W3C Editor's Draft

RIF Core Dialect

W3C Editor's Draft 5 June 2009

This version:
http://www.w3.org/2005/rules/wg/draft/ED-rif-core-20090605/
Latest editor's draft:
http://www.w3.org/2005/rules/wg/draft/rif-core/
Editors:
Harold Boley, National Research Council Canada
Gary Hallmark, Oracle Corporation
Michael Kifer, State University of New York at Stony Brook, USA
Adrian Paschke, Freie Universitaet Berlin
Axel Polleres, DERI
Dave Reynolds, Hewlett-Packard Laboratories, Bristol UK

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

Abstract

This document, developed by the Rule Interchange Format (RIF) Working Group, specifies RIF-Core, a common subset of RIF-BLD and RIF-PRD based on RIF-DTB 1.0. The RIF-Core presentation syntax and semantics are specified by restriction in two different ways. First, RIF-Core is specified by restricting the syntax and semantics of RIF-BLD, and second, by restricting RIF-PRD. The XML serialization syntax of RIF-Core is specified by a mapping from the presentation syntax. A normative XML schema is also provided.

Status of this Document

May Be Superseded

This section describes the status of this document at the time of its publication. Other documents may supersede this document. A list of current W3C publications

W3C Editor's Draft

and the latest revision of this technical report can be found in the W3C technical reports index at http://www.w3.org/TR/.

Summary of Changes

@@TDB

Last Call

The Working Group believes it has completed its design work for the technologies specified this document, so this is a "Last Call" draft. The design is not expected to change significantly, going forward, and now is the key time for external review, before the implementation phase.

Please Comment By 3 July 2009

The Rule Interchange Format (RIF) Working Group seeks public feedback on this Working Draft. Please send your comments to public-rif-comments@w3.org (public archive). If possible, please offer specific changes to the text that would address your concern. You may also wish to check the Wiki Version of this document and see if the relevant text has already been updated.

No Endorsement

Publication as a Working Draft does not imply endorsement by the W3C Membership. This is a draft document and may be updated, replaced or obsoleted by other documents at any time. It is inappropriate to cite this document as other than work in progress.

Patents

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

Table of Contents

W3C Editor's Draft

1 Overview

This specification describes ***RIF-Core*** (the Core dialect of the Rule Interchange Format). From a theoretical perspective, RIF-Core corresponds to the language of definite Horn rules without function symbols (often called 'Datalog') with a standard first-order semantics. RIF-Core thus is a subset of RIF-BLD [RIF-BLD]. At the same time, RIF-Core is a language of production rules where conclusions are interpreted as assert actions. RIF-Core thus also is a subset of RIF-PRD [RIF-PRD]. Moreover, RIF-Core is based on the built-ins of RIF-DTB 1.0 [RIF-DTB]. The common subset of RIF-BLD and RIF-PRD is specified based on RIF-DTB 1.0.

Syntactically, RIF-Core has a number of Datalog extensions to support features such as objects and frames as in F-logic [KLW95], internationalized resource identifiers (or IRIs, defined by [RFC-3987]) as identifiers for concepts, and XML Schema datatypes [XML-SCHEMA2]. In addition, RIF RDF and OWL Compatibility [RIF-RDF+OWL] defines the syntax and semantics of integrated RIF-Core/RDF and RIF-Core/OWL languages. These features make RIF-Core a Web-aware language. However, it should be kept in mind that RIF is designed to enable

interoperability among rule languages in general, and its uses are not limited to the Web.

RIF-Core is defined as a specialization of RIF-BLD (hence of [RIF-FLD], making it a starting point of the RIF extensibility framework). It is a syntactic subset of RIF-BLD, so that a well-formed RIF-Core formula (including document and condition formulas) is also a well-formed RIF-BLD formula.

RIF-Core is also a syntactic subset of [RIF-PRD]. It is intended that a RIF-PRD consumer can treat a RIF-Core document as if it was a RIF-PRD rule set while it also conforms to the normative RIF-Core first order semantics. However, due to the presence of builtin functions and predicates there are rule sets in the syntactic intersection of RIF-PRD and RIF-BLD which would not terminate under RIF-PRD semantics. We therefore define a notion of safe RIF-Core rules, which is a subset of RIF-Core rules that can be executed using a forward chaining strategy, and we define conformance in terms of such safe rules. These notions of safeness and conformance are defined formally in section 5 Conformance and Safeness.

RIF-Core is not the *maximal* common subset of RIF-BLD and RIF-PRD. It omits some features from the intersection which do not significantly add to the expressiveness of the language and are judged to be not widely supported by rule languages.

To give a preview, here is a simple complete RIF-Core example deriving a ternary relation from its inverse.

Example 1 (An introductory RIF-Core example).

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

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

The fact *Mary buys LeRif from John* can be logically derived by a *modus ponens* argument. Assuming Web IRIs for the predicates `buy` and `sell`, as well as for the individuals `John`, `Mary`, and `LeRif`, the above English phrase can be represented in RIF-Core Presentation Syntax as follows.

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

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

```
        )

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

For the interchange of documents containing such rules (and facts), an equivalent RIF-Core XML syntax is provided in this specification. To formalize their meaning, a RIF-Core Semantics is specified.

This document assumes familiarity with [RIF-BLD] or [RIF-PRD], as RIF-Core is derived from these documents via syntactic restrictions.

2 RIF-Core Presentation Syntax

Like RIF-BLD and RIF-PRD, RIF-Core has both a ***presentation syntax*** and an ***XML syntax***. It is defined in "mathematical English," a special form of English for communicating mathematical definitions, examples, etc. and by an EBNF syntax. The mathematical English is normative, the EBNF is not normative; both instances of the presentation syntax are not intended to be a concrete syntax for RIF-Core. The English presentation syntax deliberately leaves out details such as the delimiters of the various syntactic components, escape symbols, parenthesizing, precedence of operators, and the like. Since RIF is an interchange format, it uses XML, and only XML, as its concrete syntax. RIF-Core conformance is described in terms of semantics-preserving mappings.

Since RIF-Core is a syntactic subset of RIF-BLD, this section defines the presentation syntax of RIF-Core as a restriction on the presentation syntax of RIF-BLD.

2.1 Alphabet of RIF-Core

The ***alphabet*** of the presentation language of RIF-Core is the alphabet of the RIF-BLD presentation language with the exclusion of the symbol `##` (subclass) and the set of symbols `ArgNames` (used for named-argument uniterms).

2.2 Terms of RIF-Core

The ***Terms*** of RIF-Core are the terms of RIF-BLD with the exclusion of *subclass terms* and of *terms with named arguments*. In Core there are only *closed ground* lists.

Definition (List Term)

- A ***closed ground list*** has the form `List(`t_1 `...` t_m`)`, where $m \geq 0$ and $t_1, \ldots, t_m$ are ground terms (no tail and no variables are allowed).

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

2.3 Formulas of RIF-Core

The ***Formulas*** of RIF-Core are the formulas of RIF-BLD with the following restrictions.

- Subterms that occur inside atomic formulas can be variables, constants, ground list, or external positional terms. This implies that RIF-Core only allows external function applications.
- Equality terms and class membership terms *cannot* occur in rule conclusions -- they are allowed only in rule premises.
- Terms with named arguments and subclass terms are excluded from RIF-Core.

2.4 Annotations and Documents

RIF-Core allows every term and formula to be optionally annotated in the same way as in RIF-BLD. The frame formulas that are allowed as part of an annotation must be syntactically correct for RIF-Core. In particular, no function symbols are allowed in such a formula.

2.5 Well-formed Formulas

A syntactically correct RIF-Core formula that passes the well-formedness test for RIF-BLD is also a well-formed RIF-Core formula.

Recall that RIF-Core does not allow uninterpreted (i.e., non-external) function symbols. Therefore no symbol in RIF-Core can occur in the context of an (uninterpreted) function symbol.

2.6 EBNF Grammar for the Presentation Syntax of RIF-Core

Until now, we have used mathematical English to specify the syntax of RIF-Core as a restriction on RIF-BLD. Tool developers, however, may prefer EBNF notation, which provides a more succinct view of the syntax. However, EBNF is unable to express all of the well-formedness conditions. For instance, the requirement that each symbol appear in only one context cannot be expressed in EBNF. As a result, the EBNF grammar defines a strict superset of RIF-Core. For that reason this section is *not normative*.

The EBNF for the RIF-Core presentation syntax is given as follows. For convenience of reading we show the entire EBNF divided into three parts (rules,

conditions, and annotations); these are derived from the ENBF for RIF-BLD by applying the restrictions described above.

Rule Language:

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

Condition Language:

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

Annotations:

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

`ANGLEBRACKIRI`, `CURIE`, `CONSTSHORT`, and `UNICODESTRING` are defined in Section Shortcuts for Constants in RIF's Presentation Syntax of [RIF-DTB].

The following subsections explain and exemplify the Condition Language, Rule Language, and Annotations parts.

2.6.1 EBNF for the RIF-Core Condition Language

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

This is a specialization of the EBNF for the RIF-BLD condition language specified in the RIF-BLD conditions part reflecting the syntax restrictions on RIF-Core described normatively in sections 2.1 through 2.5 above.

Example 3 from the RIF-BLD document, illustrates some RIF-BLD conditions. All the conditions, except for the terms with named arguments and the equalities with (non-ground) list terms, are also RIF-Core conditions.

2.6.2 EBNF for the RIF-Core Rule Language

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

Again, this is a specialization of the EBNF for the RIF-BLD rule language specified in the RIF-BLD rules part reflecting the syntax restrictions on RIF-Core described normatively in sections 2.1 through 2.5 above.

Example 4 from the RIF-BLD document also illustrates a set of RIF-Core rules. In contrast, Example 7 from the RIF-BLD document shows a formula that is *not* in RIF-Core because it includes terms with named arguments, which are not allowed in this dialect.

2.6.3 EBNF for RIF-Core Annotations

The presentation syntax for RIF-Core annotations uses the production shown in the above annotations part.

This defines the specialization of the EBNF for the RIF-BLD annotation language specified through the RIF-BLD annotations part where annotation frames use the more restricted TERMs defined in the above conditions part of RIF-Core.

Example 5 from the RIF-BLD document also illustrates a RIF-Core document that contains an annotated group formula.

3 RIF-Core as a Specialization of RIF-PRD

RIF-Core is a syntactic subset of RIF-PRD, and this section defines the presentation syntax of RIF-Core as a restriction on the presentation syntax of RIF-PRD Conditions, Actions, and Rules.

3.1 Alphabet of RIF-Core

The ***alphabet*** of the presentation language of RIF-Core is the alphabet of the RIF-PRD presentation language (Conditions, Actions, and Rules) with the exclusion of the symbols `##`, `such that`, `Not`, `INeg`, `Do`, `Assert`, `Retract`, `Modify`, `Execute`, and `New`.

3.2 Terms of RIF-Core

The ***Terms*** of RIF-Core are the terms of RIF-PRD with the exclusion of *subclass terms*. In Core there are only *closed ground* lists.

3.3 Formulas of RIF-Core

The ***Formulas*** of RIF-Core are the formulas of RIF-PRD with the exclusion of *negation formulas*.

3.4 Annotations and Documents

RIF-Core allows every term and formula to be optionally annotated in the same way as in RIF-PRD. The frame formulas that are allowed as part of an annotation must be syntactically correct for RIF-Core.

3.5 Well-formed Formulas

A syntactically correct RIF-Core formula that passes the well-formedness test for RIF-PRD is also a well-formed RIF-Core formula.

3.6 Rules and Groups

A RIF-Core rule is a well-formed RIF-PRD rule rule with no nested forall, no binding pattern, and where the action block is a single atom, a single frame, or a conjunction of atoms and/or frames. A RIF-Core group is a RIF-PRD group without `strategy` and without `priority`.

4 RIF-Core Semantics

RIF-Core is a syntactic subset of RIF-BLD, and the semantics of RIF-Core is identical to the semantics of RIF-BLD for that subset. RIF-Core is also a syntactic subset of RIF-PRD, and the semantics of RIF-Core is also identical to the semantics of RIF-PRD for that subset.

5 XML Serialization Syntax for RIF-Core

The XML syntax of RIF-Core is a subset of the XML syntax of RIF-BLD. All XML tags of RIF-BLD (except `Subclass`, `sub` and `super`) are supported, but the XML schema of RIF-Core restricts their context with respect to what is allowed by the XML schema of RIF-BLD. The semantics of the XML syntax for RIF-Core is defined through the same RIF-BLD XML-to-presentation syntax mapping.

XML serialization of a complete RIF-Core document appears in the RIF-BLD specification as Example 8.

6 Safeness Criteria

RIF-Core is a syntactic subset of both RIF-BLD and RIF-PRD. The semantics of a RIF-Core formula is the same as the semantics given to it by RIF-BLD.

All RIF-Core documents are also syntactically valid RIF-PRD documents. However, some formulas may be *unsafe* and cannot be executed under the RIF-PRD operational semantics. Thus, in order to allow production rule systems and logic programming systems to interchange rules via RIF-Core, we restrict RIF-Core to *safe* rules so that the logical semantics of RIF-BLD and the operational fixed-point semantics of RIF-PRD coincide.

6.1 Safeness

Intuitively, safeness of rules guarantees that when performing reasoning in a forward-chaining manner, it is possible to find bindings for all the variables in the rule so that the condition can be evaluated.

To define safeness in the face of external predicates and functions, we define the notion of ***binding patterns***, which are lists of the form (p_1, . . . , p_n), such that p_i=`b` or p_i=`u`, for $1 \leq i \leq n$. Intuitively, `b` stands for a "bound" and `u` stands for an "unbound" argument.

Each external function or predicate has an associated list of ***valid binding patterns***. We define here the binding patterns valid for the functions and predicates defined in [DTB].

Every function or predicate `f` defined in [DTB] has a valid binding pattern for each of its schemas with only the symbol `b` such that its length is the number of arguments in the schema. In addition,

- the external predicate `pred:iri-string` has the valid binding patterns (`b`, `u`) and (`u`, `b`) and
- the external predicate `pred:list-contains` has the valid binding pattern (`b`, `u`).

The functions and predicates defined in [DTB] have no other valid binding patterns.

For dealing with disjunction, existential quantifiers, and equality in rule premises we need a number of preliminary definitions before we can define safeness.

Let ψ be a condition formula. With ψ' we denote the formula obtained from ψ by replacing all subformulas of the form `Exists ?V1 ... ?Vn(φ)` with φ', which is obtained from φ by replacing all occurrences of `?V1, ..., ?Vn` with `?V1', ..., ?Vn'`, which are variable symbols not appearing anywhere outside of φ.

The *tree corresponding to* ψ', denoted $T_{\psi'}$ is a labelled tree (N,E,L), where each node n ∈ N is labelled (using the labelled function L) with a set of formulas and is constructed using the following rules. We define $T_{\psi'}$ as the smallest labelled tree such that n_0 is the root, $\psi' \in L(n_0)$, and for every node n ∈ N we have that

- if some conjunction `And(φ1 ... φk)` ∈ L(n), then $\varphi_i \in L(n)$ for every φ_i, $1 \leq i \leq k$ and
- if $\varphi_1, \ldots, \varphi_h$ is the set of disjunctions in L(n) and φ_1=`Or(ψ1 ... ψm)`, L(n) has `m` child nodes $n'_1,\ldots,n'_m$ such that $L(n'_j)=L(n)\backslash\varphi_1\cup\{\psi_j\}$, for $1 \leq j \leq m$.

With B_ψ we denote the collection of sets of atomic formulas in the leaf nodes of $T_{\psi'}$, i.e., $A \in B_\psi$ iff A is L(n), restricted to atomic formulas, for some leaf node n of the $T_{\psi'}$.

Given a set of atomic formulas A, the *equivalence class* $E_{?V}$ of a variable `?V` in A is the smallest set such that $?V \in E_{?V}$ and for every $?V' \in E_{?V}$ and every variable `?V''`, if `?V'=?V''` ∈ A or `?V''=?V'` ∈ A, $?V'' \in E_{?V}$.

Definition (Safeness). Let ψ be a condition formula. A variable `?V` is ***safe*** in ψ if, for every $A \in B_\psi$, a variable `?V'` in the equivalence class of `?V` in A appears in an atomic formula in A that is not an equality term involving two variables.

Given an A ∈ B_ψ. Every constant symbol `c` is ***bounded*** in A. A variable `?V` is bounded in A if whenever `?V` appears in A, a variable `?V'` in the equivalence class of `?V` in A is an argument of a non-external atomic formula in A that is not an equality term involving two variables or `?V'` is the `i`th argument of an external atomic formula `External(f(`t_1`,...,`t_n`))` such that `f` has a valid binding pattern (p_1, ..., p_n) such that p_i=`u` and, for every `j` ∈ `{1,...,n}\i`, p_j=`b` iff t_j is bounded in A. Finally, an external term `External(f(`t_1`,...,`t_n`))` is bounded in A if `f` has a valid binding pattern (p_1, ..., p_n) such that p_j=`b` iff t_j is bounded in A, for `1 ≤ j ≤ n`.

A variable `?V` is bounded in ψ if `?V` is bounded in every A ∈ B_ψ.

A document formula `Γ` is ***safe*** if every group formula in `Γ` is safe. A group formula `Group(`φ_1 `...` φ_n`)` is safe if φ_1, ..., and φ_n are safe. Every variable-free atomic formula is safe. List terms are safe by definition, because they are always ground in Core, i.e. contain no variables. A universal fact `Forall ?V1 ... ?Vn (`φ`)` is safe iff φ is a variable-free atomic formula. A universal rule `Forall ?V1 ... ?Vn (`φ ` :- `ψ`)` is safe iff φ ` :- ` ψ is safe. A rule implication φ ` :- ` ψ is safe iff all variables appearing in φ are safe in ψ and all variables are bounded in ψ. □

Consider the following formula:

```
Forall ?x ?y ?z ?u (ex:p(?x) :- Or(
  And( ex:q(?z) External(pred:iri-string(?x ?z))))
  And( ?x=?y ?y=?u ex:q(?u)))
```

One can verify that this formula is safe, in the following way: the only variable appearing in the conclusion of the rule is `?x`; `?x` is safe in the first component of the disjunction, because it appears in the atomic formula `pred:iri-string(?x,?z)`. We have that `(u,b)` is a valid binding pattern for `pred:iri-string`, and `?z` is bound by virtue of `ex:q(?z)` and so `?x` is bounded as well. We then conclude that `?x` is also safe in the second component, because `?u` is in the equivalent class of `?x` and appears in `ex:q(?u)`. Then, clearly the variables `?z` and `?u` are bounded.

6.2 Strong Safeness (Informative)

While safeness guarantees the possibility to do forward chaining with the rules, it does not guarantee that it is possible to construct a finite grounding. For this purpose we define strong safeness.

The conformance clauses for RIF-Core only require conformance over safe rule sets as defined above. However, some rule engines, such as some Datalog engines, are only able to process rule sets which can be finitely grounded. For maximum interoperability with such systems it is recommended that RIF-Core producers restrict themselves to strongly safe rule sets where possible.

Let R be a set of safe rule implications φ :- ψ and let P be the set of pairs (p,n), where p is a predicate symbol and n is a nonnegative integer (an arity). For the purposes of the definitions in this section we view frames a[b -> c] and membership formulas a#b, respectively, as ternary and binary predicate symbols, and so (->,3), (#,2) $\in$ P. Note that equality = does not appear in P.

We define the *graph of variable dependencies* of a set of atomic formulas A as a labeled directed graph G_R=(V, E, L), where the labeling function L maps edges to sets of external function and predicate symbols, V is the set of variables appearing in A, and E is the smallest set and L' is the smallest function such that for every variable ?V

- for every atomic formula ?V=t or t=?V in A and every variable ?V'$\neq$?V appearing in t such that $f_1, \ldots, f_n$, $0 \leq n$, are the function symbols of the terms in t (including t itself) in which ?V' appears, (?V,?V') $\in$ E and $\{f_1, \ldots, f_n\} \in$ L'((?V,?V')), and
- for every external atomic formula External(f(t_1,...,t_n)) in A, every $i \in \{1,\ldots,n\}$ such that t_i=?V, every valid binding pattern ($p_1, \ldots, p_n$) of f such that p_i=u, and every variable ?V' appearing in some t_j such that p_i=b and $f_1, \ldots, f_m$, $0 \leq m$, are the function symbols of the terms in t_j in which ?V' appears, (?V,?V') $\in$ E and $\{f_1, \ldots, f_m\} \in$ L'((?V,?V')).

Finally, L is defined as: for every (e,e') $\in$ E, L((e,e')) is the union of the minimal sets in L'((e,e')).

The *dependency graph* of a set of implications R is a labelled directed graph G_R=(V, E), where edges are triples (v,v',l) such that v, v' $\in$ V and l is a set of external function and predicate symbols. V is defined as: for every (p,n) $\in$ P and every integer i such that $1 \leq i \leq n$, (p,n)/i $\in$ V. E is the smallest set such that for every (p,n)/i $\in$ V and every φ :- ψ in R such that there is an atomic subformula p(t_1,...,t_i,...,t_n) of φ, then for every variable ?V appearing in t_i:

- for every non-external and non-equality atomic formula with predicate symbol p' and m arguments in any A $\in B_\psi$ and every $j \in \{1,\ldots,m\}$ such that a variable ?V' is the jth argument and there is a path from ?V to ?V' in the graph of variable dependencies of A and F is the union of the labels of the shortest path, ((p,n)/i,(p',m)/j, F $\cup$ {f_1,...,f_l}) $\in$ E, where $f_1, \ldots, f_l$, $0 \leq l$, are the function symbols of the terms in t_i in which ?V appears.

Definition (Strong safeness). A set of rule implications R is ***strongly safe*** if its dependency graph does not contain cycles involving edges labelled with sets involving a function defined in [RIF-DTB] that is not a casting function. A RIF document *R* is strongly safe if the set of rule implications that are subformulas of *R* is strongly safe.

Editor's Note: We might want to have a restricted set of function symbols to check, because possibly not every external function generates new values.

7 Conformance Clauses

RIF-Core conformance is described in terms of semantics-preserving transformations.

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 set of external predicates and functions that includes the built-ins listed in [RIF-DTB]. We say that a formula φ is a *Core*$_{T,E}$ formula iff

- φ is a well-formed Core formula,
- all the datatypes and symbol spaces used in φ are in T, and
- all the externally defined functions and predicates used in φ are in E.

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

A RIF processor is a ***conformant*** *Core*$_{T,E}$ ***producer*** iff it implements a *semantics-preserving mapping* from the language *L* of the processor to a set of *safe Core*$_{T,E}$ formulas.

An ***admissible document*** is an XML document that conforms to all the syntactic constraints of RIF-Core, including ones that cannot be checked by an XML Schema validator. Note that the concrete presentation syntax given in Section 2.6 is purely informative (to help implementers see the set of language structures supported by RIF-Core); the only normative concrete syntax for RIF-Core is the XML syntax.

In addition:

- Conformant BLD producers and consumers are required to support only the entailments of the form φ $|=_{CORE}$ ψ, where ψ is a *closed RIF-Core condition formulas*, that is a RIF-Core condition formula which also meets the criteria for closed condition formula defined in RIF-BLD.
- A ***conformant Core consumer*** is a conformant Core$_{T,E}$ consumer in which T consists only of the symbol spaces and datatypes, and E consists only of the externally defined functions and predicates that are required by RIF-Core. The required symbol spaces are `rif:iri` and `rif:local`, and the datatypes and externally defined terms (built-ins) are the ones specified in [RIF-DTB]. A conformant RIF-Core consumer must reject all inputs that do not match the syntax of safe Core formulas. If it implements

extensions, it may do so under user control -- having a "strict Core" mode and a "run-with-extensions" mode.
- A ***conformant Core producer*** is a conformant Core$_{T,E}$ producer which produces documents that include only the symbol spaces, datatypes, and externals that are required by Core.

Feature At Risk #3: Strictness Requirement

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

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

8 Acknowledgements

This document is the product of the Rules Interchange Format (RIF) Working Group (see below) whose members deserve recognition for their time and commitment. The editors extend special thanks to Jos de Bruijn for his safeness definition and to: Jos de Bruijn, Leora Morgenstern, Christian de Sainte-Marie, Stella Mitchell and Changhai Ke for their thorough reviews and insightful discussions; the working group chairs, Chris Welty and Christian de Sainte-Marie, for their invaluable technical help and inspirational leadership; and W3C staff contact Sandro Hawke, a constant source of ideas, help, and feedback.

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

9 References

9.1 Normative References

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

[RFC-3066]
RFC 3066 - Tags for the Identification of Languages, H. Alvestrand, IETF, January 2001. This document is http://www.isi.edu/in-notes/rfc3066.txt.

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

[RIF-BLD]
RIF Basic Logic Dialect, Boley H. and Kifer M. (Editors), W3C Rule Interchange Format Working Group Draft. Latest Version available at http://www.w3.org/2005/rules/wiki/BLD.

[RIF-DTB]
RIF Datatypes and Built-Ins 1.0, Polleres A., Boley H. and Kifer M. (Editors), W3C Rule Interchange Format Working Group Draft. Latest Version available at http://www.w3.org/2005/rules/wiki/DTB.

[RIF-FLD]
RIF Framework for Logic Dialects, Boley H. and Kifer M. (Editors), W3C Rule Interchange Format Working Group Draft. Latest Version available at http://www.w3.org/2005/rules/wiki/FLD.

[RIF-RDF+OWL]
RIF RDF and OWL Compatibility, de Bruijn, J. (Editor), W3C Rule Interchange Format Working Group Draft. Latest Version available at http://www.w3.org/2005/rules/wiki/SWC.

[RIF-PRD]
RIF Production Rule Dialect, de Sainte Marie C., Paschke A., Hallmark G. (Editors), W3C Rule Interchange Format Working Group Draft. Latest Version available at http://www.w3.org/2005/rules/wiki/PRD.

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

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

[XML-SCHEMA2]
XML Schema Part 2: Datatypes, W3C Recommendation, World Wide Web Consortium, 2 May 2001. This version is http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/. The latest version is available at http://www.w3.org/TR/xmlschema-2/.

9.2 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.

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

[CURIE]
CURIE Syntax 1.0: A syntax for expressing Compact URIs, Mark Birbeck, Shane McCarron. W3C Working Draft 2 April 2008. Available at http://www.w3.org/TR/curie/.

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

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

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

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

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

[RIF-UCR]
RIF Use Cases and Requirements, Paschke A., Hirtle D., Ginsberg A., Patranjan P-L., McCabe F. (Editors), W3C Rule Interchange Format Working Group Draft. Latest Version available at http://www.w3.org/2005/rules/wiki/UCR.

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

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

10 Appendix: XML Schema for RIF-Core

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

XML schemas for the RIF-Core sublanguages are defined below and are also available here with additional examples.

10.1 Condition Language

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

<xs:schema
 xmlns:xs="http://www.w3.org/2001/XMLSchema"
 xmlns="http://www.w3.org/2007/rif#"
 targetNamespace="http://www.w3.org/2007/rif#"
 elementFormDefault="qualified"
 version="Id: CoreCond.xsd, v. 0.8, 2009-04-29, hboley">

 <xs:annotation>
   <xs:documentation>
   This is the XML schema for the Condition Language as defined by
   the RIF Core Dialect.

   The schema is based on the following EBNF for the RIF-Core Condition La
   (prepared for generalization to the RIF-BLD and RIF-PRD Condition Langu

 FORMULA        ::= IRIMETA? 'And' '(' FORMULA* ')' |
                    IRIMETA? 'Or' '(' FORMULA* ')' |
                    IRIMETA? 'Exists' Var+ '(' FORMULA ')' |
                    ATOMIC |
                    IRIMETA? Equal |
                    IRIMETA? Member |
                    IRIMETA? 'External' '(' Atom ')'
 ATOMIC         ::= IRIMETA? (Atom | Frame)
 Atom           ::= UNITERM
 UNITERM        ::= Const '(' (TERM* ')'
 GROUNDUNITERM  ::= Const '(' (GROUNDTERM* ')'
 Equal          ::= TERM '=' TERM
 Member         ::= TERM '#' TERM
 Frame          ::= TERM '[' (TERM '->' TERM)* ']'
 TERM           ::= IRIMETA? (Const | Var | List | 'External' '(' Expr ')'
 GROUNDTERM     ::= IRIMETA? (Const | List | 'External' '(' 'Expr' '(' GRO
```

```
Expr            ::= UNITERM
List            ::= 'List' '(' GROUNDTERM* ')'
Const           ::= '"' UNICODESTRING '"^^' SYMSPACE | CONSTSHORT
Name            ::= UNICODESTRING
Var             ::= '?' UNICODESTRING
SYMSPACE        ::= ANGLEBRACKIRI | CURIE

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

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

<xs:group name="FORMULA">
  <!--
FORMULA         ::= IRIMETA? 'And' '(' FORMULA* ')' |
                    IRIMETA? 'Or' '(' FORMULA* ')' |
                    IRIMETA? 'Exists' Var+ '(' FORMULA ')' |
                    ATOMIC |
                    IRIMETA? Equal |
                    IRIMETA? Member |
                    IRIMETA? 'External' '(' Atom ')'
  -->
  <xs:choice>
    <xs:element ref="And"/>
    <xs:element ref="Or"/>
    <xs:element ref="Exists"/>
    <xs:group ref="ATOMIC"/>
    <xs:element ref="Equal"/>
    <xs:element ref="Member"/>
    <xs:element name="External" type="External-FORMULA.type"/>
  </xs:choice>
</xs:group>

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

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

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

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

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

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

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

<xs:group name="ATOMIC">
  <!--
ATOMIC         ::= IRIMETA? (Atom | Frame)
  -->
```

```
  <xs:choice>
    <xs:element ref="Atom"/>
    <xs:element ref="Frame"/>
  </xs:choice>
</xs:group>

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

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

<xs:group name="GROUNDUNITERM">
  <!-- sensitive to ground terms
GROUNDUNITERM        ::= Const '(' (GROUNDTERM* ')'
  -->
  <xs:sequence>
    <xs:group ref="IRIMETA" minOccurs="0" maxOccurs="1"/>
    <xs:element ref="op"/>
    <xs:element name="args" type="args-GROUNDUNITERM.type"/>
  </xs:sequence>
</xs:group>

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

<xs:complexType name="args-UNITERM.type">
```

```
  <!-- sensitive to UNITERM (TERM) context-->
  <xs:sequence>
    <xs:group ref="TERM" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute name="ordered" type="xs:string" fixed="yes"/>
</xs:complexType>

<xs:complexType name="args-GROUNDUNITERM.type">
  <!-- sensitive to GROUNDUNITERM (TERM) context-->
  <xs:sequence>
    <xs:group ref="GROUNDTERM" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute name="ordered" type="xs:string" fixed="yes"/>
</xs:complexType>


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

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

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

<xs:element name="Member">
  <!--
Member         ::= TERM '#' TERM
```

```
  -->
  <xs:complexType>
    <xs:sequence>
      <xs:group ref="IRIMETA" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="instance"/>
      <xs:element ref="class"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

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

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

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

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

<xs:complexType name="slot-Frame.type">
```

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



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

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


<xs:element name="List">
  <!--
    List           ::= 'List' '(' GROUNDTERM* ')'
  -->
  <xs:complexType>
    <xs:sequence>
      <xs:group ref="GROUNDTERM" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>


<xs:complexType name="External-TERM.type">
  <!-- sensitive to TERM (Expr) context-->
```

```
  <xs:sequence>
    <xs:group ref="IRIMETA" minOccurs="0" maxOccurs="1"/>
    <xs:element name="content" type="content-TERM.type"/>
  </xs:sequence>
</xs:complexType>

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


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

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

<xs:complexType name="content-GROUNDEXPR.type">
  <!-- sensitive to GROUNDEXPR context-->
  <xs:sequence>
    <xs:element name="GROUNDUNITERM"/>
  </xs:sequence>
</xs:complexType>



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

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

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

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

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

<xs:element name="id">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="Const" type="IRICONST.type"/>   <!-- type="&rif;i
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

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

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

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

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

</xs:schema>
```

10.2 Rule Language

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

<xs:schema
 xmlns:xs="http://www.w3.org/2001/XMLSchema"
 xmlns="http://www.w3.org/2007/rif#"
 targetNamespace="http://www.w3.org/2007/rif#"
 elementFormDefault="qualified"
 version="Id: CoreRule.xsd, v. 0.8, 2009-04-16, hboley">

 <xs:annotation>
   <xs:documentation>
   This is the XML schema for the Rule Language as defined by
   the RIF Core Dialect.

   The schema is based on the following EBNF for the RIF-Core Rule Languag
```

```
  (prepared for generalization to the RIF-BLD and RIF-PRD Rule Languages)

Document  ::= IRIMETA? 'Document' '(' Base? Prefix* Import* Group? ')'
Base      ::= 'Base' '(' IRI ')'
Prefix    ::= 'Prefix' '(' Name IRI ')'
Import    ::= IRIMETA? 'Import' '(' IRICONST PROFILE? ')'
Group     ::= IRIMETA? 'Group' '(' (RULE | Group)* ')'
RULE      ::= (IRIMETA? 'Forall' Var+ '(' CLAUSE ')') | CLAUSE
CLAUSE    ::= Implies | ATOMIC
Implies   ::= IRIMETA? (ATOMIC | 'And' '(' ATOMIC* ')') ':-' FORMULA
PROFILE   ::= TERM

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

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

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

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

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

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

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

  <xs:element name="location">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Const" type="IRICONST.type"/>   <!-- type="&rif;i
      </xs:sequence>
    </xs:complexType>
  </xs:element>

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

  <xs:element name="Group">
    <!--
  Group     ::= IRIMETA? 'Group' '(' (RULE | Group)* ')'
    -->
    <xs:complexType>
     <xs:sequence>
      <xs:group ref="Group.content"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

 <xs:group name="Group.content">
     <xs:sequence>
        <xs:group ref="IRIMETA" minOccurs="0" maxOccurs="1"/>
        <xs:element ref="sentence" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
```

```
</xs:group>


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

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

 <xs:element name="Forall">
   <xs:complexType>
     <xs:sequence>
       <xs:group ref="Forall.content"/>
     </xs:sequence>
   </xs:complexType>
 </xs:element>

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


 <xs:group name="CLAUSE">
   <!--
 CLAUSE   ::= Implies | ATOMIC
```

```
  -->
  <xs:choice>
    <xs:element ref="Implies"/>
    <xs:group ref="ATOMIC"/>
  </xs:choice>
</xs:group>

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

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

<xs:element name="then">
  <xs:complexType>
   <xs:sequence>
     <xs:group ref="then.content"/>
   </xs:sequence>
  </xs:complexType>
</xs:element>

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


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

```
    </xs:sequence>
  </xs:complexType>

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

</xs:schema>
```