ formulas to the language $\mathcal{L}$ of the processor.
Formally, this means that for any pair φ, ψ of $D_{T,F}$ formulas for which $φ \models_D ψ$ is defined, $φ \models_D ψ$ if and only if $μ(φ) \models_L ψ$. Here $\models_D$ denotes the logical entailment in the RIF dialect $D$ and $\models_L$ is the logical entailment in the language $L$ of the RIF processor.

A RIF processor is a conformant $D_{T,F}$ producer iff it implements a semantics-preserving mapping, $ν$, from the language $L$ of the processor to the set of all $D_{T,F}$ formulas.

Formally, this means that for any pair φ, ψ of formulas in $L$ for which $φ \models_L ψ$ is defined, $φ \models_L ψ$ if and only if $ν(φ) \models_D ν(ψ)$.

An admissible document in a logic RIF dialect $D$ is one which conforms to all the syntactic constraints of $D$, including the ones that cannot be checked by an XML Schema validator (see Definition Admissible XML document in a logic dialect).

6 Acknowledgements

This revised version incorporates a number of improvements suggested in [DAA] and fixes errors in the definition of the semantics of document formulas pointed out in that work.

7 References

7.1 Normative References


7.2 Informational References

8 Appendix: XML Schema for RIF-FLD

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

XML schemas for the RIF-FLD language are defined below and are also available at http://www.w3.org/2010/rif-schema/fld with additional examples. For modularity, we define a Baseline schema and a Skyline schema. Baseline is the schema module that provides the foundation up to FORMULAs without Implies. Skyline provides the full schema by augmenting Baseline with the Implies FORMULA as well as with Group and Document.

8.1 Baseline Schema Module

```xml
<?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: FLDBaseline.xsd, v. 1.5, 2010-05-08, hboley/dhirtle">
    schemaLocation='http://www.w3.org/2001/xml.xsd'/>
  <xs:annotation>
    <xs:documentation>
    This is the Baseline module of FLD. It is the foundation of the full schema defined through the Skyline module. The Baseline XML schema is based on the following EBNF (compared to the full EBNF of RIF-FLD, Group and Document are omitted, and 'Implies' is missing from the productions for FORMULA and TERMULA).
    The nonterminals starting with NEW provide extensions points for FLD (cf. Section 4 XML Serialization Framework).
    </xs:documentation>
  </xs:annotation>
  <xs:element name="FORMULA"
    type="xs:complexType">
    <xs:complexType>
      <xs:choice>
        <xs:element name="IRIMETA" type="xs:string"/>
        <xs:element name="CONNECTIVE" type="xs:string"/>
        <xs:element name="TERMULA" type="xs:string"/>
      </xs:choice>
    </xs:complexType>
  </xs:element>
  <xs:element name="FORM" type="xs:string"/>
  <xs:element name="ATOMIC" type="xs:string"/>
  <xs:element name="TERM" type="xs:string"/>
  <xs:element name="EXPRIC" type="xs:string"/>
  <xs:element name="AGGREGATE" type="xs:string"/>
  <xs:element name="CONNECTIVE" type="xs:string"/>
  <xs:element name="QUANTIFIER" type="xs:string"/>
  <xs:element name="AGGRFUNC" type="xs:string"/>
</xs:schema>
```

RIF Framework for Logic Dialects

W3C Proposed Edited Recommendation 11 December 2012

http://www.w3.org/TR/2012/PER-rif-fld-20121211/
SYMSPACE ::= ANGLEBRACKIRI | CURIE
LOCATOR ::= ANGLEBRACKIRI
Var ::= '?' Name
Name ::= NCName | '"' UNICODESTRING '"'
IRIMETA ::= '(*' Const? (Frame | 'And' '(' Frame* ')')? '*)'

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

<xs:group name="FORMULA">
 <!-- 'Implies' omitted from Baseline schema, allowing its modular use

FORMULA ::= IRIMETA? CONNECTIVE '(' FORMULA* ')' |
IRIMETA? QUANTIFIER '(' FORMULA ')' |
IRIMETA? 'Neg' FORMULA |
IRIMETA? 'Naf' FORMULA |
IRIMETA? FORMULA '@' MODULEREF
FORM
CONNECTIVE ::= 'And' | 'Or' | NEWCONNECTIVE
QUANTIFIER ::= ('Exists' | 'Forall' | NEWQUANTIFIER) Var*
rewritten as
FORM
FORM
FORM
FORM
FORM
FORM

</xs:choice>
</xs:group>

<xs:complexType name="And-FORMULA.type">
 <!-- sensitive to FORMULA context-->
<xs:sequence>
<xs:group ref="IRIMETA" minOccurs="0" maxOccurs="1"/>
<xs:element name="formula" minOccurs="0" maxOccurs="unbounded"/>
</xs:sequence>
</xs:complexType>

<xs:complexType name="Or-FORMULA.type">
 <!-- sensitive to FORMULA context-->
<xs:sequence>
<xs:group ref="IRIMETA" minOccurs="0" maxOccurs="1"/>
<xs:element name="formula" minOccurs="0" maxOccurs="unbounded"/>
</xs:sequence>
</xs:complexType>

<xs:complexType name="NEWCONNECTIVE-FORMULA.type">
 <!-- sensitive to FORMULA context-->
<xs:sequence>
<xs:group ref="IRIMETA" minOccurs="0" maxOccurs="1"/>
<xs:element name="formula" minOccurs="0" maxOccurs="unbounded"/>
</xs:sequence>
</xs:complexType>

<xs:complexType name="Exists-FORMULA.type">
 <!-- sensitive to FORMULA context-->
<xs:sequence>
<xs:group ref="IRIMETA" minOccurs="0" maxOccurs="1"/>
<xs:element ref="declare" minOccurs="0" maxOccurs="unbounded"/>
<xs:element ref="formula"/>
</xs:sequence>
</xs:complexType>

<xs:complexType name="Forall-FORMULA.type">
 <!-- sensitive to FORMULA context-->
<xs:sequence>
<xs:group ref="IRIMETA" minOccurs="0" maxOccurs="1"/>
<xs:element ref="formula" minOccurs="0" maxOccurs="unbounded"/>
</xs:sequence>
</xs:complexType>

<xs:complexType name="NEWQUANTIFIER-FORMULA.type">
 <!-- sensitive to FORMULA context-->
<xs:sequence>
<xs:group ref="IRIMETA" minOccurs="0" maxOccurs="1"/>
<xs:element name="formula" minOccurs="0" maxOccurs="unbounded"/>
</xs:sequence>
</xs:complexType>

<xs:complexType name="Neg-FORMULA.type">
 <!-- sensitive to FORMULA context-->
<xs:sequence>
<xs:group ref="IRIMETA" minOccurs="0" maxOccurs="1"/>
<xs:element ref="formula" minOccurs="0" maxOccurs="unbounded"/>
</xs:sequence>
</xs:complexType>

<xs:complexType name="Naf-FORMULA.type">
 <!-- sensitive to FORMULA context-->
<xs:sequence>
<xs:group ref="IRIMETA" minOccurs="0" maxOccurs="1"/>
<xs:element ref="formula" minOccurs="0" maxOccurs="unbounded"/>
</xs:sequence>
</xs:complexType>

<xs:complexType name="Remote-FORMULA.type">
 <!-- sensitive to FORMULA context-->
<xs:sequence>
<xs:group ref="IRIMETA" minOccurs="0" maxOccurs="1"/>
<xs:element ref="formula" minOccurs="0" maxOccurs="unbounded"/>
</xs:sequence>
</xs:complexType>

<xs:complexType name="Form-FORMULA.type">
 <!-- sensitive to FORMULA context-->
<xs:sequence>
<xs:group ref="IRIMETA" minOccurs="0" maxOccurs="1"/>
<xs:element ref="formula" minOccurs="0" maxOccurs="unbounded"/>
</xs:sequence>
</xs:complexType>

</xs:choice>
</xs:group>

<xs:complexType name="And-FORMULA.type">
 <!-- sensitive to FORMULA context-->
<xs:sequence>
<xs:group ref="IRIMETA" minOccurs="0" maxOccurs="1"/>
<xs:element name="formula" minOccurs="0" maxOccurs="unbounded"/>
</xs:sequence>
</xs:complexType>

<xs:complexType name="Or-FORMULA.type">
 <!-- sensitive to FORMULA context-->
<xs:sequence>
<xs:group ref="IRIMETA" minOccurs="0" maxOccurs="1"/>
<xs:element name="formula" minOccurs="0" maxOccurs="unbounded"/>
</xs:sequence>
</xs:complexType>

<xs:complexType name="NEWCONNECTIVE-FORMULA.type">
 <!-- sensitive to FORMULA context-->
<xs:sequence>
<xs:group ref="IRIMETA" minOccurs="0" maxOccurs="1"/>
<xs:element name="formula" minOccurs="0" maxOccurs="unbounded"/>
</xs:sequence>
</xs:complexType>

<xs:complexType name="Exists-FORMULA.type">
 <!-- sensitive to FORMULA context-->
<xs:sequence>
<xs:group ref="IRIMETA" minOccurs="0" maxOccurs="1"/>
<xs:element ref="declare" minOccurs="0" maxOccurs="unbounded"/>
<xs:element ref="formula"/>
</xs:sequence>
</xs:complexType>

<xs:complexType name="Forall-FORMULA.type">
 <!-- sensitive to FORMULA context-->
<xs:sequence>
<xs:group ref="IRIMETA" minOccurs="0" maxOccurs="1"/>
<xs:element ref="formula" minOccurs="0" maxOccurs="unbounded"/>
</xs:sequence>
</xs:complexType>

<xs:complexType name="NEWQUANTIFIER-FORMULA.type">
 <!-- sensitive to FORMULA context-->
<xs:sequence>
<xs:group ref="IRIMETA" minOccurs="0" maxOccurs="1"/>
<xs:element name="formula" minOccurs="0" maxOccurs="unbounded"/>
</xs:sequence>
</xs:complexType>

<xs:complexType name="Neg-FORMULA.type">
 <!-- sensitive to FORMULA context-->
<xs:sequence>
<xs:group ref="IRIMETA" minOccurs="0" maxOccurs="1"/>
<xs:element ref="formula" minOccurs="0" maxOccurs="unbounded"/>
</xs:sequence>
</xs:complexType>

<xs:complexType name="Naf-FORMULA.type">
 <!-- sensitive to FORMULA context-->
<xs:sequence>
<xs:group ref="IRIMETA" minOccurs="0" maxOccurs="1"/>
<xs:element ref="formula" minOccurs="0" maxOccurs="unbounded"/>
</xs:sequence>
</xs:complexType>

<xs:complexType name="Remote-FORMULA.type">
 <!-- sensitive to FORMULA context-->
<xs:sequence>
<xs:group ref="IRIMETA" minOccurs="0" maxOccurs="1"/>
<xs:element ref="formula" minOccurs="0" maxOccurs="unbounded"/>
<xs:element name="declare">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="Var"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:complexType name="NEWQUANTIFIER-FORMULA.type">
  <!-- sensitive to FORMULA context-->
  <xs:sequence>
    <xs:group ref="IRIMETA" minOccurs="0" maxOccurs="1"/>
    <xs:element ref="declare" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element ref="formula"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="Neg-FORMULA.type">
  <!-- sensitive to FORMULA context-->
  <xs:sequence>
    <xs:group ref="IRIMETA" minOccurs="0" maxOccurs="1"/>
    <xs:element ref="formula" minOccurs="1" maxOccurs="1"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="Naf-FORMULA.type">
  <!-- sensitive to FORMULA context-->
  <xs:sequence>
    <xs:group ref="IRIMETA" minOccurs="0" maxOccurs="1"/>
    <xs:element ref="formula" minOccurs="1" maxOccurs="1"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="Remote-FORMULA.type">
  <!-- sensitive to FORMULA context-->
  <xs:sequence>
    <xs:group ref="IRIMETA" minOccurs="0" maxOccurs="1"/>
    <xs:element ref="formula"/>
    <xs:element ref="internal"/>
  </xs:sequence>
</xs:complexType>

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

<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="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="FORM">
  <!-- FORM ::= IRIMETA? (Var | ATOMIC | 'External' '(' ATOMIC LOCATOR? ')') -->
</xs:group>
<xs:complexType name="External-FORM.type">
  <!-- sensitive to FORM (ATOMIC) context-->
  <xs:sequence>
    <xs:group ref="IRIMETA" minOccurs="0" maxOccurs="1"/>
    <xs:element name="content" type="content-FORM.type"/>
    <xs:element ref="location" minOccurs="0" maxOccurs="1"/>
  </xs:sequence>
</xs:complexType>

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

<xs:group name="ATOMIC">
  <!--
  ATOMIC ::= Const | Atom | Equal | Member | Subclass | Frame
  -->
  <xs:choice>
    <xs:element ref="Const"/>
    <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 ::= TERMULA '(' (TERMULA* | (Name '->' TERMULA)*) ')'
  -->
  <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" maxOccurs="unbounded"/>
    </xs:choice>
  </xs:sequence>
</xs:group>

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

<xs:element name="args">
  <xs:complexType>
    <xs:sequence>
      <xs:group ref="TERMULA" minOccurs="1" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:complexType name="slot-UNITERM.type">
  <!-- sensitive to UNITERM (Name) context-->
  <xs:sequence>
    <xs:element ref="Name"/>
    <xs:group ref="TERMULA"/>
  </xs:sequence>
</xs:complexType>
<xs:sequence>
  <xs:group ref="TERMULA"/>
</xs:sequence>
</xs:complexType>
</xs:element>

<xs:element name="Frame">
  <!--
  Frame ::= TERMULA '{' (TERMULA '->' TERMULA)* '}'
  -->
  <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" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

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

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

<xs:group name="TERMULA">
  <!-- 'Implies' omitted from Baseline schema, allowing its modular use
  TERMULA ::= IRIMETA? CONNECTIVE '{' (TERMULA '->' TERMULA)* '}'
  IRIMETA? QUANTIFIER '{' TERMULA '}' |
  IRIMETA? 'Neg' TERMULA |
  IRIMETA? 'Naf' TERMULA |
  IRIMETA? TERMULA '@' MODULEREF |
  TERM
  CONNECTIVE ::= 'And' | 'Or' | NEWCONNECTIVE
  QUANTIFIER ::= ("Exists" | "Forall" | NEWQUANTIFIER) Var*
 rewritten as
  TERMULA ::= IRIMETA? 'And' '{' TERMULA '}' |
  IRIMETA? 'Or' '{' TERMULA '}' |
  IRIMETA? 'NEWCONNECTIVE' '{' TERMULA '}' |
  IRIMETA? 'Exists' Var* '{' TERMULA '}' |
  IRIMETA? 'Forall' Var* '{' TERMULA '}' |
  IRIMETA? 'NEWQUANTIFIER' Var* '{' TERMULA '}' |
  IRIMETA? 'Neg' TERMULA |
  IRIMETA? 'Naf' TERMULA |
  IRIMETA? 'Remote' '{' TERMULA MODULEREF '}'
  TERM
  -->
  <xs:choice>
    <xs:element name="And" type="And-TERMULA.type"/>
    <xs:element name="Or" type="Or-TERMULA.type"/>
    <xs:element name="NEWCONNECTIVE" type="NEWCONNECTIVE-TERMULA.type"/>
    <xs:element name="Exists" type="Exists-TERMULA.type"/>
    <xs:element name="Forall" type="Forall-TERMULA.type"/>
    <xs:element name="NEWQUANTIFIER" type="NEWQUANTIFIER-TERMULA.type"/>
    <xs:element name="Neg" type="Neg-TERMULA.type"/>
    <xs:element name="Naf" type="Naf-TERMULA.type"/>
    <xs:element name="Remote" type="Remote-TERMULA.type"/>
  </xs:choice>
</xs:group>

<xs:complexType name="And-TERMULA.type">
  <!-- sensitive to TERMULA context-->
  <xs:sequence>
    <xs:group ref="IRIMETA" minOccurs="0" maxOccurs="1"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="Or-TERMULA.type">
  <!-- sensitive to TERMULA context-->
  <xs:sequence>
    <xs:group ref="IRIMETA" minOccurs="0" maxOccurs="1"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="NEWCONNECTIVE-TERMULA.type">
  <!-- sensitive to TERMULA context-->
  <xs:sequence>
    <xs:group ref="IRIMETA" minOccurs="0" maxOccurs="1"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="Exists-TERMULA.type">
  <!-- sensitive to TERMULA context-->
  <xs:sequence>
    <xs:group ref="IRIMETA" minOccurs="0" maxOccurs="1"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="Forall-TERMULA.type">
  <!-- sensitive to TERMULA context-->
  <xs:sequence>
    <xs:group ref="IRIMETA" minOccurs="0" maxOccurs="1"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="NEWQUANTIFIER-TERMULA.type">
  <!-- sensitive to TERMULA context-->
  <xs:sequence>
    <xs:group ref="IRIMETA" minOccurs="0" maxOccurs="1"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="Neg-TERMULA.type">
  <!-- sensitive to TERMULA context-->
  <xs:sequence>
    <xs:group ref="IRIMETA" minOccurs="0" maxOccurs="1"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="Naf-TERMULA.type">
  <!-- sensitive to TERMULA context-->
  <xs:sequence>
    <xs:group ref="IRIMETA" minOccurs="0" maxOccurs="1"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="Remote-TERMULA.type">
  <!-- sensitive to TERMULA context-->
  <xs:sequence>
    <xs:group ref="IRIMETA" minOccurs="0" maxOccurs="1"/>
  </xs:sequence>
</xs:complexType>
<xs:element name="termula" minOccurs="0" maxOccurs="unbounded"/>
</xs:sequence>
</xs:complexType>
<xs:complexType name="NEWCONNECTIVE-TERMULA.type">
<!-- sensitive to TERMULA context-->
<xs:sequence>
  <xs:group ref="IRIMETA" minOccurs="0" maxOccurs="1"/>
  <xs:element name="termula" minOccurs="0" maxOccurs="unbounded"/>
</xs:sequence>
</xs:complexType>
<xs:complexType name="Exists-TERMULA.type">
<!-- sensitive to TERMULA context-->
<xs:sequence>
  <xs:group ref="IRIMETA" minOccurs="0" maxOccurs="1"/>
  <xs:element ref="declare" minOccurs="0" maxOccurs="unbounded"/>
  <xs:element ref="termula"/>
</xs:sequence>
</xs:complexType>
<xs:complexType name="Forall-TERMULA.type">
<!-- sensitive to TERMULA context-->
<xs:sequence>
  <xs:group ref="IRIMETA" minOccurs="0" maxOccurs="1"/>
  <xs:element ref="declare" minOccurs="0" maxOccurs="unbounded"/>
  <xs:element ref="termula"/>
</xs:sequence>
</xs:complexType>
<xs:complexType name="NEWQUANTIFIER-TERMULA.type">
<!-- sensitive to TERMULA context-->
<xs:sequence>
  <xs:group ref="IRIMETA" minOccurs="0" maxOccurs="1"/>
  <xs:element ref="declare" minOccurs="0" maxOccurs="unbounded"/>
  <xs:element ref="termula"/>
</xs:sequence>
</xs:complexType>
<xs:complexType name="Neg-TERMULA.type">
<!-- sensitive to TERMULA context-->
<xs:sequence>
  <xs:group ref="IRIMETA" minOccurs="0" maxOccurs="1"/>
  <xs:element ref="termula" minOccurs="1" maxOccurs="1"/>
</xs:sequence>
</xs:complexType>
<xs:complexType name="Naf-TERMULA.type">
<!-- sensitive to TERMULA context-->
<xs:sequence>
  <xs:group ref="IRIMETA" minOccurs="0" maxOccurs="1"/>
  <xs:element ref="termula" minOccurs="1" maxOccurs="1"/>
</xs:sequence>
</xs:complexType>
<xs:complexType name="Remote-TERMULA.type">
<!-- sensitive to TERMULA context-->
<xs:sequence>
  <xs:group ref="IRIMETA" minOccurs="0" maxOccurs="1"/>
  <xs:element ref="termula"/>
  <xs:element ref="internal"/>
</xs:sequence>
</xs:complexType>
<xs:element name="termula">
</xs:complexType>
<xs:group name="TERM">
<!--
TERM ::= IRIMETA? (Var | EXPRIC | List | 'External' '(' EXPRIC LOCATOR? ')' | AGGREGATE | NEWTERM)
-->
<xs:choice>
  <xs:element ref="Var"/>
  <xs:group ref="EXPRIC"/>
  <xs:element ref="List"/>
  <xs:element ref="External" type="External-TERM.type"/>
  <xs:element ref="AGGREGATE"/>
  <xs:element ref="NEWTERM"/>
</xs:choice>
`List` ::= 'List' '(' `TERM`+ ')' | 'List' '(' `TERM` | `TERM` ')'
rewritten as
`List` ::= 'List' '(' `LISTELEMENTS`? ')

`LISTELEMENTS` ::= TERM+ ('|' TERM)?

---

`items`:

`items`: as
`items`:

---

`rest`:

`rest`:

---

`content`: as

---

`EXPRIC`:

`EXPRIC` as

---

`Expr`:

`Expr` as

---
<xs:element name="AGGREGATE" abstract="true">
  <!--
  AGGREGATE ::= AGGRFUNC '{' Var '|' Var+ '}'? '| FORMULA ')
  AGGRFUNC ::= 'Min' | 'Max' | 'Sum' | 'Prod' | 'Avg' | 'Count' |
  'Set' | 'Bag' | NEWAGGRFUNC
  -->
  <xs:complexType>
    <xs:sequence>
      <xs:group ref="IRIMETA" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="declare" minOccurs="2" maxOccurs="unbounded"/>
      <xs:element ref="formula"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="Min" substitutionGroup="AGGREGATE"/>
<xs:element name="Max" substitutionGroup="AGGREGATE"/>
<xs:element name="Sum" substitutionGroup="AGGREGATE"/>
<xs:element name="Prod" substitutionGroup="AGGREGATE"/>
<xs:element name="Avg" substitutionGroup="AGGREGATE"/>
<xs:element name="Count" substitutionGroup="AGGREGATE"/>
<xs:element name="Set" substitutionGroup="AGGREGATE"/>
<xs:element name="Bag" substitutionGroup="AGGREGATE"/>
<xs:element name="NEWAGGRFUNC" substitutionGroup="AGGREGATE"/>

<xs:element name="NEWTERM">
  <!--
  This uses the XSD wildcard schema component, any, allowing a NEWTERM
to have zero or more child elements (role tags).
  -->
  <xs:complexType>
    <xs:sequence>
      <xs:any processContents="skip" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="Const">
  <!--
  Const ::= '' UNICODESTRING ''^^ SYMSPACE | CONSTSHORT
  -->
  <xs:complexType mixed="true">
    <xs:sequence>
      <xs:group ref="IRIMETA" minOccurs="0" maxOccurs="1"/>
    </xs:sequence>
    <xs:attribute name="type" type="xs:anyURI" use="required"/>
    <xs:attribute ref="xml:lang"/>
  </xs:complexType>
</xs:element>

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

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

<xs:group name="IRIMETA">
  <!--
  IRIMETA ::= ('*' Const? (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:element ref="Const"/>
  </xs:complexType>
</xs:element>

<xs:element name="meta">
  <xs:complexType>
    <xs:element ref="Const"/>
  </xs:complexType>
</xs:element>
8.2 Skyline Schema Module

<?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#">
  <xs:annotation>
    <xs:documentation>
      This is the Skyline schema module of FLD. It is split off from the Baseline
      schema for modularity. The Skyline XML schema is based on the following EBNF
      (which adds Group and Document, and brings 'Implies' into FORMULA and TERMULA):

      Dialect ::= 'Dialect' ('Name')
      Base ::= 'Base' ('ANGLEBRACKIRI')
      Prefix ::= 'Prefix' ('NCName ANGLEBRACKIRI')
      Import ::= IRIMETA? 'Import' ('LOCATOR PROFILE?')
      Module ::= IRIMETA? 'Module' ('(Const | Expr) LOCATOR')
      Group ::= IRIMETA? 'Group' ('(FORMULA | Group)*')
      Implies ::= IRIMETA? FORMULA '--:' FORMULA
      FORMULA ::= Implies |
                IRIMETA? CONNECTIVE ('FORMULA') |
                IRIMETA? QUANTIFIER ('FORMULA') |
                IRIMETA? 'Neg' FORMULA |
                IRIMETA? 'Naf' FORMULA |
                IRIMETA? FORMULA '@' MODULEREF |
                FORM
      TERMULA ::= Implies |
                IRIMETA? CONNECTIVE ('TERMULA') |
                IRIMETA? QUANTIFIER ('TERMULA') |
                IRIMETA? 'Neg' TERMULA |
                IRIMETA? 'Naf' TERMULA |
                IRIMETA? TERMULA '@' MODULEREF |
                TERM
      PROFILE ::= ANGLEBRACKIRI

      Note that this is an extension of the syntax for the Baseline schema (FLDBaseline.xsd).
    </xs:documentation>
  </xs:annotation>
  <!--
  The Skyline schema extends, with Implies, the FORMULA and TERMULA groups
  of the Baseline schema from the same directory
  -->
  <xs:redefine schemaLocation="FLDBaseline.xsd"/>
</xs:schema>
<xs:group name="FORMULA">
  <xs:choice>
    <xs:group ref="FORMULA"/>
    <xs:element ref="Implies"/>
  </xs:choice>
</xs:group>

<xs:group name="TERMULA">
  <xs:choice>
    <xs:group ref="TERMULA"/>
    <xs:element ref="Implies"/>
  </xs:choice>
</xs:group>

<xs:element name="Document">
  <!--
  Dialect ::= 'Dialect' '{' Name '}' represented with a dialect attribute.
  Base and Prefix represented directly in XML.
  -->
  <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:attribute name="dialect" type="xs:string"/>
  </xs:complexType>
</xs:element>

<xs:element name="directive">
  <xs:complexType>
    <xs:choice>
      <xs:element ref="DIRECTIVE-IMPORT"/>
      <xs:element ref="DIRECTIVE-MODULE"/>
    </xs:choice>
  </xs:complexType>
</xs:element>

<xs:element name="DIRECTIVE-IMPORT">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="Import"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="DIRECTIVE-MODULE">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="Module"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

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

<xs:element name="Import">
  <!--
  Import ::= IRIMETA? 'Import' '{' LOCATOR PROFILE? '}'
  LOCATOR ::= ANGELBRACKIRI
  PROFILE ::= ANGELBRACKIRI
  -->
<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="Module">
  <!--
  Module ::= IRIMETA? 'Module' '(' (Const | Expr) LOCATOR ')'
  LOCATOR ::= ANGLEBRACKIRI
  -->
  <xs:complexType>
    <xs:sequence>
      <xs:group ref="IRIMETA" minOccurs="0" maxOccurs="1"/>
      <xs:choice>
        <xs:element ref="Const"/>
        <xs:element ref="Expr"/>
      </xs:choice>
      <xs:element ref="location"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

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

<xs:element name="Group">
  <!--
  Group ::= IRIMETA? 'Group' '(' (FORMULA | 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="FORMULA"/>
      <xs:element ref="Group"/>
    </xs:choice>
  </xs:complexType>
</xs:element>

<xs:element name="Implies">
  <!--
  Implies ::= IRIMETA? FORMULA ':-' 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:sequence>
      <xs:group ref="FORMULA"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:schema>
</xs:schema>
9 Appendix: A Subframework for Herbrand Semantic Structures

The semantics of most languages in Logic Programming, including the well-founded semantics [GRS91, Prz94] and the answer set (or stable model) semantics [SL88, GL91, GL92] are defined with respect to Herbrand semantic structures [CL73]. This appendix introduces the concepts of Herbrand Universe, Herbrand Structures, and related notions in the context of RIF-FLD in order to facilitate specializations of the RIF logical framework to logic programming dialects.

A RIF-FLD semantic structure, $I = < TV, DTS, D, I_c, I_h, I_l, I_{lit}, I_{iat}, I_{name}, I_{sub}, I_{obj}, I_{external}, I_{concrete}, f_{truth}>$, is Herbrand if its domain, $D$, is Herbrand and the mappings $I_c, I_h, I_l, I_{lit}, I_{iat}, I_{name}, I_{sub}, I_{obj}, I_{external}, I_{concrete}, f_{truth}$ satisfy certain conditions. The definitions, below, will make this statement precise.

In what follows, we will be calling any variable-free term a ground term.

Definition (Herbrand Universe and Domain). Given a language of RIF-FLD, a Herbrand RIF-FLD universe, $HU$, is a set consisting of all the ground well-formed terms defined by RIF-FLD except the aggregate terms, external terms, and remote term references.

Given a semantic structure $I$, as above, we say that it has a Herbrand RIF-FLD domain if $D$ (its domain) is a factor of $I$. $I$ is Herbrand and the mappings $I_c, I_h, I_l, I_{lit}, I_{iat}, I_{name}, I_{sub}, I_{obj}, I_{external}, I_{concrete}, f_{truth}$ satisfy certain conditions. The definitions, below, will make this statement precise.

We will use the symbol $HD$ to denote Herbrand domains.

Note that the general properties of $TV$ in semantic structures (Definition Truth valuation) also imply that:

- If $s, t \in HU$ are terms with named arguments that differ only in the order of the arguments then $(s, t) \in E$.
- If $s, t \in HU$ are frame terms that differ only in the order of their attribute/value pairs then $(s, t) \in E$.

Definition (Herbrand Semantic Structure). A RIF-FLD semantic structure $I$ of the above form is a Herbrand RIF-FLD semantic structure iff:

- It has a Herbrand RIF-FLD domain $HD$.
- The term-interpreting mapping $I$ is such that it maps every ground term in $HU$ to its equivalence class in $HD$.

This mapping implicitly also defines the subdomains of $HD$, which correspond to the various signatures and are defined earlier -- see the effect of signatures on the semantics. Namely, for any signature $sg$, its subdomain $HD_{sg} \subseteq HD$ is precisely the set $(I(t)| t \in \text{a well-formed term that has sg as one of its signatures})$.

RIF-FLD semantic multi-structures that are built out of Herbrand RIF-FLD semantic structures will be called Herbrand RIF-FLD semantic multi-structures.

Logic programming dialects often use the following notion of minimal Herbrand RIF-FLD models.

Definition (Minimal Model with Respect to the Truth Order). Let $Γ$ be a ground RIF-FLD document. A Herbrand RIF-FLD semantic multi-structure $I$ that is a model of $Γ$ is said to be a minimal model in the truth order $<$ of $Γ$ if there is no other Herbrand RIF-FLD model $I'$ of $Γ$ such that:

- $TV(I(φ)) = t$ implies $TV(I'(φ)) = t$ and
- $TV(I(φ)) = f$ implies $TV(I'(φ)) = f$

for every formula $φ$ of the form $L$ or Neg $L$, where $L$ is a ground atomic formula. Dialects may further specialize this notion by imposing additional restrictions.

Least semantic structures, defined below, are often used in the definitions of various fixpoint operators as starting points of the iteration process that computes the least fixpoint.

Definition (Least Herbrand Structure with Respect to the Truth Order). A Herbrand RIF-FLD semantic structure $I$ is said to be the least in the truth order $<$ if $I_{name}$ maps every element of the Herbrand domain to $f$ except for those elements that correspond to tautological formula terms (for example, And($\{}$)) -- these are mapped to $t$. Dialects might have additional requirements. For example, some elements of the Herbrand domain might be "tautologically undefined," i.e., always mapped to $u$.

The standard definitions of the well-founded semantics typically employ so-called "empty" semantic structures -- structures where everything that can be undefined is undefined. The following definition adapts this concept to RIF-FLD. It applies to dialects that have a special undefinedness truth value $u$ such that $f < u < t$. The usual general definition for $u$ is that it is the smallest element in $TV$ with respect to the knowledge order $<$ and an order on the sets of truth values, which is sometimes used in addition to the truth order [Ft02]. In many cases, however, the mention of $<$ is omitted as, for example, in the case of the well-founded semantics (where it is implicitly assumed that $u < k$ and $u < t$) and in the case of stable models (where $k$ is an empty relation).

Definition (Empty Herbrand Structure). Let $I$ be a Herbrand RIF-FLD semantic structure with the set of truth values $TV$ that has a special undefinedness truth value $u$ such that $f < u < t$. Then $I$ is said to be empty if $I_{name}$ maps everything to $u$ except for the elements of the Herbrand domain that correspond to tautological formula terms (for example, And($\{}$)), which are mapped to $t$, and elements of the domain that correspond to unsatisfiable formulas (e.g., Or($\{}$)), which are mapped to $f$. Dialects may have additional requirements. For example, some elements of the Herbrand domain might always have some other truth values specific to the particular dialects.

The above concepts were defined exclusively of ground RIF-FLD documents, but practically interesting RIF documents are usually non-ground. A typical mechanism by which non-ground documents are reduced to the ground ones is called ground instantiation. It applies only to universal RIF-FLD documents.

Definition (Universal RIF-FLD Document). A RIF-FLD document is universal if it has the form Document(direction1; ...; directionn; Γ) or Document(direction1; ...; directionn). In the former case, when the group formula $Γ$ is present, $Γ$ must be a universal formula.

A non-group, non-document formula is universal if it has the form Forall $∀V_1 \ldots ∀V_n(η)$, where $η$ has no quantifiers and all of its variables are among $∀V_1 \ldots ∀V_n$. A group formula Group($ψ_1 \ldots ψ_n$) is universal if either $n=0$ (i.e., it is an empty group formula) or each $ψ_i$ is universal.
Ground instantiations are now defined as follows.

**Definition (Ground Instantiations).** Let $\Gamma$ be a universal RIF-FLD document. Its **ground instantiation** is a set of RIF-FLD documents obtained from $\Gamma$ by replacing every RIF-FLD non-group formula in $\Gamma$ that is a direct subformula of a group formula with the set of all their ground instances.

A universal formula $\phi$ is said to be a **ground instance** of another formula, $\psi$, if and only if $\phi$ is obtained from $\psi$ by a coherent replacement of variables with ground terms in $\text{HU}$. Coherence here means that, while constructing $\phi$ from $\psi$, the same variables are always substituted with the same terms. (Note: $I(\psi)$ is a ground instance of $\psi$, but there can be many others, since variables can be mapped to arbitrary ground terms in $\text{HU}$.)

Note that according to the definition of the **truth valuation**, $\text{TVal}(\phi) = \neg \text{TVal}(\text{Naf } \phi)$, i.e., when $\phi$ is true then $\text{Naf } \phi$ is false, and vice versa (and, for example, when one is undefined in three-valued semantics then so is the other). However, RIF-FLD imposes no constraints on $\text{Neg } \phi$. Many logic programming theories require consistency of the Herbrand models, and the following definition is provided for the use by the corresponding dialects. However, the definitions in this section do not rely on this consistency assumption.

**Definition (Consistent Semantic Structure).** A Herbrand RIF-FLD semantic structure $I$ is said to be **consistent** if $\text{TVal}(\phi)$ and $\text{TVal}(\text{Neg } \phi)$ are not both $\text{t}$ (which is equivalent to saying that $I(\phi)$ and $I(\text{Neg } \phi)$ are not both $\text{t}$). However, they can both be $\text{f}$ (or, for example, $\text{u}$, in three-valued dialects). A semantic multi-structure is consistent if all its component semantic structures are consistent.

10 Appendix: Change Log (Informative)

This appendix summarizes the main changes to this document.

Changes since the **draft of July 3, 2009**.
- "All RIF dialects are expected to support certain symbols spaces" was added.
- "instance" of an external schema was replaced with "instantiation" of an external schema.
- More examples were added; some examples were better explained.
- IRICONST was replaced with ANYURICONST in FLDSkyline.xsd, v. 1.3.
- The xs:include was dropped and the two xs:redefine’s merged in FLDSkyline.xsd, v. 1.4.
- A number of typos were found and fixed.

Changes since the **Candidate Recommendation of October 1, 2009**.
- Import’s anyURIs were moved directly into location and profile.
- Appendix on Herbrand Semantic Structures was added.
- Several typos fixed and clarifications added.
- Fixed List by permitting IRIMETA and aligning syntax to Expr and Atom.

Changes since the **Recommendation of June 22 2010**.
- Added anti-monotonicity requirement for $\neg$ in **Definition (set of truth values).**
- Changed **Definition (truth valuation of document formulas)** to define truth valuation of not only document formulas but other formulas as well.
- Acknowledgements added.
- The old notion of semantic multi-structures has been changed to fix the problems with the semantics of document formulas found in **Recommendation of June 22 2010**. These problems were pointed out in [DAA]. As a result, sections Interpretation of Documents and Logical Entailment have largely been rewritten.
- Two new rows added in the table of Section Mapping of the Non-annotated RIF-FLD Language for empty args in Atom and Expr; restriction of earlier Atom and Expr rows to non-empty args (replaced $n$ with $m$).
- Other minor corrections (see [here](http://www.w3.org/TR/2012/PER-rif-fld-20121211/) and [here](http://www.w3.org/TR/2012/PER-rif-fld-20121211/)).