W3C Proposed Edited Recommendation

RIF Framework for Logic Dialects

W3C Proposed Edited Recommendation 11 December 2012

This version:
http://www.w3.org/TR/2012/PER-rif-fld-20121211/
Latest version:
http://www.w3.org/TR/rif-fld/
Previous version:
http://www.w3.org/TR/2010/REC-rif-fld-20100622/
Editors:
Harold Boley, National Research Council Canada
Michael Kifer, State University of New York at Stony Brook, USA

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

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

Abstract

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May Be Superseded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.

Patents

This document was produced by a group operating under the 5 February 2004 W3C Patent Policy. W3C maintains a public list of any patent disclosures made in connection with the deliverables of the group; that page also includes instructions for disclosing a patent. An individual who has actual knowledge of a patent which the individual believes contains Essential Claim(s) must disclose the information in accordance with section 6 of the W3C Patent Policy.

Table of Contents

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 Semantic Web services languages such as [SWSL-Rules] and [WSML-Rules].

This document is mostly intended for the *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.* The semantic 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 Relfun [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 logic 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 Section 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`). Those 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 Relfun [RF99].

- *Formula terms*. These terms are the ones 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 *reify* formulas and manipulate them as objects.
- *External*. These terms are used to represent built-ins and external data sources that are treated as "black boxes."
- *Aggregation*. These are the terms that are used to represent aggregation functions over sets.
- *Remote*. 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 (`NEWAGGRFUNC`), 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/rif#iri). 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 [Enderton01]. 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 like `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, Mendelson97]). 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 `g` if for every semantic structure ***I*** in some set **S**, if `G` is true in ***I*** then `g` 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 [Lloyd87] of G and, furthermore, only the minimal models of a special kind. See [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 [RIF-BLD]. A core dialect, which is defined by further specializing RIF-BLD, is specified in [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 `NEWSYMBOL`, `NEWCONNECTIVE`, `NEWQUANTIFIER`, and `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 [KLW95]). This is done by actualizing the extension point `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 symbols 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., [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`.

  Constants are written as `"literal"^^symspace`, where `literal` is a sequence of Unicode characters and `symspace` is an identifier for a symbol space. This syntax is explained in Section Symbol Spaces.

- A countably infinite set of ***variable symbols*** `Var`.

  Variables are written as Unicode strings preceded by the symbol `?` (e.g., `?x`, `?ABC`). This makes the sets `Var` and `Const` disjoint.

- A countably infinite set of ***argument names*** `ArgNames`. The set `ArgNames` is disjoint from both `Const` and `Var`.

  Argument names in `ArgNames` are written as Unicode strings that do not start with a `?` (e.g., `Name`, `age`). They are used in predicates and functions that have named arguments.

- A finite set of ***connective 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 set of 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 $\mathtt{Exists}_{?X_1,...,?X_n}$ and $\mathtt{Forall}_{?X_1,...,?X_n}$, where $?X_1, \ldots, ?X_n$, $n \geq 0$, 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 $\mathtt{Exists}\ ?X_1,...,?X_n$ and $\mathtt{Forall}\ ?X_1,...,?X_n$ instead of $\mathtt{Exists}_{?X_1,...,?X_n}$ and $\mathtt{Forall}_{?X_1,...,?X_n}$.

  Every quantifier symbol has an ***associated list of variables*** that are ***bound*** by that quantifier. For the standard quantifiers $\mathtt{Exists}_{?X_1,...,?X_n}$ and $\mathtt{Forall}_{?X_1,...,?X_n}$, the associated list of variables is $?X_1,...,?X_n$.

- 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 $sym_{?V[?X_1 \ldots ?X_n]}$, where $n \geq 0$, *sym* is a symbol that denotes an aggregate function, and $?V$, $?X_1, \ldots, ?X_n$ are variable symbols. The symbol `?V` is called the ***comprehension variable*** of the aggregate symbol and $?X_1, \ldots, ?X_n$ 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, `NEWAGGRFUNC`, which must be actualized. Dialects can specialize the aforesaid set of aggregate functions by

  - Replacing `NEWAGGRFUNC` 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 [Clark87], 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.
☐

W3C Proposed Edited Recommendation

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/1999/02/22-rdf-syntax-ns#`
- `pred:` stands for `http://www.w3.org/2007/rif-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-DTB].

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 logic language of RIF-FLD.

RIF requires that all dialects include the symbol spaces listed and described in Section Constants and Symbol Spaces of [RIF-DTB] 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 t_1, ..., t_n are terms then $t(t_1 \ ... \ t_n)$ 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 ?W)(?Y ?Z(?V "33"^^xs:integer))` is a positional term (as in HiLog [CKW93]).

3. *Terms with named arguments*. A ***term with named arguments*** is of the form $t(s_1\text{->}v_1 \ ... \ s_n\text{->}v_n)$, where `t`, v_1, ..., v_n are terms, and s_1, ..., s_n are (not necessarily distinct) symbols from the set `ArgNames`.

   The term `t` here represents a predicate or a function; s_1, ..., s_n represent argument names; and v_1, ..., v_n represent argument values. Terms with named arguments are like regular positional terms except that the arguments 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.

   For instance, `"person"^^xs:string("http://example.com/name"^^rif:iri->?Y "http://example.com/address"^^rif:iri->?Z)`, `?X("123"^^xs:integer ?W)(arg->?Y arg2->?Z(?V))`, and `"Closure"^^rif:local("http://example.com/relation"^^rif:iri->"http://example.com/Flight"^^rif:iri)("from"^^rif:local->?X "to"^^rif:local->?Y)` are terms with named arguments. The second of these named-argument terms uses a positional term, `?X("123"^^xs:integer ?W)`, in the role of the function, and the third term's function is itself represented by a named-argument term.

4. *List terms*. There are two kinds of list terms: *open* and *closed*.

- A ***closed list*** has the form List(t_1 ... t_m), where m≥0 and t_1, ..., t_m are terms.
- An ***open list*** (or a list with a tail) has the form OpenList(t_1 ... t_m t), where m>0 and t_1, ..., t_m, t are terms. Open lists are written in the presentation syntax as follows: List(t_1 ... t_m | 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 3 is not). Such general open lists correspond to Lisp's dotted lists [Steele90]. Note that they can be the result of instantiating an open list with a variable in the tail, hence are hard to avoid. For instance, List(1 2 | 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*.
   - t#s is a ***membership term*** if t and s are terms.
   - t##s is a ***subclass term*** if t and s are terms.

   Classification terms are used to describe class hierarchies.

7. *Frame terms*. t[p_1->v_1 ... p_n->v_n] is a ***frame term*** (or simply a ***frame***) if t, p_1, ..., p_n, v_1, ..., v_n, $n \geq 0$, are 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 p_i->v_i 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 used for representing built-in functions and predicates as well as "procedurally attached" terms or predicates, which might exist in various rule-based systems, but are not specified by RIF. 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/acme"^^rif:iri["http://example.com/mycompany/president"^^rif:iri(?Year) -> ?Pres] <http://example.com/acme>) 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/foobar"^^rif:iri then p* is http://example.com/foobar).

9. *Formula term.* If S is a connective or a quantifier symbol and t_1, ..., t_n are terms then S(t_1 ... t_n) 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(t_1 ... t_n), where S is a quantifier, and let ?X_1,...,?X_n 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 τ is a term and ψ a formula term that occurs in τ then all occurrences of the variables that are bound in φ are also said to be bound in τ. 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* $_{?V[?X_1 \ldots ?X_n]}$(τ), where *sym* $_{?V[?X_1 \ldots ?X_n]}$ is an aggregate symbol, n≥0, and τ is a term. For readability, we will usually write aggregate terms as *sym*{?V [?X_1 ... ?X_n] | τ}. 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 ?V, ?X_1, ..., ?X_n have free occurrences in τ, and all occurrences of other variables in τ are bound.

    The comprehension variable ?V and the grouping variables ?X_1, ..., ?X_n of the symbol *sym* $_{?V[?X_1 \ldots ?X_n]}$ are also said to be the comprehension and grouping variables of the above aggregate term. The *comprehension variable* ?V *is considered bound* by the aggregation term, but the *grouping variables* ?X_1, ..., ?X_n *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 [?X_1 ... ?X_n] | τ} is treated as a shorthand for *sym*{?V [?X_1 ... ?X_n] | And(τ ?V=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 extensible 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), where t could be a positional term and φ a formula term. (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->e f->g)). Some language environments, like FLORA-2 [FL2], OO jDREW [OOjD], NxBRE [NxBRE], 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, p_1, ..., p_n, 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 symbol (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(?X 5) "abc")
- Term with named arguments: "http://example.com/Person"^^rif:iri(id->"http://example.com/John"^^rif:iri "http://example.com/age"^^rif:iri->?X "http://example.com/spouse"^^rif:iri->?Y)
- Frame term: "http://example.com/John"^^rif:iri[age->?X spouse->?Y]
- Lists
  - Empty list: List()
  - Closed list with variable inside: List("a"^^rif:local ?Y "c"^^rif:local)
  - Open list with variables: List("a"^^rif:local ?Y "c"^^xs:string | ?Z)
  - Equality term with lists inside: List(?Head | ?Tail) = List("a"^^rif:local ?Y "c"^^xs:string)
  - Nested list: List("a"^^rif:local List(?X "b"^^rif:local) "c"^^xs:string)
- Classification terms
  - Membership: ?X # ?Y
  - Subclass: ?X ## "http://example.com/ex1"^^rif:iri(?Y)
  - Membership: "http://example.com/John"^^rif:iri # "http://example.com/Person"^^rif:iri
  - Subclass: "http://example.com/Student"^^rif:iri ## "http://example.com/Person"^^rif:iri
- External term: External(pred:numeric-greater-than(?diffdays 10)))
- Formula terms
  - :-("p"^^rif:local(?X) ?X("q"^^xs:string)) (usually written as "p"^^rif:local(?X) :- ?X("q"^^xs:string))
  - Forall$_{?X,?Y}$(Exists$_{?Z}$("p"^^rif:local(?X ?Y ?Z))) (usually written as Forall ?X ?Y (Exists ?Z ("p"^^rif:local(?X ?Y ?Z)))
  - Or("http://example.com/to-be"^^rif:iri(?X) Neg("http://example.com/to-be"^^rif:iri(?X)))
- Aggregate term: avg{?Sal [?Dept]|Exists ?Empl "http://example.com/salary"^^rif:local(?Empl ?Dept ?Sal)}
- Remote term: ?O[?N -> "John"^^rif:string "http://example.com/salary"^^rif:iri -> ?S]@"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 which externally defined terms are acceptable in a RIF dialect. Externally defined terms include RIF built-ins, which are specified in [RIF-DTB], 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 (?X_1 ... ?X_n; 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.
- ?X_1 ... ?X_n 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, (?X ?Y; ?X["foo"^^xs:string->?Y]; loc) and (?V ?W; ?V["foo"^^xs:string->?W]; loc) are considered to be indistinguishable, but (?X ?Y; ?X["foo"^^xs:string->?Y]; loc) and (?Y ?X; ?X["foo"^^xs:string->?Y]; loc) are viewed as different schemas.

An external term External(t loc1) is an ***instantiation*** of an external schema (?X_1 ... ?X_n; τ; loc) iff loc1=loc and t can be obtained from τ by a simultaneous substitution ?X_1/s_1 ... ?X_n/s_n of the variables ?X_1 ... ?X_n with terms s_1 ... s_n, respectively. Some of the terms s_i can be variables themselves. For example, External(?Z["foo"^^xs:string->f("a"^^rif:local ?P)] loc) is an instantiation of (?X ?Y; ?X["foo"^^xs:string->?Y]; loc) by the substitution ?X/?Z ?Y/f("a"^^rif:local ?P). ☐

Observe that a variable cannot be an instantiation of an external schema, 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(?X loc) is not well-formed in RIF.

The intuition behind the notion of an external schema, such as (?X ?Y; ?X["foo"^^xs:string->?Y] <http://example.com/acme>) and (?V; pred:isTime(?V) <pred:isTime>), is that ?X["foo"^^xs:string->?Y] or pred:isTime(?V) 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(t) 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, t, that is an instantiation of two distinct schemas in E. ☐

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"^^xs:string->?Y]; loc)` and `(?Y ?X;  ?X["foo"^^xs: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 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 `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) => 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 `SigNames` 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 `Const`, `Var`, and `ArgNames`. We require that this set includes at least the following *reserved* signature names:

- `atomic` -- used to represent the syntactic context where atomic formulas are allowed to appear.
- `formula` -- represents the context where formulas (atomic or composite) may appear.
- `∞-connective`-- the signature for the connectives, such as `And` and `Or`, that can take any number of arguments.
- `2-connective` -- the signature for the connectives, such as the rule implication connective `:-`, that take exactly two arguments.
- `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 `Forall` and `Exists`.
- `=` -- used for representing contexts where equality terms can appear.
- `#` -- a signature name reserved for membership terms.
- `##` -- a signature reserved for subclass terms.
- `->` -- a signature reserved for frame terms.
- `aggregate` -- a signature reserved for aggregate functions.
- `remote` -- a signature reserved for the symbol `@` that is used to build remote terms.
- `list` -- a signature reserved for the symbol `List` that is used to represent closed lists.
- `openlist` -- a signature reserved for the symbol `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 `SigNames` is dialect-specific; it is used in the definition of well-formed terms below.

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

Definition (Signature). A ***signature*** has the form $\eta\{e_1, \ ..., \ e_n, \ ...\}$ where $\eta \in$ `SigNames` is the name of the signature and $\{e_1, \ ..., \ e_n, \ ...\}$ 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 Relfun [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 $\kappa, \kappa_1, ..., \kappa_n \in$ `SigNames`, $n \ge 0$, are signature names then $(\kappa_1 \ ... \ \kappa_n) \Rightarrow \kappa$ is a ***positional arrow expression***.

  For instance, `() ⇒ individual` and `(individual) ⇒ individual` are positional arrow expressions, if `individual` is a signature name.

- If $\kappa, \kappa_1, ..., \kappa_n \in$ `SigNames`, $n \ge 0$, are signature names and $p_1, ..., p_n \in$ `ArgNames` are argument names then $(p_1\text{->}\kappa_1 \ ... \ p_n\text{->}\kappa_n) \Rightarrow \kappa$ is an ***arrow expression with named arguments***.

  For instance, `(arg1->individual arg2->individual) => individual` is an arrow signature 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 **Σ** of signatures is ***coherent*** iff

1. **Σ** contains the special signatures `atomic{ }` and `formula{ }`, which represent the context of atomic formulas and more generally, composite formulas, respectively. Furthermore, it is required that `atomic < formula`.
2. **Σ** contains the special signature $\infty\text{-connective}\{e_1, \ ..., \ e_n, \ ...\}$, where each e_n has the form `(formula ... formula) ⇒ formula` (the left-hand side of this signature is a sequence of n symbols `formula`). This signature is assigned to the connectives `And` and `Or`.
3. **Σ** contains the special signature `2-connective{(formula formula) ⇒ formula}`. This signature is assigned to the rule implication connective.
4. **Σ** contains the signature `1-connective{(formula) ⇒ formula}`. This signature is assigned to the negation connectives `Naf` and `Neg`, and to the reserved quantifiers of RIF-FLD, $\text{Exists}_{?X_1,...,?X_n}$ and $\text{Forall}_{?X_1,...,?X_n}$, for all variable sequences $?X_1,...,?X_n$ and $n \ge 0$.
5. **Σ** contains the signature $=\{e_1, \ ..., \ e_n, \ ...\}$ for the equality symbol.

All arrow expressions e_i here have the form `(κ κ) ⇒ γ` (the arguments in an equation must be compatible) and at least one of these expressions must have the form `(κ κ) ⇒ atomic` (i.e., equation terms are also atomic formulas). Dialects may further specialize this signature.

6. **Σ** contains the signature `#{`e_1`, ..., `e_n`...}` for membership terms.

   Here all arrow expressions e_i are binary (have two arguments) and at least one has the form `(κ γ) ⇒ atomic`. Dialects may further specialize this signature.

7. **Σ** contains the signature `##{`e_1`, ..., `e_n`...}` for subclass terms.

   Here all arrow expressions e_i have the form `(κ κ) ⇒ γ` (the arguments must be compatible) and at least one of these arrow expressions has the form `(κ κ) ⇒ atomic`. Dialects may further specialize this signature.

8. **Σ** contains the signature `->{`e_1`, ..., `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 `(`$κ_1$ $κ_2$ $κ_3$`) ⇒ atomic`. Dialects may further specialize this signature.
9. **Σ** contains the signatures `list` and `openlist` for representing list terms.
   - The signature `list`, for closed lists, has arrow expressions of the form `() ⇒ κ`, `(κ) ⇒ κ`, `(κ κ) ⇒ κ`, and so on, where κ is a signature.
   - The signature `openlist`, for open lists, has arrow expressions of the form `(κ κ) ⇒ κ`, `(κ κ κ) ⇒ κ`, and so on, where κ is a signature.
10. **Σ** contains the signature `aggregate{`e_1`, `e_2`, ...}` for aggregate terms.

    Here each arrow expression e_i has the form `(formula) ⇒ `$κ_i$, for some signatures $κ_1$, $κ_2$, ...

11. **Σ** contains the signature `remote{`e_1`, `e_2`, ...}`, where at least one of the e_i is an arrow expression of the form `(formula κ) ⇒ formula` for some signature κ. This signature is assigned to the remote term symbol `@`.
12. **Σ** has at most one signature for any given signature name.
13. Whenever **Σ** contains a pair of signatures, ηA and κB, such that η<κ then B⊆A.

    Here ηA denotes a signature with the name η and the associated set of arrow expressions A; similarly κB is a signature named κ with the set of expressions B. The requirement that B⊆A ensures that symbols that have signature η can be used wherever the symbols with signature κ are allowed. ☐

The requirement that coherent sets of signatures must include the signatures for =, #, ->, 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 `mysig{() ⇒ atomic}` and `mysig{(atomic) ⇒ atomic}` because it has two different signatures with the same name. Likewise, if a set contains $mysig_1$`{() ⇒ atomic}` and $mysig_2$`{(atomic) ⇒ atomic}` and $mysig_1$ < $mysig_1$ then it is incoherent because the set of arrow expressions of $mysig_1$ does not contain the set of arrow expressions of $mysig_2$.

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-connective`, `2-connective`, 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 `:-` with the arrow expression `(formula formula) ⇒ 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 `(κ κ) ⇒ atomic` for the equality symbol, one has to tell what κ 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 `={(individual individual) ⇒ atomic, (individual individual) ⇒ 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 `atomic{ }`, we may simply say that symbol `f` *has* signature `atomic`.

Definition (Well-formed term).

1. A *constant* or *variable* symbol with signature η is a well-formed term with signature η.
2. A *positional term* $t(t_1 \ldots t_n)$, $0 \le n$, is well-formed and has a signature σ iff
   - `t` is a well-formed term that has a signature that contains an arrow expression of the form $(\sigma_1 \ldots \sigma_n) \Rightarrow \sigma$; and
   - Each t_i is a well-formed term whose signature is γ_i such that $\gamma_i, \le \sigma_i$.

   As a special case, when `n=0` we obtain that `t()` is a well-formed term with signature σ, if t's signature contains the arrow expression $() \Rightarrow \sigma$.

3. A *term with named arguments* $t(p_1\text{->}t_1 \ldots p_n\text{->}t_n)$, $0 \le n$, is well-formed and has a signature σ iff
   - `t` is a well-formed term that has a signature that contains an arrow expression with named arguments of the form $(p_1\text{->}\sigma_1 \ldots p_n\text{->}\sigma_n) \Rightarrow \sigma$; and
   - Each t_i is a well-formed term whose signature is γ_i, such that $\gamma_i \le \sigma_i$.

   As a special case, when `n=0` we obtain that `t()` is a well-formed term with signature σ, if t's signature contains the arrow expression $() \Rightarrow \sigma$.

4. An *equality term* of the form $t_1=t_2$ is well-formed and has a signature κ iff
   - The signature = has an arrow expression $(\sigma\ \sigma) \Rightarrow \kappa$
   - t_i and t_2 are well-formed terms with signatures γ_1 and γ_2, respectively, such that $\gamma_i \le \sigma$, `i=1,2`.
5. A *membership term* of the form $t_1\#t_2$ is well-formed and has a signature κ iff
   - The signature # has an arrow expression $(\sigma_1\ \sigma_2) \Rightarrow \kappa$
   - t_1 and t_2 are well-formed terms with signatures γ_1 and γ_2, respectively, such that $\gamma_i \le \sigma_i$, `i=1,2`.
6. A *subclass term* of the form $t_1\#\#t_2$ is well-formed and has a signature κ iff
   - The signature ## has an arrow expression $(\sigma\ \sigma) \Rightarrow \kappa$
   - t_1 and t_2 are well-formed terms with signatures γ_1 and γ_2, respectively, such that $\gamma_i \le \sigma$, `i=1,2`.
7. A *frame term* of the form $t[s_1\text{->}v_1 \ldots s_n\text{->}v_n]$ is well-formed and has a signature κ iff
   - The signature -> has arrow expressions $(\sigma\ \sigma_{11}\ \sigma_{12}) \Rightarrow \kappa, \ldots, (\sigma\ \sigma_{n1}\ \sigma_{n2}) \Rightarrow \kappa$ (these `n` expressions need not be distinct).
   - `t`, s_j, and v_j are well-formed terms with signatures γ, γ_{j1}, and γ_{j2}, respectively, such that $\gamma \le \sigma$ and $\gamma_{ji} \le \sigma_{ji}$, where `j=1,...,n` and `i=1,2`.
8. An *externally defined term*, `External(t loc)`, is well-formed and has signature κ iff
   - `t` is well-formed and has signature κ.
   - `External(t 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 $S(t_1 \ldots t_n)$, $0 \le n$ is well-formed if S is a connective or a quantifier whose signature has an arrow expression $(\sigma_1 \ldots \sigma_n) \Rightarrow$ `formula` and each t_i is a well-formed term whose signature is $\le \sigma_i$.

   In the special case of our reserved connectives and quantifiers, $t_1, \ldots, t_n$ must have signatures that are below `formula` (i.e., ≤ `formula`). Also, if S is `:-` then n must be equal 2 and if S is `Neg`, `Naf`, `Forall`, or `Exists` then n=1.

10. An *aggregate term* of the form $sym\{?V\ [?X_1 \ldots ?X_n]\ |\ \tau\}$ is well formed if the aggregate symbol $sym_{\ ?V[?X_1 \ldots ?X_n]}$ is assigned signature `aggregate` and the term $sym_{\ ?V[?X_1 \ldots ?X_n]}(\tau)$ is well-formed (as a positional term).

    This implies that τ must have the signature `formula` or < `formula`. Unless a dialect introduces additional signatures, this also means that τ must be a formula term (i.e., a compound formula) or an atomic formula (see below).

11. A *remote term* of the form φ@r is well-formed if the positional term `@(φ r)` is well-formed. This implies that φ must be well-formed and have the signature `formula`, that `r` must a well-formed term, and that the term φ@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 < `atomic` (less than `atomic` in the order <). 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 < `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, `a :- b`, has the canonical form `:-(a b)` and the canonical form for negation, `Naf p` and `Neg p`, is `Naf(p)` and `Neg(p)`.

1. *Atomic*: If φ is a well-formed atomic formula then it is also a well-formed formula.
2. *Remote*: A well-formed remote term φ@r is also a well-formed formula.
3. *Conjunction*: If $\varphi_1, \ldots, \varphi_n$, $n \ge 0$, are well-formed formula terms then so is $And(\varphi_1 \ldots \varphi_n)$.

   As a special case, `And()` is allowed and is treated as a tautology, i.e., a formula that is always true.

4. *Disjunction*: If $\varphi_1, \ldots, \varphi_n$, $n \ge 0$, are well-formed formula terms then so is $Or(\varphi_1 \ldots \varphi_n)$.

   As a special case, `Or()` is treated as a contradiction, i.e., a formula that is always false.

5. *Symmetric negation*: If φ is a well-formed formula term then so is `Neg φ`.

6. *Default negation*: If φ is a well-formed formula term then so is `Naf φ`.
7. *Rule implication*: If φ and ψ are well-formed formula terms then so is `φ :- ψ`.
8. *Constraint*: If ψ is a well-formed formula term then so is `:- ψ`.

   This type of formulas is also known as an *error-producing constraint*. The intent is that the constraint formula is satisfied if the premise ψ 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 $?V_1$ ... $?V_n$(φ)
   - Exists $?V_1$ ... $?V_n$(φ)

   are well-formed formula terms. Recall that $\mathtt{Forall}_{?V_1,...,?V_n}$ and $\mathtt{Exists}_{?V_1,...,?V_n}$ are the reserved universal and existential quantifiers, respectively. The notation Forall $?V_1$ ... $?V_n$(φ) is an alternative for $\mathtt{Forall}_{?V_1,...,?V_n}$(φ), and similarly for `Exists`.

10. *Group*: If $φ_1$, ..., $φ_n$ are well-formed formula terms or `Group`-formulas then Group($φ_1$ ... $φ_n$) 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(*directive₁* ... *directiveₙ* 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.
    - *directive*$_1$, ..., *directive*$_n$ 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 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 [GRS91]. These two dialects are examples where the `Dialect` directive might be necessary.)
      - A ***base directive*** has the form `Base(<iri>)`, where `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 `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: `Import(loc)` or `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 <IRI>, where `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 `Import(loc)`. The two-argument directive `Import(loc p)` is reserved for RIF dialects, which can use it to import non-RIF logical entities, such as RDF 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 `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 `<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, $φ_i$, $ψ_i$, 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 `mysig{(individual)⇒individual, (individual individual)⇒individual, (individual individual individual)⇒individual}` and `a`, `b`, `c` each has the signature `individual{ }` then `p(p(a) p(a b c))` is a well-formed term with signature `individual{ }`. If instead `p` had the signature `mysig2{(individual`

`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 h_2`{(individual individual)⇒atomic}`; and `closure` has signature hh_1`{(`h_2`)⇒`p_2`}`, where p_2 is the name of the signature p_2`{(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 *one* ***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(_foo[meta_for_frame->"this is an annotation for the entire frame"] _bar[meta_for_object->"this is an annotation for t" meta_for_property->"this is an annotation for w"] *) t[w -> v]`. Generally, the convention associates each annotation to the largest term or formula it precedes.

We suggest to use Dublin Core, RDFS, and OWL properties for metadata, along the lines of Section 7.1 of [OWL-Reference]-- specifically `owl:versionInfo`, `rdfs:label`, `rdfs:comment`, `rdfs:seeAlso`, `rdfs:isDefinedBy`, `dc:creator`, `dc:description`, `dc:date`, and `foaf:maker`.

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-DTB], 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"^^rif:iri`, if `ex` is defined as in the `Prefix(ex ...)` directive below. The second shortcut notation uses angle brackets as a way to shorten the `"..."^^rif:iri` idiom. For instance, the prevous `rif:iri` constant can be alternatively represented as `<http://example.org/ontology#man>`. The last shortcut notation lets one write `rif:iri` constants using IRIs relative to a base, where the base IRI is specified in a `Base` directive. For instance, with the Base directive, below, both <Yorick> and "Yorick"^^`rif:iri` expand into the `rif:iri` constant `"http://www.shakespeare-literature.com/Hamlet/Yorick"^^rif:iri`. The example also illustrates attachment of annotations.

```
Document(
  Base(<http://www.shakespeare-literature.com/Hamlet/>)
  Prefix(dc     <http://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
     )
  )
)
```

The above RIF formulas are (admittedly awkward) logical renderings of the following statements from Shakespeare's Hamlet: "Something is rotten in the state of Denmark," "To be, or not to be," and "Every man has business and desire."

Observe that the above set of formulas has a nested subset with its own annotation, `<facts>`, which contains only a global IRI. ☐

The following example illustrates the use of imported RIF documents and of remote terms.

Example 4 (A RIF-FLD document with imports, remote module references, and aggregation).

The first document, below, imports the second document, which is assumed to be located at the IRI `http://example.org/universityontology`. In addition, the first document has references to two remote modules, which are located at `http://example.org/university#1` and `http://example.org/university#2`, respectively. These modules are assumed to be knowledge bases that provide the usual information about university enrollment, courses offered in different semesters, and so on. The rules corresponding to the remote modules are not shown, as they do not illustrate new features. In the simplest case, these knowledge bases can simply be sets of facts for the predicates/frames that supply the requisite information.

```
Document(
  Prefix(u    <http://example.org/universityontology#>)
  Prefix(pred <http://www.w3.org/2007/rif-builtin-predicate#>)
  Import(<http://example.org/universityontology>)
  Module(_univ(1) <http://example.org/university#1>)
  Module(_univ(2) <http://example.org/university#2>)

  Group(
    Forall ?Stud ?Crs ?Semester ?U (u:takes(?Stud ?Crs ?Semester) :-
                                  ?Stud[u:takes(?Semester)->?Crs]@ _univ(?U))
    Forall ?Prof ?Crs ?Semester ?U (u:teaches(?Prof ?Crs ?Semester) :-
                                  u:teaches(?Prof ?Crs ?Semester)@ _univ(?U))
    Forall ?Crs (u:popular_course(?Crs) :-
                     And(?Crs#u:Course
                         pred:numeric-less-than(500
                                           count{?Stud[?Crs]|Exists ?Semester (u:takes(?Stud ?Crs ?Semester))})))
  )
)
```

The imported document, located at `http://example.org/universityontology`, has the following form:

```
Document(
  Group(
    Forall ?Stud ?Prof ?Sem
              (u:studentOf(?Stud ?Prof) :-
                      And(u:takes(?Stud ?Crs ?Sem) u:teaches(?Prof ?Crs ?Sem)))
  )
)
```

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 `s` takes a course `c` in a certain semester `s` 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:studentOf` (a student is a `studentOf` 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:studentOf` 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.

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.

```
Document      ::= IRIMETA? 'Document' '(' Dialect? Base? Prefix* Import* Module* Group? ')'
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
PROFILE        ::= ANGLEBRACKIRI
FORM           ::= IRIMETA? (Var | ATOMIC |
                              'External' '(' ATOMIC LOCATOR? ')')
ATOMIC         ::= Const | Atom | Equal | Member | Subclass | Frame
Atom           ::= UNITERM
UNITERM        ::= TERMULA '(' (TERMULA* | (Name '->' TERMULA)*) ')'
Equal          ::= TERMULA '=' TERMULA
Member         ::= TERMULA '#' TERMULA
Subclass       ::= TERMULA '##' TERMULA
Frame          ::= TERMULA '[' (TERMULA '->' TERMULA)* ']'
TERMULA        ::= Implies |
                   IRIMETA? CONNECTIVE '(' TERMULA* ')' |
                   IRIMETA? QUANTIFIER '(' TERMULA ')' |
                   IRIMETA? 'Neg' TERMULA |
                   IRIMETA? 'Naf' TERMULA |
                   IRIMETA? TERMULA '@' MODULEREF |
                   TERM
TERM           ::= IRIMETA? (Var | EXPRIC | List |
                              'External' '(' EXPRIC LOCATOR? ')' |
                              AGGREGATE | NEWTERM)
EXPRIC         ::= Const | Expr | Equal | Member | Subclass | Frame
Expr           ::= UNITERM
List           ::= 'List' '(' TERM* ')' | 'List' '(' TERM+ '|' TERM ')'
AGGREGATE      ::= AGGRFUNC '{' Var ('[' Var+ ']')? '|' FORMULA '}'
Const          ::= '"' UNICODESTRING '"^^' SYMSPACE | CONSTSHORT
MODULEREF      ::= Var | Const | Expr
CONNECTIVE     ::= 'And' | 'Or' | NEWCONNECTIVE
QUANTIFIER     ::= ('Exists' | 'Forall' | NEWQUANTIFIER) Var*
AGGRFUNC       ::= 'Min' | 'Max' | 'Sum' | 'Prod' | 'Avg' | 'Count' |
                   'Set' | 'Bag' | NEWAGGRFUNC
Var            ::= '?' Name
Name           ::= NCName | '"' UNICODESTRING '"'
SYMSPACE       ::= ANGLEBRACKIRI | CURIE

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

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 CONSTSHORT. 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 (*...*)-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 $\mathrm{Exists}_{?X_1,...,?X_n}$ and $\mathrm{Forall}_{?X_1,...,?X_n}$ are linearized as Exists $?X_1,...,?X_n$ and Forall $?X_1,...,?X_n$. 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 integers, strings, and rif:local symbols. The semantics, below, is described using the full syntax, i.e., we assume that all shortcuts have already been expanded, as defined in [RIF-DTB], Section Constants and Symbol Spaces.

W3C Proposed Edited Recommendation

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 ($\boldsymbol{I}_{\mathtt{frame}}$ and $\boldsymbol{I}_{\mathrm{NF}}$ in this case) 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 as part of its 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 [GRS91, GL88].

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 $<_t$. In some dialects, $<_t$ can be a total order. We write $a \le_t b$ if either $a <_t b$ or a and b are the same element of ***TV***. In addition,

- ***TV*** must be a complete lattice with respect to $<_t$, i.e., the least upper bound (lub_t) and the greatest lower bound (glb_t) must exist for any subset of ***TV***.
- ***TV*** is required to have two distinguished elements, **f** and **t**, such that $\mathbf{f} \le_t elt$ and $elt \le_t \mathbf{t}$ for every $elt \in \boldsymbol{TV}$.
- ***TV*** has an ***operator of negation***, $\sim: \boldsymbol{TV} \to \boldsymbol{TV}$, such that
    - ~ is a self-inverse function: applying ~ twice gives the identity mapping.
    - ~ is anti-monotonic: $elt_1 \le_t elt_2$ implies $\sim elt_2 \le_t \sim elt_1$.
    - $\sim\mathbf{t} = \mathbf{f}$ (and thus $\sim\mathbf{f} = \mathbf{t}$).

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 $\mathbf{f} <_t \mathbf{u} <_t \mathbf{t}$. Handling of contradictions and uncertainty usually requires at least four truth values: **t**, **u**, **f**, and **i** (inconsistent). In this case, the truth order is partial: $\mathbf{f} <_t \mathbf{u} <_t \mathbf{t}$ and $\mathbf{f} <_t \mathbf{i} <_t \mathbf{t}$. The negation operator ~ is then defined to be the identity on the new truth values **u** and **i**.

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***, $\boldsymbol{I}$, is a tuple of the form <$\boldsymbol{TV}$, $\boldsymbol{DTS}$, $\boldsymbol{D}$, $\boldsymbol{I}_C$, $\boldsymbol{I}_V$, $\boldsymbol{I}_F$, $\boldsymbol{I}_{NF}$, $\boldsymbol{I}_{list}$, $\boldsymbol{I}_{tail}$, $\boldsymbol{I}_{frame}$, $\boldsymbol{I}_{sub}$, $\boldsymbol{I}_{isa}$, $\boldsymbol{I}_=$, $\boldsymbol{I}_{external}$, $\boldsymbol{I}_{connective}$, $\boldsymbol{I}_{truth}$>. Here $\boldsymbol{D}$ is a non-empty set of elements called the ***domain*** of $\boldsymbol{I}$. We will continue to use `Const` to refer to the set of all constant symbols and `Var` to refer to the set of all variable symbols. $\boldsymbol{TV}$ denotes the set of truth values that the semantic structure uses and $\boldsymbol{DTS}$ is a set of identifiers for datatypes.

The other components of $\boldsymbol{I}$ are *total* mappings defined as follows:

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

   This mapping interprets constant symbols.

2. $\boldsymbol{I}_V$ maps `Var` to elements of $\boldsymbol{D}$.

   This mapping interprets variable symbols.

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

   This mapping interprets positional terms.

4. $\boldsymbol{I}_{NF}$ interprets terms with named arguments. It is a total mapping from $\boldsymbol{D}$ to the set of total functions of the form `SetOfFiniteBags(ArgNames` × $\boldsymbol{D}$) → $\boldsymbol{D}$.

   This is analogous to the interpretation of positional terms with two differences:

   - Each pair `<s,v>` ∈ `ArgNames` × $\boldsymbol{D}$ 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->c)`. 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->b a->b)` if the variables `?A` and `?B` are both instantiated with the symbol `b`.

5. $\boldsymbol{I}_{list}$ and $\boldsymbol{I}_{tail}$ are used to interpret lists. They are mappings of the following form:
   - $\boldsymbol{I}_{list} : \boldsymbol{D}^* \rightarrow \boldsymbol{D}$
   - $\boldsymbol{I}_{tail} : \boldsymbol{D}^+ \times \boldsymbol{D} \rightarrow \boldsymbol{D}$

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

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

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

6. $\boldsymbol{I}_{frame}$ is a total mapping from $\boldsymbol{D}$ to total functions of the form `SetOfFiniteBags(`$\boldsymbol{D} \times \boldsymbol{D}$`)` → $\boldsymbol{D}$.

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

7. $\boldsymbol{I}_{sub}$ gives meaning to the subclass relationship. It is a total function $\boldsymbol{D} \times \boldsymbol{D} \rightarrow \boldsymbol{D}$.

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

8. $\boldsymbol{I}_{isa}$ gives meaning to class membership. It is a total function $\boldsymbol{D} \times \boldsymbol{D} \rightarrow \boldsymbol{D}$.

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

9. $\boldsymbol{I}_=$ is a total function $\boldsymbol{D} \times \boldsymbol{D} \rightarrow \boldsymbol{D}$.

   It gives meaning to the equality operator.

10. $\boldsymbol{I}_{\text{truth}}$ is a total mapping $\boldsymbol{D} \rightarrow \boldsymbol{TV}$.

    It is used to define truth valuation for formulas.

11. $\boldsymbol{I}_{\text{external}}$ is a mapping from the coherent set of schemas for externally defined terms to total functions $\boldsymbol{D}^* \rightarrow \boldsymbol{D}$. For each external schema σ = `(?X1 ... ?Xn; τ; loc)` in the coherent set of such schemas associated with the language, $\boldsymbol{I}_{\text{external}}(\sigma)$ is a function of the form $\boldsymbol{D}^n \rightarrow \boldsymbol{D}$.

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

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

12. $\boldsymbol{I}_{\text{connective}}$ is a mapping that assigns every connective, quantifier, or aggregate symbol a function $\boldsymbol{D}^* \rightarrow \boldsymbol{D}$.

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

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

1. $\boldsymbol{I}$(k) = $\boldsymbol{I}_C$(k), if k is a symbol in `Const`
2. $\boldsymbol{I}$(?v) = $\boldsymbol{I}_V$(?v), if ?v is a variable in `Var`
3. $\boldsymbol{I}$(`f(t1 ... tn)`) = $\boldsymbol{I}_F(\boldsymbol{I}(f))(\boldsymbol{I}(t_1),...,\boldsymbol{I}(t_n))$
4. $\boldsymbol{I}$(`f(s1->v1 ... sn->vn)`) = $\boldsymbol{I}_{NF}(\boldsymbol{I}(f))(\{<s_1,\boldsymbol{I}(v_1)>,...,<s_n,\boldsymbol{I}(v_n)>\})$

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

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

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

    - $\boldsymbol{I}$(`List(t1 ... tn)`) = $\boldsymbol{I}_{\text{list}}(\boldsymbol{I}(t_1), ..., \boldsymbol{I}(t_n))$, if n>0.
    - $\boldsymbol{I}$(`List(t1 ... tn | t)`) = $\boldsymbol{I}_{\text{tail}}(\boldsymbol{I}(t_1), ..., \boldsymbol{I}(t_n), \boldsymbol{I}(t))$, if n>0.

6. $\boldsymbol{I}$(`o[a1->v1 ... an->vn]`) = $\boldsymbol{I}_{\text{frame}}(\boldsymbol{I}(o))(\{<\boldsymbol{I}(a_1),\boldsymbol{I}(v_1)>, ..., <\boldsymbol{I}(a_n),\boldsymbol{I}(v_n)>\})$

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

7. $\boldsymbol{I}$(c1##c2) = $\boldsymbol{I}_{\text{sub}}(\boldsymbol{I}(c1), \boldsymbol{I}(c2))$
8. $\boldsymbol{I}$(o#c) = $\boldsymbol{I}_{\text{isa}}(\boldsymbol{I}(o), \boldsymbol{I}(c))$
9. $\boldsymbol{I}$(x=y) = $\boldsymbol{I}_{=}(\boldsymbol{I}(x), \boldsymbol{I}(y))$
10. $\boldsymbol{I}$(`External(t loc)`) = $\boldsymbol{I}_{\text{external}}(\sigma)(\boldsymbol{I}(s_1), ..., \boldsymbol{I}(s_n))$, if `External(t loc)` is an instantiation of the external schema σ = `(?X1 ... ?Xn; τ; loc)` by substitution ?X1/s1 ... ?Xn/sn.

    Note that, by definition, `External(t 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 $\boldsymbol{I}$(`External(t loc)`) is well-defined.

11. If S is a connective, a quantifier, or an aggregate and `S(t1 ... tn)` is a well-formed formula term (for an aggregate, n=1) then

    $\boldsymbol{I}$(`S(t1 ... tn)`) = $\boldsymbol{I}_{\text{connective}}(S)(\boldsymbol{I}(t_1) ... \boldsymbol{I}(t_n))$

12. For standard aggregates, the mapping $\boldsymbol{I}$ is defined as follows.

    Let *aggr*{`?X [?X1 ... ?Xn] | τ`} be an aggregate and let *S* be the following set:

    $S = \{(\boldsymbol{I}_V^*(?X),\boldsymbol{I}_V^*(?X_1), ..., \boldsymbol{I}_V^*(?X_n))$ | for all semantic structures $\boldsymbol{I}^*$ such that $\boldsymbol{I}^*(\tau)$ = **t** and $\boldsymbol{I}^*$ is exactly like $\boldsymbol{I}$ except that $\boldsymbol{I}_V^*(?X)$ can be different from $\boldsymbol{I}_V(?X)\}$.

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

    a. *Set aggregate*:
        - $\boldsymbol{I}$(`setof{?X [?X1 ... ?Xn] | τ}`) = $\boldsymbol{I}_{\text{list}}(L)$

          where L is a sorted list of the elements in $\overline{S}_{\text{set}}$. 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 $\boldsymbol{I}$ is indeterminate (i.e., it can be any element of the domain $\boldsymbol{D}$).

          The requirement that the list L must be sorted comes from the fact that there can be many ways to represent $\overline{S}_{\text{set}}$ as a list, while $\boldsymbol{I}$(`setof{?X [?X1 ... ?Xn] | τ}`) must be defined as one concrete element of the domain $\boldsymbol{D}$. Sorting a set is a standard way of providing the requisite unique representation.

    b. *Bag aggregate*:
        - $\boldsymbol{I}$(`bagof{?X [?X1 ... ?Xn] | τ}`) = $\boldsymbol{I}_{\text{list}}(L)$

          where L is a sorted list of the elements in $\overline{S}_{\text{bag}}$. This is well-defined only for semantic structures with totally ordered domains. If L is infinite then the value of the aggregate in $\boldsymbol{I}$ is indeterminate (i.e., it can be any element of the domain $\boldsymbol{D}$).

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

    c. *Min aggregate*:

- $\boldsymbol{I}$(`min{?X [?X1 ... ?Xn] | τ}`) = `min`($\bar{S}_{bag}$), if the function `min` is defined for $\bar{S}_{bag}$ in the dialect. If not, the value of the aggregate in $\boldsymbol{I}$ is indeterminate. The bag $\bar{S}_{bag}$ must have a well-defined total order and `min` must compute the minimum elements of finite totally ordered bags.

d. *Max aggregate*:
   - $\boldsymbol{I}$(`max{?X [?X1 ... ?Xn] | τ}`) = `max`($\bar{S}_{bag}$), if the function `max` is defined for $\bar{S}_{bag}$ in the dialect. If not, the value of the aggregate in $\boldsymbol{I}$ is indeterminate. The bag $\bar{S}_{bag}$ must have a well-defined total order and `max` must compute the maximum elements of finite totally ordered bags.

e. *Count aggregate*:
   - $\boldsymbol{I}$(`count{?X [?X1 ... ?Xn] | τ}`) = `count`($\bar{S}_{bag}$), if the function `count` is defined for $\bar{S}_{bag}$ in the dialect. If not, the value of the aggregate in $\boldsymbol{I}$ is indeterminate. The function `count` must compute the cardinality of finite bags.

f. *Sum aggregate*:
   - $\boldsymbol{I}$(`sum{?X [?X1 ... ?Xn] | τ}`) = `sum`($\bar{S}_{bag}$), if the function `sum` is defined for $\bar{S}_{bag}$ in the dialect. If not, the value of the aggregate in $\boldsymbol{I}$ is indeterminate. The function `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*:
   - $\boldsymbol{I}$(`prod{?X [?X1 ... ?Xn] | τ}`) = `prod`($\bar{S}_{bag}$), if the function `prod` is defined for $\bar{S}_{bag}$ in the dialect. If not, the value of the aggregate in $\boldsymbol{I}$ is indeterminate. The function `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*:
   - $\boldsymbol{I}$(`avg{?X [?X1 ... ?Xn] | τ}`) = `avg`($\bar{S}_{bag}$), if the function `avg` is defined for $\bar{S}_{bag}$ in the dialect. If not, the value of the aggregate in $\boldsymbol{I}$ is indeterminate. The function `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 `φ@r`, the mapping $\boldsymbol{I}$ is defined in Section Interpretation of Documents.

Note that the definitions of $\boldsymbol{I}_{NF}$, $\boldsymbol{I}_{frame}$, and of $\boldsymbol{I}$(x=y) imply that the terms with named arguments that differ only in the order of their arguments are mapped by $\boldsymbol{I}$ to the same element in the domain. 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->1 b->2 c->3) = t(c->3 a->1 b->2)` and `ex:o[ex:a->1 ex:b->"abc" ex:a->1] = ex:o[ex:b->"abc" ex:a->1]` are tautologies in RIF-FLD.

The effect of signatures. For every signature, `sg`, supported by a dialect, there is a subset $\boldsymbol{D}_{sg} \subseteq \boldsymbol{D}$, called the ***domain of the signature***. Terms that have a given signature, `sg`, must be mapped by $\boldsymbol{I}$ to $\boldsymbol{D}_{sg}$, 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 $\boldsymbol{D}_{sg} \subseteq \boldsymbol{D}_{sg'}$.
2. If k is a constant that has signature sg then $\boldsymbol{I}_C(k) \in \boldsymbol{D}_{sg}$.
3. If ?v is a variable that has signature sg then $\boldsymbol{I}_V(?v) \in \boldsymbol{D}_{sg}$.
4. If sg has an arrow expression of the form (s1 ... sn)⇒s then, for every d∈$\boldsymbol{D}_{sg}$, $\boldsymbol{I}_F$(d) must map $\boldsymbol{D}_{s1}\times \ldots \times\boldsymbol{D}_{sn}$ to $\boldsymbol{D}_s$.
5. If sg has an arrow expression of the form (p1->s1 ... pn->sn)⇒s then, for every d∈$\boldsymbol{D}_{sg}$, $\boldsymbol{I}_{NF}$(d) must map the set {<p1,$\boldsymbol{D}_{s1}$>, ..., <pn,$\boldsymbol{D}_{sn}$>} to $\boldsymbol{D}_s$.
6. If the signature -> has arrow expressions (sg,s_1,r_1)⇒k, ..., (sg,s_n,r_n)⇒k, then, for every d∈$\boldsymbol{D}_{sg}$, $\boldsymbol{I}_{frame}$(d) must map {<$\boldsymbol{D}_{s1}$,$\boldsymbol{D}_{r1}$>, ..., <$\boldsymbol{D}_{sn}$,$\boldsymbol{D}_{rn}$>} to $\boldsymbol{D}_k$.
7. If the signature # has an arrow expression `(s r)`⇒k then $\boldsymbol{I}_{isa}$ must map $\boldsymbol{D}_s\times\boldsymbol{D}_r$ to $\boldsymbol{D}_k$.
8. If the signature ## has an arrow expression `(s s)`⇒k then $\boldsymbol{I}_{sub}$ must map $\boldsymbol{D}_s\times\boldsymbol{D}_s$ to $\boldsymbol{D}_k$.
9. If the signature = has an arrow expression `(s s)`⇒k then $\boldsymbol{I}_=$ must map $\boldsymbol{D}_s\times\boldsymbol{D}_s$ to $\boldsymbol{D}_k$.

The effect of datatypes. The datatype identifiers in ***DTS*** impose the following restrictions. If `dt` ∈ ***DTS***, let $\boldsymbol{LS}_{dt}$ denote the lexical space of `dt`, $\boldsymbol{VS}_{dt}$ denote its value space, and $\boldsymbol{L}_{dt}$: $\boldsymbol{LS}_{dt} \rightarrow \boldsymbol{VS}_{dt}$ the lexical-to-value-space mapping. Then the following must hold:

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

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

RIF-FLD does not impose special requirements on $\boldsymbol{I}_C$ 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 `rif:local`) can be mapped to $\boldsymbol{VS}_{dt}$ of a datatype `dt`.

3.5 Annotations and the Formal Semantics

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

Note that although annotations associated with RIF-FLD 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 by 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, $\boldsymbol{I}$, determines the truth value $TVal_I(\varphi)$ of a RIF-FLD formula, φ, where φ 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, $TVal_I$, from the set of all non-document formulas to ***TV***. Note that the definition implies that $TVal_I(\varphi)$ is defined *only if* the set ***DTS*** of the datatypes of $\boldsymbol{I}$ includes all the datatypes mentioned in φ.

Definition (Truth valuation). ***Truth valuation*** for well-formed formulas in RIF-FLD is determined using the following function, denoted $TVal_{\mathbf{I}}$:

1. *Constants*: $TVal_{\mathbf{I}}$(k) = $\boldsymbol{I}_{\text{truth}}$($\boldsymbol{I}$(k)), if k ∈ `Const`.
2. *Variables*: $TVal_{\mathbf{I}}$(?v) = $\boldsymbol{I}_{\text{truth}}$($\boldsymbol{I}$(?v)), if ?v ∈ `Var`.
3. *Positional atomic formulas*: $TVal_{\mathbf{I}}$(`r(`t_1 `...` t_n`)`) = $\boldsymbol{I}_{\text{truth}}$($\boldsymbol{I}$(`r(`$t_1$ `...` t_n`)`)).
4. *Atomic formulas with named arguments*: $TVal_{\mathbf{I}}$(`p(`s_1`->`v_1 `...` s_k`->`v_k`)`) = $\boldsymbol{I}_{\text{truth}}$($\boldsymbol{I}$(`p(`$s_1$`->` v_1 `...` s_k`->`v_k`)`)).
5. *Equality*: $TVal_{\mathbf{I}}$(`x = y`) = $\boldsymbol{I}_{\text{truth}}$($\boldsymbol{I}$(`x = y`)).

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

   - $\boldsymbol{I}_{\text{truth}}$($\boldsymbol{I}$(`x = y`)) = **t** if $\boldsymbol{I}$(x) = $\boldsymbol{I}$(y) and $\boldsymbol{I}_{\text{truth}}$($\boldsymbol{I}$(`x = y`)) = **f** otherwise.
6. *Subclass*: $TVal_{\mathbf{I}}$(`sc ## cl`) = $\boldsymbol{I}_{\text{truth}}$($\boldsymbol{I}$(`sc ## cl`)).

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

   - For all well-formed terms `c1`, `c2`, `c3`: glb$_t$($TVal_{\mathbf{I}}$(`c1 ## c2`), $TVal_{\mathbf{I}}$(`c2 ## c3`)) $\le_t$ $TVal_{\mathbf{I}}$(`c1 ## c3`).

   Note that this is a restriction on $\boldsymbol{I}_{\text{truth}}$ and the mapping $\boldsymbol{I}$, which is expressed in a more succinct form using $TVal_{\mathbf{I}}$.

7. *Membership*: $TVal_{\mathbf{I}}$(`o # cl`) = $\boldsymbol{I}_{\text{truth}}$($\boldsymbol{I}$(`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`: glb$_t$($TVal_{\mathbf{I}}$(`o # cl`), $TVal_{\mathbf{I}}$(`cl ## scl`)) $\le_t$ $TVal_{\mathbf{I}}$(`o # scl`).

   Note that this is a restriction on $\boldsymbol{I}_{\text{truth}}$ and the mapping $\boldsymbol{I}$, which is expressed in a more succinct form using $TVal_{\mathbf{I}}$.

8. *Frame*: $TVal_{\mathbf{I}}$(`o[`a_1`->`v_1 `...` a_k`->`v_k`]`) = $\boldsymbol{I}_{\text{truth}}$($\boldsymbol{I}$(`o[`$a_1$`->`$v_1$ `...` a_k`->`v_k`]`)).

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

   - $TVal_{\mathbf{I}}$(`o[`a_1`->`v_1 `...` a_k`->`v_k`]`) = glb$_t$($TVal_{\mathbf{I}}$(`o[`a_1`->`v_1`]`), ..., $TVal_{\mathbf{I}}$(`o[`a_k`->`v_k`]`)).

   Observe that this is a restriction on $\boldsymbol{I}_{\text{truth}}$ and the mapping $\boldsymbol{I}$. For brevity, it is expressed in a more succinct form using $TVal_{\mathbf{I}}$.

9. *Externally defined atomic formula*: $TVal_{\mathbf{I}}$(`External(t loc)`) = $\boldsymbol{I}_{\text{truth}}$($\boldsymbol{I}_{\text{external}}$(σ)($\boldsymbol{I}$($s_1$), ..., $\boldsymbol{I}$(s_n))), if `External(t loc)` is an atomic formula that is an instantiation of the external schema `σ = (?`X_1 `...` `?`X_n`; τ; loc)` by substitution `?`X_1`/`s_1 `...` `?`X_n`/`s_n.

   Note that, by definition, `External(t 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 external schema, so $\boldsymbol{I}$(`External(t loc)`) is well-defined.

10. *Connectives and quantifiers*: if S is a connective or a quantifier and `S(`t_1 `...` t_n`)` is a well-formed formula term then $TVal_{\mathbf{I}}$(`S(`t_1 `...` t_n`)`) = $\boldsymbol{I}_{\text{truth}}$($\boldsymbol{I}$(`S(`$t_1$ `...` 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 $\boldsymbol{I}_{\text{truth}}$ and the mapping $\boldsymbol{I}$, which are expressed via $TVal_{\mathbf{I}}$, for brevity):

    a. *Conjunction*: $TVal_{\mathbf{I}}$(`And(`c_1 `...` c_n`)`) = glb$_t$($TVal_{\mathbf{I}}$(c_1), ..., $TVal_{\mathbf{I}}$(c_n)).

       The empty conjunction is treated as a tautology, so $TVal_{\mathbf{I}}$(`And()`) = **t**.

    b. *Disjunction*: $TVal_{\mathbf{I}}$(`Or(`c_1 ... c_n`)`) = lub$_t$($TVal_{\mathbf{I}}$(c_1), ..., $TVal_{\mathbf{I}}$(c_n)).

       The empty disjunction is treated as a contradiction, so $TVal_{\mathbf{I}}$(`Or()`) = **f**.

    c. *Negation*: $TVal_{\mathbf{I}}$(`Neg Neg φ`) = $TVal_{\mathbf{I}}$(φ) and $TVal_{\mathbf{I}}$(`Naf φ`) = ~$TVal_{\mathbf{I}}$(φ).

       The symbol ~ here is the self-inverse operator of negation on ***TV*** introduced in Section Truth Values.

       The symmetric negation, `Neg`, is sufficiently general to capture many different kinds of such negation. For instance, classical negation would, in addition, require $TVal_{\mathbf{I}}$(`Neg φ`) = ~$TVal_{\mathbf{I}}$(φ); strong negation (analogous to the one in [APP96]) can be characterized by $TVal_{\mathbf{I}}$(`Neg φ`) $\le_t$ ~$TVal_{\mathbf{I}}$(φ); and explicit negation (analogous to [APP96]) 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*:
       - $TVal_{\mathbf{I}}$(`Exists ?`v_1 `...` `?`v_n `(φ)`) = lub$_t$($TVal_{\mathbf{I}^*}$(φ)).
       - $TVal_{\mathbf{I}}$(`Forall ?`v_1 `...` `?`v_n `(φ)`) = glb$_t$($TVal_{\mathbf{I}^*}$(φ)).

       Here lub$_t$ (respectively, glb$_t$) is taken over all interpretations $\boldsymbol{I}$* of the form <***TV***, ***DTS***, ***D***, $\boldsymbol{I}_C$, $\boldsymbol{I}^*_V$, $\boldsymbol{I}_F$, $\boldsymbol{I}_{NF}$, $\boldsymbol{I}_{\text{list}}$, $\boldsymbol{I}_{\text{tail}}$, $\boldsymbol{I}_{\text{frame}}$, $\boldsymbol{I}_{\text{sub}}$, $\boldsymbol{I}_{\text{isa}}$, $\boldsymbol{I}_{=}$, $\boldsymbol{I}_{\text{external}}$, $\boldsymbol{I}_{\text{connective}}$, $\boldsymbol{I}_{\text{truth}}$>, which are exactly like $\boldsymbol{I}$, except that the mapping $\boldsymbol{I}^*_V$, is used instead of $\boldsymbol{I}_V$. $\boldsymbol{I}^*_V$ is defined to coincide with $\boldsymbol{I}_V$ on all variables except, possibly, on ?v_1,... ,?v_n.

    e. *Rule implication*:
       - $TVal_{\mathbf{I}}$(*head* :- *body*) = **t**, if $TVal_{\mathbf{I}}$(*head*) $\ge_t$ $TVal_{\mathbf{I}}$(*body*).
       - $TVal_{\mathbf{I}}$(*head* :- *body*) $<_t$ **t** otherwise.

    f. *Constraint*:
       - $TVal_{\mathbf{I}}$(:- *body*) = **t**, if $TVal_{\mathbf{I}}$(*body*) = **f**.
       - $TVal_{\mathbf{I}}$(:- *body*) $<_t$ **t** otherwise.

    g. Dialects that introduce additional connectives and quantifiers should define appropriate restrictions on $TVal_{\mathbf{I}}$ to give those new elements desired semantics.

11. *Groups of formulas*:

If Γ is a group formula of the form `Group(`φ_1 `...` φ_n`)` then

- $TVal_I(\Gamma) = \text{glb}_t(TVal_I(\varphi_1), ..., TVal_I(\varphi_n))$.

This means that a group of formulas is treated as a conjunction. In particular, the empty group is treated as a tautology, so $TVal_I$(`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 Δ be a document formula and `Import(`*loc*`)` be one of its import directives, where *loc* is a locator of another document formula, `Δ'`. In this case, we say that `Δ'` is ***directly imported*** into Δ.

A document formula `Δ'` is said to be ***imported*** into Δ if it is either directly imported into Δ or it is imported (directly or not) into another document, which itself is directly imported into Δ. ☐

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***, ρ, is a function that maps document formulas to document formulas subject to the following restriction:

- If ρ(Δ) = `Δ'` then `Δ'` is exactly like Δ except that all occurrences of some `rif:local` constants in Δ may be consistently renamed into other `rif:local` constants.
  By consistent renaming here we mean that different occurrences of the same `rif:local` constant in Δ are renamed identically. ☐

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

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

All these mappings are subject to the following restrictions:

- All semantic structures in $\hat{\mathbf{I}}_{set}$ have the same domain and the mappings $\boldsymbol{I}^{m_1}{}_C$, $\boldsymbol{I}^{m_2}{}_C$, ... are all the same. We will denote this mapping by $\hat{\mathbf{I}}_C$ and the common domain will be called *the domain of* $\hat{\mathbf{I}}$.
  Note that this implies that if `t` is a *ground* (i.e., variable-free) term then $\boldsymbol{I}^{m_1}$(`t`) = $\boldsymbol{I}^{m_2}$(`t`) = ..., and so we can write $\hat{\mathbf{I}}$(`t`) without ambiguity.
- The adornments m_1, m_2, ... are elements in the domain of $\hat{\mathbf{I}}$ and $\hat{\mathbf{I}}_{set}$ can have at most one semantic structure with a given adornment.
  Intuitively, the adornments represent module names and thus $\boldsymbol{I}^{m_i}$ is to be understood as the semantic structure for interpreting the documents associated with module m_i.
- If one document, Δ, imports another document, `Δ'`, either directly or indirectly then $\hat{\mathbf{I}}_{map}(\Delta) \subseteq \hat{\mathbf{I}}_{map}(\Delta')$.
  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(`*n loc*`)` appears in some document, where *loc* is a locator for another document, Δ, then
  - $\boldsymbol{I}^{\hat{\mathbf{I}}(n)} \in \hat{\mathbf{I}}_{map}(\Delta)$ ($\subseteq \hat{\mathbf{I}}_{set}$).
    Thus, in this case, $\hat{\mathbf{I}}(n)$ must adorn one (and only one) semantic structure in $\hat{\mathbf{I}}_{set}$.

  Intuitively, these conditions say that if some document associates Δ with a particular module then $\hat{\mathbf{I}}_{map}$ must respect that. Note that different directives `Module(`*n loc*$_1$`)`, ..., `Module(`*n loc*$_k$`)` (with different locators but the same module name) have the effect that multiple documents become associated with the same module 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 Δ be a document formula and let `Module(`*n loc*`)` be one of its remote module directives, where *loc* is a locator for another document formula, `Δ'`. In this case, we say that `Δ'` is a ***directly linked remote module*** of Δ.

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

In the definition of term-interpreting mappings, we postponed defining remote term references till now. The next definitions fill in this gap. The reason why 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 $TVal_{\boldsymbol{I}/\hat{\mathbf{I}}}(\varphi)$, the notion of truth valuation in the context of a semantic multi-structure $\hat{\mathbf{I}}$.

Definition (Term-interpreting mapping for remote term references). Let φ be a non-document formula, $\hat{\mathbf{I}}$ a semantic multi-structure, and $\boldsymbol{I} \in \hat{\mathbf{I}}_{set}$.

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

However, if φ *is* a remote reference ψ`@`r then $\boldsymbol{I}_{\hat{\mathbf{I}}}(\varphi)$ is defined as follows:

Let $\boldsymbol{I}$(r) = d. If $\mathbf{\hat{I}}_{set}$ has *no* semantic structure adorned with d, the value of $\boldsymbol{I}_{\mathbf{\hat{I}}}(\varphi)$ is indeterminate (i.e., it can be any element of the domain of $\mathbf{\hat{I}}$). Otherwise, let $\boldsymbol{J} \in \mathbf{\hat{I}}_{set}$ be the structure adorned with d (it is then unique, by definition). In that case, we define:

- $\boldsymbol{I}_{\mathbf{\hat{I}}}(\varphi)$ to be $\boldsymbol{J}_{\mathbf{\hat{I}}}(\psi)$
- $TVal_{\boldsymbol{I}/\mathbf{\hat{I}}}(\varphi) = \boldsymbol{J}_{truth}(\boldsymbol{I}_{\mathbf{\hat{I}}}(\varphi))$ ☐

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

Definition (Truth valuation of formulas in multi-document structures). Let Δ be a document formula and let $\Delta_1, \ldots, \Delta_n, \ldots$ be all the RIF-FLD document formulas that are *imported* into Δ or linked to it (directly or indirectly). Let furthermore $\Gamma, \Gamma_1, \ldots, \Gamma_n, \ldots$ be the respective group formulas associated with these documents. Let $\mathbf{\hat{I}} = (\mathbf{\hat{I}}_{ren}, \mathbf{\hat{I}}_{map}, \mathbf{\hat{I}}_{set})$ be a semantic multi-structure where $\mathbf{\hat{I}}_{set} = \{\boldsymbol{I}^{m_1}, \boldsymbol{I}^{m_2}, \ldots\}$.

We define the truth valuation for Δ as follows.

- $TVal_{\mathbf{\hat{I}}}(\Delta) = glb_t\{ glb_t\{TVal_{\boldsymbol{J}/\mathbf{\hat{I}}}(\mathbf{\hat{I}}_{ren}(\Gamma)) \mid \boldsymbol{J} \in \mathbf{\hat{I}}_{map}(\Delta)\},\ glb_t\{TVal_{\boldsymbol{J}/\mathbf{\hat{I}}}(\mathbf{\hat{I}}_{ren}(\Gamma_1)) \mid \boldsymbol{J} \in \mathbf{\hat{I}}_{map}(\Delta_1)\},\ \ldots,\ glb_t\{TVal_{\boldsymbol{J}/\mathbf{\hat{I}}}(\mathbf{\hat{I}}_{ren}(\Gamma_n)) \mid \boldsymbol{J} \in \mathbf{\hat{I}}_{map}(\Delta_n)\},\ \ldots \}$.

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 $\mathbf{\hat{I}}$. Suppose ψ@r is such a term that occurs somewhere in a document, Δ. If $\boldsymbol{J} \in \mathbf{\hat{I}}_{map}(\Delta)$ is an ordinary semantic structure assigned to Δ then $\boldsymbol{J}$(r) determines the semantic structure in which ψ is to be evaluated: it is the structure in $\mathbf{\hat{I}}_{set}$ adorned with $\boldsymbol{J}$(r), say $\boldsymbol{M}$. This is what $\boldsymbol{J}_{\mathbf{\hat{I}}}$(ψ@r) is all about. Note that ψ may also contain remote term references (as ψ may be a compound formula). In this case, since ψ is evaluated using $\boldsymbol{M}_{\mathbf{\hat{I}}}$, the same principle applies.

Definition (Models). Let $\boldsymbol{I}$ be a semantic structure or a multi-structure. We say that $\boldsymbol{I}$ is a ***model*** of a formula, φ, written as $\boldsymbol{I} \models \varphi$, iff $TVal_{\boldsymbol{I}}(\varphi)$ = **t**. Here φ is a document formula, if $\boldsymbol{I}$ is a multi-structure, and a non-document formula, if $\boldsymbol{I}$**'** is an ordinary semantic structure. ☐

3.8 Intended Semantic Structures

The ***semantics of a set of formulas***, Γ, 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], 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, Γ, is the unique minimal Herbrand model [Lloyd87] of Γ. For dialects in which rule bodies may contain literals negated with the default negation connective `Naf`, only *some* of the minimal Herbrand models of Γ 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 Γ consists of a single rule formula `p :- Naf q`. If `Naf` were interpreted as classical negation, then this rule would be simply equivalent to `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 :- 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 `Or(p q)`. The logic underlying rule-based systems also assumes that only the *minimal* Herbrand models are intended (minimality here is with respect to the set of true facts). Furthermore, although our example has two minimal Herbrand models -- one where p is true and q is false, and the other where p is false, but 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 φ and ψ be RIF-FLD formulas. We say that φ ***entails*** ψ, written as `φ |= ψ`, if and only if the following holds:

- *φ and ψ are both document formulas*:
  for every intended semantic multi-structure $\mathbf{\hat{I}}$ it is the case that $TVal_{\mathbf{\hat{I}}}(\varphi) \le_t TVal_{\mathbf{\hat{I}}}(\psi)$.
- *φ and ψ are both non-document formulas*:
  for every intended semantic multi-structure $\mathbf{\hat{I}}$ and for every $\boldsymbol{J} \in \mathbf{\hat{I}}$ it is the case that $TVal_{\boldsymbol{J}/\mathbf{\hat{I}}}(\varphi) \le_t TVal_{\boldsymbol{J}/\mathbf{\hat{I}}}(\psi)$.
  Intuitively, this means that φ must entail ψ in any module the two formulas might happen to be.
- *φ is a document formula, but ψ is not*:
  for every intended semantic multi-structure $\mathbf{\hat{I}}$ and every $\boldsymbol{J} \in \mathbf{\hat{I}}_{map}(\varphi)$ it is the case that $TVal_{\mathbf{\hat{I}}}(\varphi) \le_t TVal_{\boldsymbol{J}/\mathbf{\hat{I}}}(\psi)$.
  Intuitively, this means that φ must entail ψ in any module associated with φ. ☐

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.

Note that one consequence of the multi-document semantics is that local constants specified in one document cannot be queried from another document. For instance, if one document, `Δ'`, has the fact `"http://example.com/ppp"^^rif:iri("abc"^^rif:local)` while another document formula, `Δ`, imports `Δ'` and has the rule `"http://example.com/qqq"^^rif:iri(?X) :- "http://example.com/ppp"^^rif:iri(?X)` , then `Δ |= "http://example.com/qqq"^^rif:iri("abc"^^rif:local)` does *not* hold. This is because the symbol `"abc"^^rif:local` in `Δ'` and `Δ` is treated as different constants due to the process of renaming apart that takes place prior to truth valuation.

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

4 XML Serialization Framework

The RIF-FLD XML serialization framework defines

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

As explained in the overview section, the design of RIF envisions that the presentation syntaxes of future logic RIF dialects will be specializations of the presentation syntax of RIF-FLD. This means that every well-formed formula in the presentation syntax of a standard logic RIF dialect must also be well-formed in a specialization of RIF-FLD, which includes actualizing the RIF-FLD extension points (see overview section). The goal of the XML serialization framework is to provide a similar yardstick for the RIF XML syntax. This amounts to the requirement that any admissible XML document for a logic RIF dialect must also be an admissible XML document for a specialized RIF-FLD (*admissibility* is defined below). In terms of the presentation-to-XML syntax mappings, this means that each mapping for a logic RIF dialect must be a restriction of the corresponding mapping for RIF-FLD. For instance, the mapping from the presentation syntax of RIF-BLD to XML in [RIF-BLD] is a restriction of the presentation-syntax-to-XML mapping for RIF-FLD. In this way, RIF-FLD provides a framework for extensibility and mutual compatibility between XML syntaxes of RIF dialects.

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

Definition (Specialization of RIF-FLD schema to a dialect schema). If a dialect, *D*, specializes RIF-FLD then its XML schema must be a ***specialization of the XML schema*** of RIF-FLD. This includes elimination of some elements and attributes, restriction of the XML types of the others, and replacement of the extension points with appropriate concrete elements of the specified (possibly restricted) types. ☐

Definition (Valid XML document in RIF-FLD). A ***valid RIF-FLD document*** in the XML syntax is an XML document that is valid with respect to the XML schema in Appendix XML Schema for RIF-FLD, where the extension points `NEWCONNECTIVE`, `NEWQUANTIFIER`, `NEWAGGRFUNC`, and `NEWTERM` are specialized as concrete elements of the types prescribed by the RIF-FLD XML schema.

If a dialect, *D*, specializes RIF-FLD then a ***valid XML document in dialect*** *D* is one that is valid with respect to the specialized XML schema of *D*. ☐

Definition (Admissible XML document in a logic dialect). An ***admissible*** RIF-FLD document in the XML syntax is a valid FLD document in that syntax that is the image of a well-formed RIF-FLD document in the presentation syntax (see Definition Well-formed formula) under the presentation-to-XML syntax mapping χ_{fld} defined in Section Mapping from the RIF-FLD Presentation Syntax to the XML Syntax.

If a dialect, *D*, specializes RIF-FLD then an XML document is admissible with respect to *D* if and only if it is a valid document in *D* and it is an image under χ_D of a well-formed document in the presentation syntax of *D*, where χ_D is the presentation-to-XML mapping defined by the dialect *D*.

Note that if *D* requires the directive `Dialect(D)` as part of its syntax then this implies that any *D*-admissible document must have this directive. ☐

A ***round-tripping*** of an admissible document in a dialect, *D*, is a semantics-preserving mapping to a document in any language *L* followed by a semantics-preserving mapping from the *L*-document back to an admissible *D*-document. While semantically equivalent, the original and the round-tripped *D*-documents need not be identical.

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 ANYURICONST)
- 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)
```

```
- formula   (formula role, containing a FORMULA)
- termula   (termula role, containing a TERMULA)
- Implies   (implication, containing if and then roles)
- if        (antecedent role, containing FORMULA)
- then      (consequent role, containing FORMULA)
- And       (conjunction)
- Or        (disjunction)
- Neg       (strong negation, containing a formula role)
- Naf       (default negation, containing a formula role)
- Atom      (atom formula, positional or with named arguments)
- Remote    (prefix version of remote term '@', containing a formula/termula and an internal role)
- External  (external call, containing a content role)
- content   (content role, containing an Atom, for predicates, or Expr, for functions)
- Member    (member formula)
- Subclass  (subclass formula)
- Frame     (Frame formula)
- object    (Member/Frame role containing a TERM or an object description)
- op        (Atom/Expr role for predicates/functions as operations)
- args      (Atom/Expr positional arguments role, with fixed 'ordered' attribute, containing n TERMs)
- instance  (Member instance role)
- class     (Member class role)
- sub       (Subclass sub-class role)
- super     (Subclass super-class role)
- slot      (Atom/Expr or Frame slot role, with fixed 'ordered' attribute, containing a Name or TERM followed by a TERM)
- Equal     (prefix version of term equation '=')
- left      (Equal left-hand side role)
- right     (Equal right-hand side role)
- Expr      (expression formula, positional or with named arguments)
- List      (list term, closed or open)
- items     (list items role, with ordered="yes" attribute, containing n TERMs)
- rest      (list rest role, corresponding to '|')
- Min       (aggregate function)
- Max       (aggregate function)
- Sum       (aggregate function)
- Prod      (aggregate function)
- Avg       (aggregate function)
- Count     (aggregate function)
- Set       (aggregate function)
- Bag       (aggregate function)
- Const     (individual, function, or predicate symbol, with optional 'type' attribute)
- Name      (name of named argument)
- Var       (logic variable)

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

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:

  Document(
   Dialect(FOL)
   Base(<http://www.shakespeare-literature.com/Hamlet/>)
   Prefix(dc     <http://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
      )
   )
  )


XML serialization:

<!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
        xml:base="http://www.shakespeare-literature.com/Hamlet/"
        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>
           <formula>
             <Member>
               <instance><Var>X</Var></instance>
               <class><Const type="&rif;iri">ex:RottenThing</Const></class>
             </Member>
           </formula>
           <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>
         </And>
       </formula>
     </Exists>
    </sentence>
    <sentence>
     <Forall>
       <declare><Var>X</Var></declare>
       <formula>
         <Or>
           <formula>
             <Atom>
               <op><Const type="&rif;iri">tobe</Const></op>
               <args ordered="yes"><Var>X</Var></args>
             </Atom>
           </formula>
           <formula>
             <Naf>
               <formula>
                 <Atom>
                   <op><Const type="&rif;iri">tobe</Const></op>
                   <args ordered="yes"><Var>X</Var></args>
                 </Atom>
               </formula>
             </Naf>
           </formula>
         </Or>
       </formula>
     </Forall>
    </sentence>
    <sentence>
     <Forall>
```

```
      <declare><Var>X</Var></declare>
      <formula>
        <Implies>
          <if>
            <Member>
              <instance><Var>X</Var></instance>
              <class><Const type="&rif;iri">ex:man</Const></class>
            </Member>
          </if>
          <then>
            <And>
              <formula>
                <Exists>
                  <declare><Var>B</Var></declare>
                  <formula>
                   <And>
                    <formula>
                      <Atom>
                        <op><Const type="&rif;iri">ex:has</Const></op>
                        <args>
                          <Var>X</Var>
                          <Var>B</Var>
                        </args>
                      </Atom>
                    </formula>
                    <formula>
                      <Member>
                        <instance><Var>B</Var></instance>
                        <class><Const type="&rif;iri">ex:business</Const></class>
                      </Member>
                    </formula>
                  </And>
                 </formula>
                </Exists>
              </formula>
              <formula>
                <Exists>
                  <declare><Var>D</Var></declare>
                  <formula>
                   <And>
                    <formula>
                      <Atom>
                        <op><Const type="&rif;iri">ex:has</Const></op>
                        <args>
                          <Var>X</Var>
                          <Var>D</Var>
                        </args>
                      </Atom>
                    </formula>
                    <formula>
                      <Member>
                        <instance><Var>D</Var></instance>
                        <class><Const type="&rif;iri">ex:desire</Const></class>
                      </Member>
                    </formula>
                  </And>
                 </formula>
                </Exists>
              </formula>
            </And>
          </then>
        </Implies>
      </formula>
    </Forall>
  </sentence>
  <sentence>
    <Group>
      <meta>
        <Frame>
          <object>
            <Const type="&rif;iri">facts</Const>
          </object>
        </Frame>
      </meta>
      <sentence>
        <Member>
          <instance><Const type="&rif;iri">Yorick</Const></instance>
          <class><Const type="&rif;iri">ex:poor</Const></class>
        </Member>
      </sentence>
      <sentence>
        <Member>
          <instance><Const type="&rif;iri">Hamlet</Const></instance>
          <class><Const type="&rif;iri">ex:prince</Const></class>
        </Member>
```

```
        </sentence>
      </Group>
     </sentence>
    </Group>
   </payload>
  </Document>
```

4.2 Mapping from the RIF-FLD Presentation Syntax to the XML Syntax

This section defines a normative mapping, χ_{fld}, from the presentation syntax of Section EBNF Grammar for the Presentation 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 χ_{fld} to these metavariables. When an expression χ_{fld}(***metavar***) occurs in an XML pattern in the right column of a translation table, it should be understood as a recursive application of χ_{fld} to the presentation syntax represented by the metavariable. The XML syntax result of such an application is substituted for the expression χ_{fld}(***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 χ_{fld} 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 χ_{fld}(Presentation) = XML. The function *remove-outer-quotes* used in the translation removes enclosing double quotes from a string and leaves unquoted strings untouched. Since the presentation syntax of RIF-FLD is context sensitive, the mapping must differentiate between the terms that occur in the position of the individuals and the terms that occur as atomic formulas. To this end, in the translation table, the positional and named-argument terms that occur in the context of atomic formulas are denoted by the expressions of the form ***pred***(...) and the terms that occur as individuals are denoted by expressions of the form ***func***(...). In the table, each metavariable for an (unnamed) positional ***argument$_i$*** is assumed to be instantiated to values unequal to the instantiations of named arguments ***name$_j$*** -> ***filler$_j$***. Regarding the last but first row, we assume that shortcuts for constants [RIF-DTB] have already been expanded to their full form ("..."^^***symspace***). The ***AGGRFUNC*** metavariable stands for any of the aggregation functions Min, Max, Count, Avg, Sum, Prod, Set, Bag, or NEWAGGRFUNC.

Thus, the mapping of the extension point for aggregate functions (NEWAGGRFUNC) is handled by the ***AGGRFUNC*** metavariable, along with the mapping of the specific aggregate functions (Min etc.). The mapping of the extension points for quantifiers (NEWQUANTIFIER) and connectives (NEWCONNECTIVE) generalizes the mapping for the specific quantifiers (Forall, Exists) and connectives (And, Or), respectively. The mapping of the extension point for terms (NEWTERM) keeps 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 NEWTERM is left entirely up to RIF dialects. Recall that the extension point for symbols (NEWSYMBOL) is part of the alphabet and is not dealt with in the EBNF and XML grammars.

Also recall that OpenList(t_1 ... t_m t), m≥1, is just an alternative form for List(t_1 ... t_m | t), so its mapping is not represented separately.

Note that the Import and Dialect directives are handled by the presentation-to-XML syntax mapping, using an XML attribute for dialect names (values: FOL, 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 χ_{fld} 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 [XML1.0] 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.

Presentation Syntax	XML Syntax

Document(
  Dialect(***name***)***?***
  Import(***iloc$_1$*** ***prfl$_1$?***)
   . . .
  Import(***iloc$_n$*** ***prfl$_n$?***)
  Module(***name$_1$*** ***mloc$_1$***)
   . . .
  Module(***name$_k$*** ***mloc$_k$***)
  group
)
n ≥ 0, k ≥ 0
	<Document dialect="***name***"***?***>
	  <directive>
	    <Import>
	      <location>χ_{fld}(***iloc$_1$***)</location>
	      <profile>χ_{fld}(***prfl$_1$***)</profile>***?***
	    </Import>
	  </directive>
	   . . .
	  <directive>
	    <Import>
	      <location>χ_{fld}(***iloc$_n$***)</location>
	      <profile>χ_{fld}(***prfl$_n$***)</profile>***?***
	    </Import>
	  </directive>
	  <directive>
	    <Module>
	      <internal>χ_{fld}(***name$_1$***)</internal>
	      <location>χ_{fld}(***mloc$_1$***)</location>
	    </Module>
	  </directive>
	   . . .
	  <directive>
	    <Module>
	      <internal>χ_{fld}(***name$_k$***)</internal>
	      <location>χ_{fld}(***mloc$_k$***)</location>
	    </Module>
	  </directive>

	<payload>χ_{fld}(**$group$**)</payload> </Document>
Group(**$clause_1$** . . . **$clause_n$**) n ≥ 0	<Group> <sentence>χ_{fld}(**$clause_1$**)</sentence> . . . <sentence>χ_{fld}(**$clause_n$**)</sentence> </Group>
Forall **$variable_1$** . . . **$variable_n$** (**$body$**) n ≥ 0	<Forall> <declare>χ_{fld}(**$variable_1$**)</declare> . . . <declare>χ_{fld}(**$variable_n$**)</declare> <formula>χ_{fld}(**$body$**)</formula> </Forall>
Exists **$variable_1$** . . . **$variable_n$** (**$body$**) n ≥ 0	<Exists> <declare>χ_{fld}(**$variable_1$**)</declare> . . . <declare>χ_{fld}(**$variable_n$**)</declare> <formula>χ_{fld}(**$body$**)</formula> </Exists>
NEWQUANTIFIER **$variable_1$** . . . **$variable_n$** (**$body$**) n ≥ 0	<NEWQUANTIFIER> <declare>χ_{fld}(**$variable_1$**)</declare> . . . <declare>χ_{fld}(**$variable_n$**)</declare> <formula>χ_{fld}(**$body$**)</formula> </NEWQUANTIFIER>
conclusion :- ***condition***	<Implies> <if>χ_{fld}(**$condition$**)</if> <then>χ_{fld}(**$conclusion$**)</then> </Implies>
And (**$conjunct_1$** . . . **$conjunct_n$**) n ≥ 0	<And> <formula>χ_{fld}(**$conjunct_1$**)</formula> . . . <formula>χ_{fld}(**$conjunct_n$**)</formula> </And>
Or (**$disjunct_1$** . . . **$disjunct_n$**) n ≥ 0	<Or> <formula>χ_{fld}(**$disjunct_1$**)</formula> . . . <formula>χ_{fld}(**$disjunct_n$**)</formula> </Or>
NEWCONNECTIVE (**$argument_1$** . . . **$argument_n$**) n ≥ 0	<NEWCONNECTIVE> <formula>χ_{fld}(**$argument_1$**)</formula> . . . <formula>χ_{fld}(**$argument_n$**)</formula> </NEWCONNECTIVE>
Neg ***form***	<Neg> <formula>χ_{fld}(**$form$**)</formula> </Neg>
Naf ***form***	<Naf> <formula>χ_{fld}(**$form$**)</formula> </Naf>

query @ ***modref***	<Remote> <formula>χ_{fld}(***query***)</formula> <internal>χ_{fld}(***modref***)</internal> </Remote>
External (***atomframexpr***)	<External> <content>χ_{fld}(***atomframexpr***)</content> </External>
pred ( )	<Atom> <op>χ_{fld}(***pred***)</op> </Atom>
pred (***argument***$_1$ . . . ***argument***$_m$) m ≥ 1	<Atom> <op>χ_{fld}(***pred***)</op> <args ordered="yes"> χ_{fld}(***argument***$_1$) . . . χ_{fld}(***argument***$_m$) </args> </Atom>
func ( )	<Expr> <op>χ_{fld}(***func***)</op> </Expr>
func (***argument***$_1$ . . . ***argument***$_m$) m ≥ 1	<Expr> <op>χ_{fld}(***func***)</op> <args ordered="yes"> χ_{fld}(***argument***$_1$) . . . χ_{fld}(***argument***$_m$) </args> </Expr>
List (***element***$_1$ . . . ***element***$_n$) n ≥ 0	<List> <items ordered="yes"> χ_{fld}(***element***$_1$) . . . χ_{fld}(***element***$_n$) </items> </List>
List (***element***$_1$ . . . ***element***$_n$ | ***remainder***) n ≥ 1	<List> <items ordered="yes"> χ_{fld}(***element***$_1$) . . . χ_{fld}(***element***$_n$) </items> <rest>χ_{fld}(***remainder***)</rest> </List>
pred (***name***$_1$ -> ***filler***$_1$ . . . ***name***$_n$ -> ***filler***$_n$) n ≥ 0	<Atom> <op>χ_{fld}(***pred***)</op> <slot ordered="yes"> <Name>χ_{bld}(***name***$_1$)</Name> χ_{fld}(***filler***$_1$) </slot> . . . <slot ordered="yes"> <Name>χ_{bld}(***name***$_n$)</Name> χ_{fld}(***filler***$_n$) </slot> </Atom>
func (***name***$_1$ -> ***filler***$_1$ . . . ***name***$_n$ -> ***filler***$_n$) n ≥ 0	<Expr> <op>χ_{fld}(***func***)</op> <slot ordered="yes"> <Name>χ_{bld}(***name***$_1$)</Name> χ_{fld}(***filler***$_1$) </slot> . . .

	<slot ordered="yes"> <Name>χ_{bld}(***name***$_n$)</Name> χ_{fld}(***filler***$_n$) </slot> </Expr>
inst [***key***$_1$ -> ***filler***$_1$ . . . ***key***$_n$ -> ***filler***$_n$] n ≥ 0	<Frame> <object>χ_{fld}(***inst***)</object> <slot ordered="yes"> χ_{fld}(***key***$_1$) χ_{fld}(***filler***$_1$) </slot> . . . <slot ordered="yes"> χ_{fld}(***key***$_n$) χ_{fld}(***filler***$_n$) </slot> </Frame>
inst # class	<Member> <instance>χ_{fld}(***inst***)</instance> <class>χ_{fld}(***class***)</class> </Member>
sub ## super	<Subclass> _{χ_{fld}(***sub***)} <super>χ_{fld}(***super***)</super> </Subclass>
left = ***right***	<Equal> <left>χ_{fld}(***left***)</left> <right>χ_{fld}(***right***)</right> </Equal>

AGGRFUNC {
variable
variable$_1$
. . .
variable$_m$
|
compform
}
m ≥ 0
	<***AGGRFUNC***>
	<declare>χ_{fld}(***variable***)</declare>
	<declare>χ_{fld}(***variable***$_1$)</declare>
	. . .
	<declare>χ_{fld}(***variable***$_m$)</declare>
	<formula>χ_{fld}(***compform***)</formula>
	</***AGGRFUNC***>

"***unicodestring***"^^***space***	<Const type="***space***">***unicodestring***</Const>
?***name***$_1$	<Name>χ_{bld}(***name***$_1$)</Name>
NEWTERM	<NEWTERM>...</NEWTERM>
name$_i$	*remove-outer-quotes*(***name***$_i$)

4.2.2 Mapping of RIF-FLD Annotations

The χ_{fld} 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, **$**, stands for any of the binary infix operator names #, ##, =, :-, or @, while ***Binop*** stands for their respective class names Member, Subclass, Equal, Implies, or Remote. The metavariable ***attr?*** is used with ***Typetag*** to capture the optional dialect attribute (with its value) of Document. Again, each metavariable for an (unnamed) positional ***argument***$_i$ is assumed to be instantiated to values unequal to the instantiations of named arguments ***name***$_j$ -> ***filler***$_j$.

Presentation Syntax	XML Syntax
(* ***const? frameconj?*** *) ***Typetag*** (***e***$_1$ . . . ***e***$_n$) n ≥ 0	<***Typetag attr?***> <id>χ_{fld}(***const***)</id>***?*** <meta>χ_{fld}(***frameconj***)</meta>***?*** ***e***$_1$' . . . ***e***$_n$'

	</*Typetag*> where *attr*, e_1', . . ., e_n' are defined by the equation χ_{fld}(*Typetag*(e_1 . . . e_n)) = <*Typetag attr?*>e_1' . . . e_n'</*Typetag*>
(* *const? frameconj?* *) *Quantifier* *variable*$_1$ . . . *variable*$_n$ (*body*) n ≥ 0	<*Quantifier*> <id>χ_{fld}(*const*)</id>*?* <meta>χ_{fld}(*frameconj*)</meta>*?* <declare>χ_{fld}(*variable*$_1$)</declare> . . . <declare>χ_{fld}(*variable*$_n$)</declare> <formula>χ_{fld}(*body*)</formula> </*Quantifier*>
(* *const? frameconj?* *) *Negation e*	<*Negation*> <id>χ_{fld}(*const*)</id>*?* <meta>χ_{fld}(*frameconj*)</meta>*?* χ_{fld}(*e*) </*Negation*>
(* *const? frameconj?* *) *pred* (*argument*$_1$ . . . *argument*$_n$) n ≥ 0	<Atom> <id>χ_{fld}(*const*)</id>*?* <meta>χ_{fld}(*frameconj*)</meta>*?* <op>χ_{fld}(*pred*)</op> <args ordered="yes"> χ_{fld}(*argument*$_1$) . . . χ_{fld}(*argument*$_n$) </args> </Atom>
(* *const? frameconj?* *) *func* (*argument*$_1$ . . . *argument*$_n$) n ≥ 0	<Expr> <id>χ_{fld}(*const*)</id>*?* <meta>χ_{fld}(*frameconj*)</meta>*?* <op>χ_{fld}(*func*)</op> <args ordered="yes"> χ_{fld}(*argument*$_1$) . . . χ_{fld}(*argument*$_n$) </args> </Expr>
(* *const? frameconj?* *) List (*element*$_1$ . . . *element*$_n$) n ≥ 0	<List> <id>χ_{fld}(*const*)</id>*?* <meta>χ_{fld}(*frameconj*)</meta>*?* <items ordered="yes"> χ_{fld}(*element*$_1$) . . . χ_{fld}(*element*$_n$) </items> </List>
(* *const? frameconj?* *) List (*element*$_1$ . . . *element*$_n$ | *remainder*) n ≥ 1	<List> <id>χ_{fld}(*const*)</id>*?* <meta>χ_{fld}(*frameconj*)</meta>*?* <items ordered="yes"> χ_{fld}(*element*$_1$) . . . χ_{fld}(*element*$_n$) </items> <rest>χ_{fld}(*remainder*)</rest> </List>
(* *const? frameconj?* *) *pred* (*name*$_1$ -> *filler*$_1$ . . . *name*$_n$ -> *filler*$_n$) n ≥ 0	<Atom> <id>χ_{fld}(*const*)</id>*?* <meta>χ_{fld}(*frameconj*)</meta>*?* <op>χ_{fld}(*pred*)</op> <slot ordered="yes"> <Name>χ_{bld}(*name*$_1$)</Name> χ_{fld}(*filler*$_1$) </slot> . . . <slot ordered="yes"> <Name>χ_{bld}(*name*$_n$)</Name>

	χ_{fld}(***filler***$_n$) </slot> </Atom>
(* ***const? frameconj?*** *) ***func*** (***name***$_1$ -> ***filler***$_1$ . . . ***name***$_n$ -> ***filler***$_n$) n ≥ 0	<Expr> <id>χ_{fld}(***const***)</id>***?*** <meta>χ_{fld}(***frameconj***)</meta>***?*** <op>χ_{fld}(***func***)</op> <slot ordered="yes"> <Name>χ_{bld}(***name***$_1$)</Name> χ_{fld}(***filler***$_1$) </slot> . . . <slot ordered="yes"> <Name>χ_{bld}(***name***$_n$)</Name> χ_{fld}(***filler***$_n$) </slot> </Expr>
(* ***const? frameconj?*** *) ***inst*** [***key***$_1$ -> ***filler***$_1$ . . . ***key***$_n$ -> ***filler***$_n$] n ≥ 0	<Frame> <id>χ_{fld}(***const***)</id>***?*** <meta>χ_{fld}(***frameconj***)</meta>***?*** <object>χ_{fld}(***inst***)</object> <slot ordered="yes"> χ_{fld}(***key***$_1$) χ_{fld}(***filler***$_1$) </slot> . . . <slot ordered="yes"> χ_{fld}(***key***$_n$) χ_{fld}(***filler***$_n$) </slot> </Frame>
(* ***const? frameconj?*** *) ***e***$_1$ **$** ***e***$_2$	<***Binop***> <id>χ_{fld}(***const***)</id>***?*** <meta>χ_{fld}(***frameconj***)</meta>***?*** ***e***$_1$***'*** ***e***$_2$***'*** </***Binop***> where ***Binop***, ***e***$_1$***'***, ***e***$_2$***'*** are defined by the equation χ_{fld}(***e***$_1$ **$** ***e***$_2$) = <***Binop***>***e***$_1$***'*** ***e***$_2$***'***</***Binop***>
(* ***const? frameconj?*** *) ***unicodestring^^symspace***	<Const type="***symspace***"> <id>χ_{fld}(***const***)</id>***?*** <meta>χ_{fld}(***frameconj***)</meta>***?*** ***unicodestring*** </Const>
(* ***const? frameconj?*** *) ?***name***$_1$	<Var> <id>χ_{fld}(***const***)</id>***?*** <meta>χ_{fld}(***frameconj***)</meta>***?*** χ_{bld}(***name***$_1$) </Var>

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

- it is a formula in the dialect *D*,
- all datatypes and symbol spaces used in φ are in T, and
- all externally defined terms used in φ are instantiations of some external schemas in E.

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

Formally, this means that for any pair φ, ψ of $D_{T,E}$ formulas for which $\varphi \models_D \psi$ is defined, $\varphi \models_D \psi$ iff $\mu(\varphi) \models_L \mu(\psi)$. 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*** $D_{T,E}$ ***producer*** iff it implements a semantics-preserving mapping, ν, from the language *L* of the processor to the set of all $D_{T,E}$ formulas.

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

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

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

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

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

[RDF-SCHEMA]
RDF Vocabulary Description Language 1.0: RDF Schema, Brian McBride, Editor, W3C Recommendation 10 February 2004, http://www.w3.org/TR/rdf-schema/.

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

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

[RIF-BLD]
RIF Basic Logic Dialect Harold Boley, Michael Kifer, eds. W3C Proposed Edited Recommendation, 11 December 2012, http://www.w3.org/TR/2012/PER-rif-bld-20121211/. Latest version available at http://www.w3.org/TR/rif-bld/.

[RIF-Core]
RIF Core Dialect Harold Boley, Gary Hallmark, Michael Kifer, Adrian Paschke, Axel Polleres, Dave Reynolds, eds. W3C Proposed Edited Recommendation, 11 December 2012, http://www.w3.org/TR/2012/PER-rif-core-20121211/. Latest version available at http://www.w3.org/TR/rif-core/.

[RIF-DTB]
RIF Datatypes and Built-Ins 1.0 Axel Polleres, Harold Boley, Michael Kifer, eds. W3C Proposed Edited Recommendation, 11 December 2012, http://www.w3.org/TR/2012/PER-rif-dtb-20121211/. Latest version available at http://www.w3.org/TR/rif-dtb/.

[RIF-PRD]
RIF Production Rule Dialect Christian de Sainte Marie, Gary Hallmark, Adrian Paschke, eds. W3C Proposed Edited Recommendation, 11 December 2012, http://www.w3.org/TR/2012/PER-rif-prd-20121211/. Latest version available at http://www.w3.org/TR/rif-prd/.

[RIF-RDF+OWL]
RIF RDF and OWL Compatibility Jos de Bruijn, Chris Welty, eds. W3C Proposed Edited Recommendation, 11 December 2012, http://www.w3.org/TR/2012/PER-rif-rdf-owl-20121211/. Latest version available at http://www.w3.org/TR/rif-rdf-owl/.

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

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

[XML-Names]
Namespaces in XML 1.1 (Second Edition), W3C Recommendation, World Wide Web Consortium, 16 August 2006. This version is http://www.w3.org/TR/2006/REC-xml-names11-20060816. The latest version is available at http://www.w3.org/TR/xml-names11/.

7.2 Informational References

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

[APP96]
Strong and Explicit Negation in Non-Monotonic Reasoning and Logic Programming, J.J. Alferes, L.M. Pereira, and T.C. Przymusinski. Lecture Notes In Computer Science, vol. 1126. Proceedings of the European Workshop on Logics in Artificial Intelligence, 1996.

[Clark87]
Negation as failure, K. Clark. Readings in nonmonotonic reasoning, Morgan Kaufmann Publishers, pages 311 - 325, 1987. (Originally published in 1978.)

[CK95]
Sorted HiLog: Sorts in Higher-Order Logic Data Languages, W. Chen, M. Kifer. Sixth Intl. Conference on Database Theory, Prague, Czech Republic, January 1995, Lecture Notes in Computer Science 893, Springer Verlag, pp. 252--265.

[CKW93]
HiLog: A Foundation for higher-order logic programming, W. Chen, M. Kifer, D.S. Warren. Journal of Logic Programming, vol. 15, no. 3, February 1993, pp. 187--230.

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

[CURIE]
CURIE Syntax 1.0, S. McCarron, M. Birbeck, Editors, W3C Working Group Note, 16 December 2010, http://www.w3.org/TR/2010/NOTE-curie-20101216 . Latest version available at http://www.w3.org/TR/curie.

[CycL]
The Syntax of CycL, Web site. Available at http://www.cyc.com/cycdoc/ref/cycl-syntax.html.

[DAA]
Modularity in the Rule Interchange Format, Carlos Viegas Damásio, Anastasia Analyti, and Grigoris Antoniou. 5th International Symposium on Rule-Based Reasoning, Programming, and Applications (RuleML 2011 - Europe), Nick Bassiliades, Guido Governatori, and Adrian Paschke (Eds.). Springer-Verlag, Berlin, Heidelberg, 313-328. 2011.

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

[Fit02]
Fixpoint semantics for logic programming a survey, M. Fitting. Theoretical Computer Science, vol. 278, no. 1-2, pp. 25-51, 2002.

[FL2]
FLORA-2: An Object-Oriented Knowledge Base Language, M. Kifer. Web site. Available at http://flora.sourceforge.net.

[GL88]
The Stable Model Semantics for Logic Programming, M. Gelfond and V. Lifschitz. Logic Programming: Proceedings of the Fifth Conference and Symposium, pages 1070-1080, 1988.

[GL91]
Classical Negation in Logic Programs and Disjunctive Databases, M. Gelfond and V. Lifschitz. New Generation Computing 9, pages 365-386, 1991.

[GLe02]
Logic programming and knowledge representation - The A-Prolog perspective, M. Gelfond and N. Leone. Artificial Intelligence 138(1-2), pages 3-38, 2002.

[GRS91]
The Well-Founded Semantics for General Logic Programs, A. Van Gelder, K.A. Ross, J.S. Schlipf. Journal of ACM, 38:3, pages 620-650, 1991.

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

[Lloyd87]
Foundations of Logic Programming (Second Edition), J.W. Lloyd, Springer-Verlag, 1987.

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

[NxBRE]
.NET Business Rule Engine, Web site. Available at http://nxbre.wiki.sourceforge.net/.

[OOjD]
Object-Oriented jDREW, Web site. Available at http://www.jdrew.org/oojdrew/.

[Prz94]
Well-founded and stationary models of logic programs, T.C. Przymusinski. Annals of Mathematics and Artificial Intelligence 12 (1994), pp. 141-187.

[RDFSYN04]
RDF/XML Syntax Specification (Revised), Dave Beckett, Editor, W3C Recommendation, 10 February 2004, http://www.w3.org/TR/2004/REC-rdf-syntax-grammar-20040210/. Latest version available at http://www.w3.org/TR/rdf-syntax-grammar/.

[RF99]
A Tight, Practical Integration of Relations and Functions, H. Boley, Springer-Verlag, 1999.

[Shoham87]
Nonmonotonic logics: meaning and utility, Y. Shoham. Proc. 10th International Joint Conference on Artificial Intelligence, Morgan Kaufmann, pp. 388--393, 1987.

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

W3C Proposed Edited Recommendation

[SWSL-Rules]
Semantic Web Services Language (SWSL), S. Battle, A. Bernstein, H. Boley, B. Grosof, M. Gruninger, R. Hull, M. Kifer, D. Martin, S. McIlraith, D. McGuinness, J. Su, S. Tabet. W3C Member Submission, September 2005. Available at http://www.w3.org/Submission/SWSF-SWSL/.

[TRT03]
Object-Oriented RuleML: User-Level Roles, URI-Grounded Clauses, and Order-Sorted Terms, H. Boley. Springer LNCS 2876, Oct. 2003, pp. 1-16. Available at http://nparc.cisti-icist.nrc-cnrc.gc.ca/npsi/ctrl?action=rtdoc&an=5764336&article=19&fd=pdf.

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

[WSML-Rules]
Web Service Modeling Language (WSML), J. de Bruijn, D. Fensel, U. Keller, M. Kifer, H. Lausen, R. Krummenacher, A. Polleres, L. Predoiu. W3C Member Submission, June 2005. Available at http://www.w3.org/Submission/WSML/.

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 `FORMULA`s 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 version="1.0" encoding="UTF-8"?>

 <xs:schema
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:xml="http://www.w3.org/XML/1998/namespace"
  xmlns="http://www.w3.org/2007/rif#"
  targetNamespace="http://www.w3.org/2007/rif#"
  elementFormDefault="qualified"
  version="Id: FLDBaseline.xsd, v. 1.5, 2010-05-08, hboley/dhirtle">

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

  <xs:annotation>
    <xs:documentation>

    This is the 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).

  FORMULA       ::= IRIMETA? CONNECTIVE '(' FORMULA* ')' |
                    IRIMETA? QUANTIFIER '(' FORMULA ')' |
                    IRIMETA? 'Neg' FORMULA |
                    IRIMETA? 'Naf' FORMULA |
                    IRIMETA?  FORMULA '@' MODULEREF |
                    FORM
  FORM          ::= IRIMETA? (Var | ATOMIC |
                             'External' '(' ATOMIC LOCATOR? ')')
  ATOMIC        ::= Const | Atom | Equal | Member | Subclass | Frame
  Atom          ::= UNITERM
  UNITERM       ::= TERMULA '(' (TERMULA* | (Name '->' TERMULA)*) ')'
  Equal         ::= TERMULA '=' TERMULA
  Member        ::= TERMULA '#' TERMULA
  Subclass      ::= TERMULA '##' TERMULA
  Frame         ::= TERMULA '[' (TERMULA '->' TERMULA)* ']'
  TERMULA       ::= IRIMETA? CONNECTIVE '(' TERMULA* ')' |
                    IRIMETA? QUANTIFIER '(' TERMULA ')' |
                    IRIMETA? 'Neg' TERMULA |
                    IRIMETA? 'Naf' TERMULA |
                    IRIMETA? TERMULA '@' MODULEREF |
                    TERM
  TERM          ::= IRIMETA? (Var | EXPRIC | List |
                             'External' '(' EXPRIC LOCATOR? ')' |
                             AGGREGATE | NEWTERM)
  EXPRIC        ::= Const | Expr | Equal | Member | Subclass | Frame
  Expr          ::= UNITERM
  List          ::= 'List' '(' TERM* ')' | 'List' '(' TERM+ '|' TERM ')'
  AGGREGATE     ::= AGGRFUNC '{' Var ('[' Var+ ']')? '|' FORMULA '}'
  Const         ::= '"' UNICODESTRING '"^^' SYMSPACE | CONSTSHORT
  MODULEREF     ::= Var | Const | Expr
  CONNECTIVE    ::= 'And' | 'Or' | NEWCONNECTIVE
  QUANTIFIER    ::= ('Exists' | 'Forall' | NEWQUANTIFIER) Var*
  AGGRFUNC      ::= 'Min' | 'Max' | 'Sum' | 'Prod' | 'Avg' | 'Count' |
                    'Set' | 'Bag' | NEWAGGRFUNC
```

W3C Proposed Edited Recommendation

```
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
FORMULA        ::= IRIMETA? 'And' '(' FORMULA* ')' |
                   IRIMETA? 'Or' '(' FORMULA* ')' |
                   IRIMETA? 'NEWCONNECTIVE' '(' FORMULA* ')' |
                   IRIMETA? 'Exists' Var* '(' FORMULA ')' |
                   IRIMETA? 'Forall' Var* '(' FORMULA ')' |
                   IRIMETA? 'NEWQUANTIFIER' Var* '(' FORMULA ')' |
                   IRIMETA? 'Neg' FORMULA |
                   IRIMETA? 'Naf' FORMULA |
                   IRIMETA? 'Remote' '(' FORMULA MODULEREF ')'
                   FORM
  -->
  <xs:choice>
    <xs:element name="And" type="And-FORMULA.type"/>
    <xs:element name="Or" type="Or-FORMULA.type"/>
    <xs:element name="NEWCONNECTIVE" type="NEWCONNECTIVE-FORMULA.type"/>
    <xs:element name="Exists" type="Exists-FORMULA.type"/>
    <xs:element name="Forall" type="Forall-FORMULA.type"/>
    <xs:element name="NEWQUANTIFIER" type="NEWQUANTIFIER-FORMULA.type"/>
    <xs:element name="Neg" type="Neg-FORMULA.type"/>
    <xs:element name="Naf" type="Naf-FORMULA.type"/>
    <xs:element name="Remote" type="Remote-FORMULA.type"/>
    <xs:group ref="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="declare" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element ref="formula"/>
  </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 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:choice>
    <xs:element ref="Var"/>
    <xs:group ref="ATOMIC"/>
    <xs:element name="External" type="External-FORM.type"/>
  </xs:choice>
</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:attribute name="ordered" type="xs:string" fixed="yes"/>
  </xs:complexType>
</xs:element>

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

```
    </xs:sequence>
    <xs:attribute name="ordered" type="xs:string" fixed="yes"/>
  </xs:complexType>

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

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

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

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

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

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

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

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

  <xs:element name="super">
    <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: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* ')' |
                   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:group ref="TERM"/>
  </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:element name="termula" minOccurs="0" maxOccurs="unbounded"/>
  </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: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:sequence>
        <xs:group ref="TERMULA"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <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 name="External" type="External-TERM.type"/>
      <xs:element ref="AGGREGATE"/>
      <xs:element ref="NEWTERM"/>
```

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

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

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

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

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

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

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

  <xs:group name="EXPRIC">
    <!--
  EXPRIC         ::= Const | Expr | Equal | Member | Subclass | Frame
    -->
    <xs:choice>
      <xs:element ref="Const"/>
      <xs:element ref="Expr"/>
      <xs:element ref="Equal"/>
      <xs:element ref="Member"/>
      <xs:element ref="Subclass"/>
      <xs:element ref="Frame"/>
    </xs:choice>
  </xs:group>

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

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

W3C Proposed Edited Recommendation

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

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

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

  <xs:complexType name="IRICONST.type" mixed="true">
    <!-- sensitive to location/id context-->
    <xs:sequence/>
    <xs:attribute name="type" type="xs:anyURI" use="required" fixed="http://www.w3.org/2007/rif#iri"/>
  </xs:complexType>

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

 </xs:schema>
```

8.2 Skyline Schema Module

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

 <xs:schema
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:xml="http://www.w3.org/XML/1998/namespace"
  xmlns="http://www.w3.org/2007/rif#"
  targetNamespace="http://www.w3.org/2007/rif#"
  elementFormDefault="qualified"
  version="Id: FLDSkyline.xsd, v. 1.5, 2010-02-02, hboley/dhirtle">

  <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):

  Document       ::= IRIMETA? 'Document' '(' Dialect? Base? Prefix* Import* Module* Group? ')'
  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">
```

```
   <!--
  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
   -->
   <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:redefine>

  <xs:element name="Document">
   <!--
  Document       ::= IRIMETA? 'Document' '(' Dialect? Base? Prefix* Import* Module* Group? ')'
  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   ::= ANGLEBRACKIRI
  PROFILE   ::= ANGLEBRACKIRI
   -->
```

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

  <xs:element name="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>
```

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 [GL88,GL91,GLe02] 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, $\boldsymbol{I}$ = <***TV***, ***DTS***, ***D***, $\boldsymbol{I}_C$, $\boldsymbol{I}_V$, $\boldsymbol{I}_F$, $\boldsymbol{I}_{NF}$, $\boldsymbol{I}_{list}$, $\boldsymbol{I}_{tail}$, $\boldsymbol{I}_{frame}$, $\boldsymbol{I}_{sub}$, $\boldsymbol{I}_{isa}$, $\boldsymbol{I}_{=}$, $\boldsymbol{I}_{external}$, $\boldsymbol{I}_{connective}$, $\boldsymbol{I}_{truth}$> is *Herbrand* if its domain, ***D***, is Herbrand and the mappings $\boldsymbol{I}_C$, $\boldsymbol{I}_V$, $\boldsymbol{I}_F$, $\boldsymbol{I}_{NF}$, $\boldsymbol{I}_{list}$, $\boldsymbol{I}_{tail}$, $\boldsymbol{I}_{frame}$, $\boldsymbol{I}_{sub}$, $\boldsymbol{I}_{isa}$, $\boldsymbol{I}_{=}$, $\boldsymbol{I}_{external}$, $\boldsymbol{I}_{connective}$, $\boldsymbol{I}_{truth}$ 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 ***HU***/ *E*, i.e., the set of all equivalence classes, of the elements in ***HU*** with respect to an equivalence relation, *E*, which is defined as the minimal relation that satisfies the following condition:

- If *s,t* ∈ ***HU*** and $TVal_I$(s=t) = **t** then (s,t) ∈ *E*.

We will use the symbol ***HD*** to denote Herbrand domains. ☐

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

- If *s,t* ∈ ***HU*** are terms with named arguments that differ only in the order of the arguments then (s,t) ∈ *E*.
- If *s,t* ∈ ***HU*** are frame terms that differ only in the order of their attribute/value pairs then (s,t) ∈ *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 $\boldsymbol{HD}_{sg}$ ⊆ ***HD*** is precisely the set {***I***(t)| t is 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 $\boldsymbol{\hat{I}}$ that is a model of **Γ** is said to be a ***minimal model in the truth order*** $<_t$ of **Γ** if there is no other Herbrand RIF-FLD model $\boldsymbol{\hat{I}'}$ of **Γ** such that:

- $TVal_{\hat{I}'}(\varphi)$ = **t** implies $TVal_{\hat{I}}(\varphi)$ = **t** and
- $TVal_{\hat{I}}(\varphi)$ = **f** implies $TVal_{\hat{I}'}(\varphi)$ = **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 iff $\boldsymbol{I}_{true}$ 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** $<_t$ **u** $<_t$ **t**. The usual general definition for **u** is that it is the smallest element in ***TV*** with respect to the *knowledge order* $<_k$ -- an order on the sets of truth values, which is sometimes used in addition to the truth order [Fit02]. In many cases, however, the mention of $<_k$ is omitted as, for example, in the case of the well-founded semantics (where it is implicitly assumed that **u** $<_k$ **f** and **u** $<_k$ **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** $<_t$ **u** $<_t$ **t**. Then ***I*** is said to be ***empty*** iff $\boldsymbol{I}_{true}$ 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(*directive₁* ... *directiveₙ* Γ) or Document(*directive₁* ... *directiveₙ*). 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 $?V_1$ ... $?V_n$(η), where η has no quantifiers and all of its variables are among $?V_1$ ... $?V_n$. A group formula Group(φ_1 ... φ_n) is ***universal*** if either n=0 (i.e., it is an empty group formula) or each φ_i is universal. ☐

Ground instantiations are now defined as follows.

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

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

Note that according to the definition of the [truth valuation](), $TVal_I(\varphi) = \sim TVal_I(\texttt{Naf}\ \varphi)$, i.e., when φ is true then `Naf φ` 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 `Neg φ`. 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 $TVal_I(\varphi)$ and $TVal_I(\texttt{Neg}\ \varphi)$ are not both **t** (which is equivalent to saying that $\boldsymbol{I}_{\text{true}}(\boldsymbol{I}(\varphi))$ and $\boldsymbol{I}_{\text{true}}(\boldsymbol{I}(\texttt{Neg}\ \varphi))$ are not both **t**). However, they can both be **f** (or, for example, **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 requerement for ~ 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]() and [here]()).