A Guide to the Basic Logic Dialect for Rule Interchange on the Web
Harold Boley and Michael Kifer

Abstract—The W3C Rule Interchange Format (RIF) is a forthcoming standard for exchanging rules among different systems and for developing intelligent rule-based applications for the Semantic Web. The RIF architecture is conceived as a family of languages, called dialects. A RIF dialect is a rule-based language with an XML syntax and a well-defined semantics. The RIF Basic Logic Dialect (RIF-BLD) semantically corresponds to a Horn rule language with equality. RIF-BLD has a number of syntactic extensions with respect to traditional textbook Horn logic, which include F-logic frames and predicates with named arguments. RIF-BLD is also well-integrated with the relevant Web standards. It provides IRIs (Internationalized Resource Identifiers), XML Schema datatypes, and is aligned with RDF and OWL. This article is a guide to the essentials of RIF-BLD, its syntax, semantics, and XML serialization. At the same time, some important RIF-BLD features are omitted due to space limitations, including datatypes, built-ins, and the integration with RDF and OWL.

Index Terms—Logic Programming, Mathematical Logic, Horn Logic, F-logic, Equality, Deduction, IRI, XML, RIF.

I. INTRODUCTION

Rule languages and systems have been playing an important role in information technology and applications, and rule-based automated inference has been at the core of diverse technologies ranging from expert and decision-support systems to deductive databases and business rules. In addition, recent progress in business rules, policy rules, workflow rules, Web service rules, and event processing rules has led to renewed interest in the research and development of rule languages.

The Internet, the World Wide Web, and now the Semantic Web opened up new applications for rule-based technologies and presented new opportunities for growth. To enable an infrastructure that would allow interoperation among rule-based systems on the Web, standards are needed for publishing and interpreting rules. It is envisioned that the Semantic Web would evolve into a network of intelligent information sources, which will publish rules in a standardized format, so that semantic rule engines will be able to import and process such rules. Responding to these circumstances, in 2005 the World Wide Web Consortium (W3C) created the Rule Interchange Format (RIF) Working Group¹ and tasked it with designing the requisite standards. This set of standards is supposed not only to ensure interoperability among the currently available rule systems, but also (and primarily) the future ones. The standards, therefore, must be extensible in order to deal with the evolution of rule technologies. The expectation is that a standard rule interchange format would facilitate the exchange of rules among different systems as well as promote the creation of intelligent rule-based applications and environments for the Web.

In many respects, the job of the RIF Working Group was significantly harder than that of other W3C groups, even within the Semantic Web activity. For instance, the original OWL (version 1) Working Group [1] had to design just one language based on a body of already existing work on Description Logic. Although more than a dozen such logics were known at the time, they all shared much of the syntax and semantics. The situation with rule languages is quite different. Reaction rules, production rules, logic programming rules, and first-order rules share relatively little in terms of syntax and semantics and, even within each category of rule languages, there are multiple different syntaxes and semantics. The decision, therefore, was to develop a rule interchange format with an architecture that supports an extensible family of languages, called dialects.

RIF’s underlying idea is as follows. A RIF dialect is a rule-based language with an XML syntax and a well-defined semantics. A dialect $D_1$ can extend a dialect $D_2$ if the syntax of $D_1$ is a superset of the syntax of $D_2$, and the dialects are semantically compatible. Given such a family of RIF dialects, a rule system, $A$, can exchange its native rule set $R$ with a system $B$ if there is a RIF dialect, $D$, such that $A$ can map $R$ to a rule set $R^D_D$ in $D$ in a semantics-preserving manner, and $B$ can map the rules in $R^D_D$ to its native rule set $S$, again preserving the semantics. The key point here is that both the syntax and the semantics of a RIF dialect, such as $D$, are standardized, and that to be RIF-compliant the rule systems $A$ and $B$ must ‘implement’ one or more dialects (e.g., $D$). A rule system implements a dialect if the native language of the system is a syntactic variant of the dialect with possible extensions. So, if $A$ and $B$ implement the dialect $D$ then there must be semantics-preserving mappings from $D$ onto some subsets of the languages of $A$ and $B$, and back from those subsets to $D$. The existence of such mappings enables the interchange of rules between $A$ and $B$, if the rules fall into the aforesaid subsets of the two systems.

To ensure coherence among the various dialects and facilitate extensions, RIF’s goal is to define frameworks, i.e., general formalisms that can be specialized to particular dialects by ‘tweaking’ features. The RIF Working Group has been focusing on two families of dialects: logic dialects and production rule dialects. The logic-based family of dialects is intended to cover rule systems that are based on logic programming, deductive databases, and pure first-order logic. At present,

¹http://www.w3.org/2005/rules/wiki/RIF_Working_Group

H. Boley is with National Research Council, Canada. M. Kifer is with State University of New York at Stony Brook, USA
the working group has defined only one major dialect in this family: the Basic Logic Dialect, or RIF-BLD [2]. RIF-BLD is the main focus of this paper. The production rule family is intended to account for many commercial condition-action rule systems and, theoretically, could be extended to reaction (event-condition-action) rules. This family of RIF dialects is currently represented by the Production Rules Dialect, or RIF-PRD [3]. The RIF-Core dialect is designed to be shared between the logic and production rule families of dialects.

As for the RIF frameworks, only the framework for the logic-based family of dialects, called the RIF Framework for Logic Dialects [4] (RIF-FLD) has been worked out. RIF-FLD has the following main components:

- The syntactic framework defines the mechanisms for specifying the presentation syntax of RIF’s logic dialects. This syntax is used to define the semantics of a dialect and to exemplify its features.
- The semantic framework describes the mechanisms that are used for specifying the models of RIF’s logic dialects.
- The serialization framework is a mechanism for defining serialization mappings from each logic dialect’s presentation syntax to its concrete XML syntax.

The RIF syntactic framework for logic dialects is quite general. Along with Prolog-like positional terms, it supports named-argument terms (which are akin to tuples in relational databases, in which columns are named and the order of the columns is immaterial) and frames (which are assertions about objects and their properties à la F-logic [5]). The syntactic framework can be used to derive concrete dialects by controlled removal or addition of features and through a number of mechanisms, such as symbol spaces and signatures. For instance, the language of a dialect can be made higher-order (à la HiLog [6]), polymorphic and/or polyadic depending on the signatures that the dialect allows for the symbols in its language. The semantic framework can be specialized by tweaking the set of truth values, datatypes, external functions and predicates, and the entailment relation.

The design of RIF has been influenced by a number of previous efforts. First, several rule languages were developed specifically as Web languages, including SWRL [7], N3 [8], and MWeb [9]. The RIF framework for logic dialects, RIF-FLD, incorporates many of the semantic ideas and syntactic features of such languages. Presently, the Horn subsets of these languages can be exchanged via RIF-BLD. Once more expressive RIF dialects are defined using RIF-FLD, larger subsets of these languages will be able to interoperate. While RIF interfaces to OWL through a special combination semantics [10], SWRL is designed as an extension of OWL with rules. Most SWRL features are covered by RIF-BLD with the exception of different-from; thus it should be possible to exchange most SWRL rules via BLD. Second, earlier proposals for rule interchange on the Web, such as RuleML [11] and R2ML [12], have pursued goals similar to RIF. Specifically, RuleML influenced RIF in the direction of adopting striped XML syntax and provided the idea of structuring RIF dialects into an extensible family of languages according to their semantics and syntax. Conversely, RuleML adopted some features developed as part of the RIF Working Group. Shared RIF/RuleML implementations and use cases should lead to further convergence.

This article is a guide to the essentials of RIF-BLD (the Basic Logic Dialect of the Rule Interchange Format). From a theoretical perspective, RIF-BLD corresponds to the language of definite Horn rules with equality and a standard first-order semantics [13]. Syntactically, RIF-BLD has a number of extensions to support features such as objects and frames, as in F-logic [5]. Particular attention in the design of RIF was paid to integration with other relevant Web standards. Thus, RIF uses internationalized resource identifiers (IRIs) [14] and XML datatypes throughout, and RIF built-ins [15] are aligned with XPath and XQuery [16]. For compatibility with the existing Semantic Web standards, RIF also defines the syntax and semantics of integrated RIF-BLD/RDF and RIF-BLD/OWL languages [10]. Web compatibility notwithstanding, RIF was designed to enable interoperability among rule languages in general, so that its use would not be limited to the Web.

To make the above discussion more concrete, let us consider an example of a RIF-BLD rule set, which also happens to be in the RIF-Core dialect.

Example 1.1 (Rule-defined ternary predicate): A rule can be written in English to define the borrow concept based on the lends concept, whose contents are assumed to be defined by a set of stored facts. In English, this simple scenario is expressed as follows:

- A borrower borrows an item from a lender if the lender lends the item to the borrower.
- Pete lends Hamlet to Beth.

This information makes it possible to derive the fact Beth borrows Hamlet from Pete by the logical derivation principle of modus ponens. We represent the concepts borrow and lend, as well as the individuals Pete, Beth, and Hamlet using Web IRIs, such as cpt:borrow or "Beth""rif:iri. This colon-separated notation for IRIs is known as CURIE or compact URI [17]. A typical URI is a Web URL. It is usually long and cumbersome to use as an identifier. The CURIE notation alleviates this problem by allowing the user to declare prefixes—short names, such as cpt and bks in our example, which are treated as shorthands for longer fragments of IRIs (e.g., http://eg.com/concepts#). The CURIE notation, such as cpt:borrow, is simply a macro, which expands into http://eg.com/concepts#borrow. RIF also allows the user to specify one base IRI, which alleviates the problem of long IRIs in a different way: it lets users write relative IRIs, each being expanded to a concatenation of the base IRI (specified in the Base directive) and that relative IRI. In the example below, "Beth""rif:iri is a shorthand for "http://eg.com/people#Beth""rif:iri. The third way in which RIF supports shortening of IRIs is via the use of angle brackets, which obviate the need to write "rif:iri. For instance, instead of "http://eg.com/people#Pete""rif:iri one could write <http://eg.com/people#Pete>. This also works in conjunction with the base IRI, which allows us to simply write <Pete> in the example below. With this...
in mind, our example can now be expressed in RIF-BLD as follows:

```
Document(
  Base(<http://eg.com/people#>)
  Prefix(cpt <http://eg.com/concepts#>)
  Prefix(bks <http://eg.com/books#>)
  
  Group {
    Forall ?Borrower ?Item ?Lender {
      cpt:borrow(?Borrower ?Item ?Lender) :-
      cpt:lend(?Lender ?Item ?Borrower)
    }
    
    cpt:lend(<Pete> bks:Hamlet "Beth"^^rif:iri)
  }

  We note that the Group construct is a way of grouping rules and facts in RIF documents. Such groups can have identifiers and metadata, which can be referenced by other RIF documents.

  RIF documents are supposed to reside on the Web or be passed over the wire among applications. Such documents are expected to be written in XML—the concrete syntax of RIF. Unfortunately, XML is too verbose and cumbersome for formal logical specifications or for human consumption. To solve this problem, RIF dialects employ presentation syntaxes for defining their semantics and for developing use cases. In accordance with this principle, the above example uses the presentation syntax of RIF-BLD.

  In the following sections, we first give the human-readable presentation syntax of RIF-BLD and then use it to define the semantics. The machine-oriented XML syntax is introduced later, in Section IV.

II. THE PRESENTATION SYNTAX

The presentation syntax of RIF-BLD has had a tumultuous history in the RIF Working Group. Initially, it was designed as an ‘abstract’ syntax whose purpose was to serve as a vehicle for defining the semantics, for developing use cases, and for illustrating RIF by examples. It seemed unnecessary to create yet another concrete, parsable syntax alongside XML. Thus, the presentation syntax does not define precedence rules, delimiters, and other common attributes of parsable languages. Although this presentation syntax proved to be adequate for semantic definitions, it was found to be too verbose for anything but simple examples. As a result, the syntax was retrofitted with various shortcuts, which are described in [15]. We will occasionally use some of these shortcuts, especially the compact URI syntax mentioned earlier.

The presentation syntax of RIF-BLD is defined in two ways: as a specialization of the RIF logical framework, RIF-FLD [4], and also directly, from scratch. The first method is very short; it is intended for the reader who is familiar with RIF-FLD, primarily a new dialect designer or a user of multiple RIF dialects. The aim of the specialization method is to make it easy to relate RIF-BLD to other logical dialects, such as the forthcoming logic programming dialects. This also serves as an example of dialect design by specialization from the RIF framework—the preferred mode of specification for various future logical dialects. On the other hand, direct specification of the syntax is much longer, but it does not assume familiarity with RIF-FLD; it is intended for readers who are interested in the basic RIF dialect alone.

In this article, we describe only the direct specification method, as we do not have the space to introduce RIF-FLD here; an overview of RIF-FLD can be found in [18]. The XML syntax of RIF-BLD is discussed in Section IV. The presentation syntax is described in what sometimes is called “mathematical English,” a rigorous form of English for communicating mathematical definitions, and examples. An EBNF specification of the syntax is also given, but it cannot fully capture the presentation syntax, as the latter is not context-free.

Due to limitation of space and in order to focus on the essentials, this guide omits (or gives only cursory treatment to) several features of RIF-BLD, including lists, datatypes, and built-in predicates and functions. Integration with RDF and OWL has also been left out due to these considerations.

A. Alphabet of RIF-BLD

Definition 2.1 (Alphabet): The alphabet of the presentation language of RIF-BLD consists of

- a countably infinite set of constant symbols Const
- a countably infinite set of variable symbols Var (disjoint from Const)
- a countably infinite set of argument names, ArgNames (disjoint from Const and Var)
- connective symbols And, Or, and :-
- quantifiers Exists and Forall
- the symbols =, #, ##, ->, External, Import, Prefix, and Base
- the symbols Group and Document
- the auxiliary symbols (, ), [ ], <, >, and "".

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

Constants have the form "literal" symspace, where literal is a sequence of Unicode characters and symspace is an identifier for a symbol space (see below). The symbols =, #, and ## are used in formulas that define equality, class membership, and subclass relationships. The

2http://www.abstractmath.org/MM/MMMathEnglish.htm
symbol $\rightarrow$ is used in terms that have named arguments and in frame formulas. The symbol $\text{External}$ indicates that an atomic formula or a function term is defined externally (e.g., a built-in) and the symbols $\text{Prefix}$ and $\text{Base}$ enable abridged representations of IRIs [14].

The symbol $\text{Document}$ is used to specify RIF-BLD documents. $\text{Import}$ specifies an import directive, and $\text{Group}$ is used to organize RIF-BLD formulas into collections.

Symbol spaces are sets that partition the set of all constants $\text{Const}$. This important RIF concept is defined next.

**Definition 2.2 (Symbol spaces):** A symbol space has an identifier and a *lexical space*, which defines the exact syntax of the symbols in that symbol space. Some symbol spaces in RIF are used as identifiers of Web entities, and their lexical space consists of strings that syntactically are internationalized resource identifiers (IRIs) [14] (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).

Another important symbol space used in RIF is $\text{rif:local}$. Constants from this symbol space are intended to be used for individual, function, and predicate symbols that are encapsulated within the documents in which they occur and are not accessible from outside of these documents.\(^3\)

Every constant in $\text{Const}$ is in exactly one symbol space. \(\Box\)

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

### B. Terms

In RIF, the main building blocks used for constructing rules are called *terms*. RIF-BLD defines several kinds of terms: *constants* and variables, *positional* and *named-argument* terms, *equality*, *membership*, *subclass*, *frame*, *list*, and external terms. Thus “term” will be used to refer to any one of these constructs.

Below, the phrase *base term* means a simple, positional, or named-argument term, or a term of the form $\text{External}(t)$, where $t$ is a positional or named-argument term.

**Definition 2.3 (Term):** RIF-BLD defines several different types of logic terms. Here we describe the syntax of the most important ones.

1) *Constants and variables*. If $t \in \text{Const}$ or $t \in \text{Var}$ then $t$ is a *simple term*.

2) *Positional terms*. If $t \in \text{Const}$ and $t_1, ..., t_n, n \geq 0$, are base terms then $t(t_1, ..., t_n)$ is a *positional term*. Positional terms correspond to the usual terms and atomic formulas of classical first-order logic.

3) *Terms with named arguments*. A *term with named arguments* is of the form $t(s_1 \rightarrow v_1, ..., s_n \rightarrow v_n)$, where $n \geq 0$, $t \in \text{Const}$ and $v_1, ..., v_n$ are base terms.

\(^3\)To avoid confusion between RIF $\text{rif:local}$ constants and RDF blank nodes [19] we stress that these are completely different concepts. Blank nodes are existentially quantified variables, while $\text{rif:local}$ constants are constants that are simply localized within the documents where they occur. The notion of local constants is also distinct from Skolem constants: the latter are local constants, but local constants do not have to be Skolem.

The constant $t$ here represents a predicate or a function; $s_1, ..., s_n$ are *argument names* and $v_1, ..., v_n$ are argument values. The argument names, $s_1, ..., s_n$, are required to be pairwise distinct.

A term with named arguments $s_1 \rightarrow v_1, ..., s_n \rightarrow v_n$ is like a positional term except that the arguments, i.e., $v_1, ..., v_n$, are named (with $s_1, ..., s_n$) and the order between any pair of named arguments, $s_i \rightarrow v_i$ and $s_j \rightarrow v_j$, is considered to be immaterial. Note that a term of the form $f()$ is, trivially, a positional term as well as a term with named arguments.

Terms with named arguments serve to support the exchange of languages that permit argument positions of predicates and functions to be named (and where the order of the named arguments does not matter).

4) *Equality terms*. $t = s$ is an *equality term*, if $t$ and $s$ are base terms.

5) *Class membership terms* (or just *membership terms*). $t##s$ is a *membership term* if $t$ and $s$ are base terms.

6) *Subclass terms*. $t##s$ is a *subclass term* if $t$ and $s$ are base terms.

7) *Frame terms*. $t[p_1 \rightarrow v_1, ..., p_n \rightarrow v_n]$ is a *frame term* (or simply a *frame*) if $t, p_1, ..., p_n, v_1, ..., v_n, n \geq 0$, are base terms.

Membership, subclass, and frame terms are used to describe objects and class hierarchies in object-oriented logic languages such as F-logic [5].

8) *Extraneously defined terms*. If $t$ is a positional or a named-argument term then $\text{External}(t)$ is an *exactly defined term*.

External terms represent built-in function and predicate invocations as well as “procedurally attached” function or predicate invocations. Procedural attachments are often provided by rule-based systems, and external terms constitute a way of supporting them in RIF.

\(\Box\)

Observe that the argument names of frame terms, $p_1, ..., p_n$, are base terms and, as a special case, they can be variables. In contrast, terms with named arguments can use only the symbols from $\text{ArgNames}$ for their argument names. They cannot be constants from $\text{Const}$ or variables from $\text{Var}$. Thus

"http://ex.ca/person"$^\text{rif:iri}$(*first* $\rightarrow$ *Jo* $^\text{xs:string}$) is a syntactically correct term (assuming $\text{first} \in \text{ArgNames}$) with one named argument, while

"http://ex.ca/person"$^\text{rif:iri}$(*last* $^\text{xs:string}$ $\rightarrow$ *7L*)
"http://ex.ca/person"$^\text{rif:iri}$(*7N* $\rightarrow$ *Jones* $^\text{xs:string}$)

are not syntactically correct terms: in the first case, a string constant, "last"$^\text{xs:string}$, is used as an argument name and in the second a variable, ?N.

The reasons for these restrictions have to do with the complexity of unification. If constants were allowed as argument names then it would be possible to equate them using equality terms, thus turning unification into set unification, which is...
C. Formulas

Any term (positional or with named arguments) of the form $p(...)$, where $p\in\text{Const}$, is also an atomic formula. Equality, membership, subclass, and frame terms are also atomic formulas. An externally defined term of the form $\text{External}(\varphi)$, where $\varphi$ is an atomic formula, is also an atomic formula, called an externally defined atomic formula. Note that simple terms (constants and variables) are externally defined.

External(...)

Atomic formulas with the help of logical connectives.

Definition 2.4 (Formula): A formula can have one of the following forms:

1) Atomic: An atomic formula is also a formula.

2) Condition formula: A condition formula is either an atomic formula or a formula that has one of the following forms:
   - Conjunction: If $\varphi_1, ..., \varphi_n, n \geq 0$, are condition formulas then so is $\text{And}(\varphi_1, ..., \varphi_n)$, called a conjunctive formula. As a special case, $\text{And}()$ is allowed and is treated as a tautology, i.e., a formula that is always true.
   - Disjunction: If $\varphi_1, ..., \varphi_n, n \geq 0$, are condition formulas then so is $\text{Or}(\varphi_1, ..., \varphi_n)$, called a disjunctive formula. As a special case, $\text{Or}()$ is permitted and is treated as a contradiction, i.e., a formula that is always false.
   - Existentials: If $\varphi$ is a condition formula and $?V_1, ..., ?V_n, n>0$, are distinct variables then $\text{Exists}\ ?V_1, ..., ?V_n(\varphi)$ is an existential formula.

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

3) Rule implication: $\varphi :- \psi$ is a formula, called rule implication, if:
   - $\varphi$ is an atomic formula or a conjunction of atomic formulas,
   - $\psi$ is a condition formula, and
   - none of the atomic formulas in $\varphi$ is an externally defined term (i.e., a term of the form $\text{External}(...)$).

Note that external terms can occur in the arguments of atomic formulas in the rule conclusion, but they cannot occur as atomic formulas. For instance, $\text{External}(p(...)) :- \text{body}$ is not allowed, but $q(\text{External}(p(...))) :- \text{body}$ is a valid rule implication. In this case, $p$ is an external function. The semantics is arranged so as to make the above rule equivalent to the rewritten version $q(?v) :- \text{And(body ?v = \text{External}(p(...)))}$, where $?v$ is a fresh variable. This has the net effect that $p(...)$ is evaluated to some value, $?v$, and $q(?v)$ is derived, if body is true, where body may instantiate variables occurring as arguments to $p(...)$. Observe that equality is allowed in rule conclusions, as the RIF Working Group considers it to be an important feature for Semantic Web applications. However, this feature might still be restricted or eliminated in the final recommendation due to difficulties with the logically complete implementation of equality. An example of a rule with such an equality is

\begin{align*}
\text{Forall}\ ?x\ ?y\ ?t (\text{ex:color}(?x ?t) = \text{ex:color}(?y ?t)) :- \text{And(ex:night(?t) ?x#ex:Cat ?y#ex:Cat) (at night all cats have the same color). If ex:night("23:12:00"^^xs:time) is true and ex:Jane and ex:Tarzan are both known to be cats, then our rule implies the equality ex:color(ex:Jane "23:12:00"^^xs:time) = ex:color(ex:Tarzan "23:12:00"^^xs:time)). Thus, if in addition we know ex:likes(ex:Ted ex:color(ex:Jane "23:12:00"^^xs:time)), we can derive, by substitutivity, ex:likes(ex:Ted ex:color(ex:Tarzan "23:12:00"^^xs:time)). In general, applying the axioms of equality, especially substitutivity, is computationally expensive. The difficulties associated with unrestricted equality in rule conclusions could be alleviated by introducing conformance levels (cf. Section V). For instance, a certain conformance level might require systems to support only equality facts and only those that equate ground terms. Another level might restrict equality even further by requiring support for equations among constants only. Yet another conformance level might mandate support for oriented equations alone.

4) Universal rule: If $\varphi$ is a rule implication and $?V_1, ..., ?V_n, n>0$, are distinct variables then $\text{Forall}\ ?V_1, ..., ?V_n(\varphi)$ is a universal rule formula. It is required that all the free variables in $\varphi$ occur among the variables $?V_1, ..., ?V_n$ in the quantification part. Generally, an occurrence of a variable $?v$ is free in $\varphi$ if it is not inside a subformula of $\varphi$ of the form $\text{Exists}\ ?v(\psi)$ and $\psi$ is a formula. Universal rules are also referred to as RIF-BLD rules.

5) Universal fact: If $\varphi$ is an atomic formula and $?V_1, ..., ?V_n, n>0$, are distinct variables then $\text{Forall}\ ?V_1, ..., ?V_n(\varphi)$ is a universal fact formula, provided that all the free variables in $\varphi$ occur among the variables $?V_1$
Universal facts are treated as rules without premises.

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

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

7) **Document:** An expression of the form $\text{Document}(\text{directive}_1 \ldots \text{directive}_n \; \Gamma)$ is a RIF-BLD document formula (or simply a document formula), if

- $\Gamma$ is an optional group formula; it is called the group formula associated with the document.
- $\text{directive}_1, \ldots, \text{directive}_n$ is an optional sequence of directives. A directive can be a base directive, a prefix directive or an import directive.
  - A base directive has the form $\text{Base}(\text{iri})$, where iri is a Unicode string in the form of an IRI [14]. The Base directive defines a syntactic shortcut for expanding relative IRIs into full IRIs, as described in [15].
  - A prefix directive has the form $\text{Prefix}(p \; <v>)$, where $p$ is an alphanumeric string that serves as the prefix name and $<v>$ is an expansion for $p$—a string that forms an IRI. (An alphanumeric string is a sequence of ASCII characters, where each character is a letter, a digit, or an underscore “_”, and the first character is a letter.) Like the Base directive, the Prefix directives define shorthands to enable the compact URI representation (cf. Example 1.1) for constants that come from the symbol space rif:iri (we call such constants rif:iri constants).
  - An import directive can have one of these two forms: $\text{Import}(<t>)$ or $\text{Import}(<t> <p>)$. Here $t$ is a rif:iri constant and $p$ is a term. The constant $t$ indicates the location of another document to be imported and $p$ is called the profile of import.

Section III of this document defines the semantics for the directive $\text{Import}(<t>)$ only. The two-argument directive, $\text{Import}(<t> <p>)$, is intended for importing non-RIF-BLD documents, such as rules from other RIF dialects, RDF data, or OWL ontologies. The profile, $p$, indicates what kind of entity is being imported and under what semantics (for instance, the various RDF entailment regimes have different profiles). The semantics of $\text{Import}(<t> <p>)$ (for various $p$) are expected to be given by other specifications on a case-by-case basis. For instance, [10] defines the semantics for the profiles that are recommended for importing RDF and OWL.

A document formula contains at most one Base directive. If present, this directive comes first. It may be followed by any number of Prefix directives, then by any number of Import directives.

The next example illustrates a function that is defined using a recursive equality in the rule head. It shows that, as a Horn logic with equality, RIF-BLD subsumes the expressive power of first-order functional programming, where only oriented equations are used. While typical of functional-logic programming, this facility is currently out of scope for most logic rule languages. This, along with the previously discussed complexity and implementation issues with unrestricted equality, explains why the RIF Working Group has been cautious about including equality in rule heads in RIF-BLD.

**Example 2.1 (The factorial function):** The rule set in this example has two formulas. One is an atomic equation and the other a rule with an equality in the conclusion. At first glance, the rule seems simple: it uses two built-in functions (designated with the External construct), $\text{func:numeric-multiply}$ and $\text{func:numeric-subtract}$, and one built-in predicate, $\text{pred:numeric-greater-than}$. The rule does not even use recursive predicates and seems to be rather innocent. However, this first impression is deceptive, since the factorial function has a recursive occurrence within the equation in the rule head.

```xml
Document { 
  Prefix(pred <http://www.w3.org/2007/rif-builtin-predicate#>)
  Prefix(func <http://www.w3.org/2007/rif-builtin-function#>)
  Prefix(xs <http://www.w3.org/2001/XMLSchema#>)
  Group { 
    _factorial("0"^^xs:integer) = "1"^^xs:integer
    Forall ?N { 
      _factorial(?N) = 
        External(func:numeric-multiply { 
          _factorial{ 
            External(func:numeric-subtract { 
              ?N "1"^^xs:integer) } ) } ) 
          pred:numeric-greater-than { 
            ?N "0"^^xs:integer) }
    ) 
  } 
  ... other rules here can use _factorial ...
}
```

Semantically, this RIF-BLD document represents a world in which the following equalities are true:

```
(_factorial("i"^^xs:integer) = "j"^^xs:integer | i ≥ 0, j = i!).
```

The ostentatious underscore in front of the factorial function symbol is a shorthand for the constant "factorial" rif:local. The use of the rif:local symbol space here indicates that the name of that function is local, i.e., it is not accessible from other documents. It can, however, be accessed in definitions of other functions or predicates within the same document. The semantics of local constants is given in Section III.

D. RIF-BLD Annotations in the Presentation Syntax

RIF-BLD allows every term and formula (including terms and formulas that occur inside other terms and formulas) to be optionally preceded by one annotation of the form (*
id $\varphi$ $\ast$), where \texttt{id} is a \texttt{rif:iri} constant and $\varphi$ is a frame formula or a conjunction of frame formulas. Both items inside the annotation are optional. The \texttt{id} part represents the identifier of the term or formula to which the annotation is attached and $\varphi$ is the metadata part of the annotation. RIF-BLD does not impose any restrictions on $\varphi$ apart from what is stated above. This means that $\varphi$ may include variables, function symbols, constants from the symbol space \texttt{rif:local}, and so on. The purpose of annotations is to endow RIF with a general metadata facility, but semantically annotations are just comments; they are intended to be specified by document authors. The restrictions on annotations are intended to make their syntax simple and yet sufficient for supporting very general forms of metadata.

Document formulas with and without annotations will be referred to as \textit{RIF-BLD documents}. A convention is used to avoid syntactic ambiguity in the above definition. For instance, in \texttt{(* id $\varphi$ *) $t$[w $\rightarrow$ v]} the metadata annotation could be attributed to the term $t$ or the entire frame $t$[w $\rightarrow$ v]. The convention is that the above annotation is considered to be syntactically attached to the entire frame. Yet, some slots of $\varphi$ can be used to provide metadata targeted to the object part $t$ of the frame. Here is an example:

\begin{verbatim}
(* md[meta_for_frame->"about frame"^^xs:string] 
   meta_for_object->"about t"^^xs:string 
   meta_for_properties->"about w"^^xs:string) *)

t[w $\rightarrow$ v]
\end{verbatim}

Generally, the convention associates each annotation to the largest term or formula it precedes.

Although RIF does not prescribe any particular vocabulary for metadata, one can be expected to use Dublin Core\(^4\), RDFS, and OWL properties for metadata, along the lines of Section 7.1 of the OWL recommendation [1]—specifically owl:versionInfo, rdfs:label, rdfs:comment, rdfs:seeAlso, rdfs:isDefinedBy, dc:creator, dc:description, dc:date, and foaf:maker.

**E. Well-formed Formulas**

Not all formulas or documents are well-formed in RIF-BLD. The well-formedness restriction is similar to standard first-order logic: it is required that no constant appear in more than one context. We will not rehash here the tedious details of the definitions from [2], Section 2.5. Informally, unique context means that no constant symbol can occur within the same document as an individual, a function symbol, or a predicate in different places. Likewise, a symbol cannot occur as an external predicate or function in one place and as a normal predicate or function symbol in another. However, the same symbol can occur as a function symbol of different arities or as a predicate symbol of different arities.

**F. EBNF Grammar for the Presentation Syntax of RIF-BLD**

Until now, we have been using mathematical English to specify the syntax of RIF-BLD. Since tool developers might prefer a more succinct overview of the syntax using familiar grammar notation, the RIF-BLD specification also supplies an EBNF definition. However, one should keep in mind the following limitations of EBNF with respect to the presentation syntax:

- The syntax of first-order logic is not context-free, so EBNF cannot capture the syntax of RIF-BLD precisely. For instance, it cannot capture the requirement that all variables under a quantifier must be distinct. As a result, the EBNF grammar defines a proper superset of RIF-BLD: not all formulas that are derivable using the EBNF productions are well-formed formulas in RIF-BLD. For implementing a strict syntax validator, the corresponding mathematical English definitions must be used.

- The EBNF grammar does not address all details of how constants and variables are represented, and it is not sufficiently precise about precedence rules, delimiters, and escape symbols. White space is informally used as a delimiter, and is implied in productions that use Kleene star. For instance, \texttt{TERM*} is to be understood as \texttt{TERM TERM ... TERM}, where each space abstracts from one or more blanks, tabs, newlines, etc.

The EBNF grammar for the RIF-BLD presentation syntax is as follows:

**Rule Language:**

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

**Base** ::= 'Base' '(' ANGLEBRACKIRI ')

**Prefix** ::= 'Prefix' '(' Name ANGLEBRACKIRI ')

**Import** ::= IRIMETA? 'Import' '(' ANGLEBRACKIRI PROFILE? ')

**Group** ::= IRIMETA? 'Group' '(' '}' (RULE | Group)* ')

**RULE** ::= (IRIMETA? 'Forall' Var+ '->' ATOMIC* ')' | 'Or' '(' FORMULA* ')' | 'And' '(' (RULE | Group)* ')' | 'Implies' '(' ATOMIC ')'

**CLAUSE** ::= Implies | ATOMIC

**PROFILE** ::= ANGLEBRACKIRI

**Condition Language:**

**FORMULA** ::= IRIMETA? 'And' '(' FORMULA* ')' | IRIMETA? 'Or' '(' FORMULA* ')' | IRIMETA? 'Exists' Var+ '(' FORMULA ')'

**ATOMIC** ::= IRIMETA? (Atom | Equal | Member | Subclass | Frame)

**Atom** ::= UNITERM

**UNITERM** ::= Const '(' (TERM* | (Name '->' TERM)* ')' ')'

**Equal** ::= TERM '=' TERM

**Member** ::= TERM '##' TERM

**Subclass** ::= TERM '##' TERM

**Frame** ::= TERM '(' (TERM '->' TERM)* ')' ')

**TERM** ::= IRIMETA? (Const | Var | Expr | 'External' '(' Expr ')')

**Expr** ::= UNITERM

**Const** ::= '##' UNICODESTRING SYMSPACE | CONSTSHORT

**Name** ::= UNICODESTRING

**Var** ::= '??' UNICODESTRING

**SYMSPACE** ::= ANGLEBRACKIRI | CURIE

**Annotations:**

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

\(^4\text{http://dublincore.org/}\)
1) EBNF for the Condition Language: The Condition Language represents formulas that can be used as queries or in the premises of RIF-BLD rules. The production rule for the non-terminal FORMULA represents RIF condition formulas (defined earlier). The connectives And and Or define conjunctions and disjunctions of conditions, respectively. Exists introduces existentially quantified variables. Here Var+ stands for the list of variables that are free in FORMULA. RIF-BLD conditions permit only existential variables. A RIF-BLD FORMULA can also be an ATOMIC term, i.e., an Atom, External Atom, Equal, Member, Subclass, or Frame. A TERM can be a constant, variable, Expr, or External Expr.

The presentation syntax does not commit to any particular vocabulary and permits arbitrary Unicode strings in constant symbols, argument names, and variables. Constant symbols have the following form: "UNICODESTRING" SYMSPACE, where SYMSPACE is an ANGELBRACKIRI or is a CURIE that represents the identifier of the symbol space of the constant. UNICODESTRING is a Unicode string from the lexical space of that symbol space. ANGELBRACKIRI is an IRI enclosed in angle brackets, as in N3. For instance, <http://eg.com/foo/bar>. The CURIE syntax has been introduced earlier.

Constant symbols can have several short forms, which are represented by the non-terminal CONSTSHORT and are defined in [15]. Apart from the CURIE notation for IRI constants and the underscore notation for local symbols (cf. _factorial in Example 2.1), other shortcuts exist for strings (e.g., "abc" for "abc" xs:string) and integers (e.g., 123 for "abc" xs:integer).

Names are Unicode character sequences. Variables are composed of UNICODESTRING symbols prefixed with a ?-sign. Equality, membership, and subclass terms are self-explanatory. An Atom and Expr (expression) can either be positional or have named arguments. A frame term is a term composed of an object Id and a collection of attribute-value pairs. The term External(Atom) represents invocations of externally defined predicates, such as built-ins. Likewise, External(Expr) represents invocations of externally defined functions.

Example 2.2 (RIF-BLD conditions): This example shows conditions that are composed of atoms, expressions, frames, and existentials. In frame formulas, variables are shown in the positions of object Ids, object properties, and property values. When convenient, we take advantage of the CURIE notation and other shortcuts.

2) EBNF for the Rule Language: The EBNF for RIF-BLD rules and documents is given in Section II-F. A RIF-BLD Document consists of an optional Base directive, followed by any number of Prefixes and then any number of Imports. These then may be followed by an optional Group. Base and Prefix are employed by the shortcut mechanisms for IRIs. IRI has the form of an internationalized resource identifier as defined by [14]. An Import directive indicates the location of a document to be imported and an optional profile. A RIF-BLD Group is a collection of any number of RULE elements along with any number of nested Groups.

Rules are generated using CLAUSE elements. The RULE production has two alternatives:
- In the first, a CLAUSE is in the scope of the Forall quantifier. In that case, all variables mentioned in CLAUSE are required to also appear among the variables in the Var+ sequence.
- In the second alternative, CLAUSE appears on its own. In that case, CLAUSE cannot have variables.

Var, ATOMIC, and FORMULA were defined as part of the syntax for positive conditions in Section II-F1. In the CLAUSE production, ATOMIC is what is usually called a fact. An Implies rule can have an ATOMIC element or a conjunction of ATOMIC elements as its conclusion; it has a FORMULA as its premise. Note that, by Definition 2.4, externally defined atoms (i.e., formulas of the form External(Atom)) are not allowed in the conclusion part of a rule (ATOMIC does not expand to External).

Example 2.3 (RIF-BLD rules): This example shows a business rule from [21].

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

Again, we employ various shortcuts, including the compact URI notation. Two versions of the main part of the document are given. In the first, all variables are quantified universally outside of the rule. In the second, some variables are quantified existentially in the rule premise. Semantically, both versions are equivalent according to Section III.
In defining the semantics, we employ only the full syntax of RIF-BLD. The various shortcuts permitted by the syntax (e.g., the Prefix and Base directives and the CURIEs) are assumed to be already expanded into their full form by a simple preprocessor. To save space, in describing the semantics we omit lists and datatypes, and we simplify the semantics of external functions and predicates.

A. Semantic Structures

The semantics of RIF-BLD is an adaptation of the standard semantics for Horn clauses. Although this semantics is specified using general models while the semantics for Horn clauses is usually given using Herbrand models, the two styles of semantics are known to be equivalent [22]. We will use \( TV \) to denote \( \{ l.f. \} \)—the set of truth values used in the semantics. \( TV \) is used in RIF because it is intended to address (through the RIF-FLD framework) a range of logic languages, including those that are based on multi-valued logics. Since RIF-BLD is based on the classical two-valued logic, its \( TV \) set is particularly simple.

The key concept in a model-theoretic semantics for a logic language is the notion of semantic structures [23], which is defined next.

Definition 3.1 (Semantic structure, \( \mathcal{I} \)): A semantic structure, \( \mathcal{I} \), is a tuple of the form \( \langle TV, DTS, D, D_{ind}, D_{func}, I_C, I_Y, I_F, I_{NF}, I_{frame}, I_{sub}, I_{ind}, I_{ext}, I_{truth} \rangle \). Here \( D \) is a non-empty set of elements called the domain of \( \mathcal{I} \), and \( D_{ind} \), \( D_{func} \) are nonempty subsets of \( D \). \( D_{ind} \) is used to interpret the elements of \( Const \) that play the role of individuals and \( D_{func} \) is used to interpret the constants that play the role of function symbols. As before, \( Const \) denotes the set of all constant symbols and \( Var \) denotes the set of all variable symbols. \( DTS \) denotes a set of identifiers for primitive datatypes, which are mostly borrowed from XML Schema [24]. These include datatypes such as \( xs:string \) and \( xs:integer \). A full description of supported datatypes is found in [15]. To preserve the main focus of this article, we will omit the semantics of datatypes in this exposition.

The remaining components of \( \mathcal{I} \) are total mappings defined as follows:

1) \( I_C \) maps \( Const \) to \( D \). This mapping interprets constant symbols. In addition:
   - If a constant, \( c \in Const \), is an individual then it is required that \( I_C(c) \in D_{ind} \).
   - If \( c \in Const \) is a function symbol (positional or with named arguments) then it is required that \( I_C(c) \in D_{func} \).

2) \( I_Y \) maps \( Var \) to \( D_{ind} \). This mapping interprets variable symbols.

3) \( I_F \) maps \( D \) to total functions \( D_{ind}^* \rightarrow D \) (here \( D_{ind}^* \) is a set of all finite sequences over the domain \( D_{ind} \)). This mapping interprets positional terms. In addition:
   - If \( d \in D_{func} \) then \( I_F(d) \) must be a function \( D_{ind}^* \rightarrow D_{ind} \).
   - This implies that when a function symbol is applied to arguments that are individual objects then the result is also an individual object.
We also define the following generic mapping from terms to 

\[ I_{\text{NF}}(D) = \text{SetOfFiniteSets}(\text{ArgNames} \times D_{\text{ind}}) \rightarrow D. \]

This mapping interprets function symbols with named arguments. It is also required that:

- If \( d \in D_{\text{func}} \) then \( I_{\text{NF}}(d) \) must be a function \( \text{SetOfFiniteSets}(\text{ArgNames} \times D_{\text{ind}}) \rightarrow D_{\text{ind}}. \)

The mapping \( I_{\text{NF}} \) is analogous to the interpretation of positional terms via the mapping \( I_{f} \) with two differences:

- Each pair \( <s, v> \in \text{ArgNames} \times D_{\text{ind}} \) represents an attribute-value pair instead of just a value in the case of a positional term.
- The arguments of a term with named arguments constitute a finite set of attribute-value pairs rather than a finite ordered sequence of simple elements.

So, the order of the arguments in named-argument terms is immaterial.

5) \( I_{\text{frame}} \) maps \( D_{\text{ind}} \) to total functions of the form \( \text{SetOfFiniteBags}(D_{\text{ind}} \times D_{\text{ind}}) \rightarrow D. \)

This mapping interprets frame terms. An argument \( d \in D_{\text{ind}} \) of \( I_{\text{frame}} \) represents an object, and \( \{<a_1, v_1>, ..., <a_k, v_k>\} \) is a finite bag of attribute-value pairs for \( d \).

We will see shortly how \( I_{\text{frame}} \) is used to determine the truth valuation of frame terms.

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

6) \( I_{\text{sub}} \) gives meaning to the subclass relationship. It is a total mapping of the form \( D_{\text{ind}} \times D_{\text{ind}} \rightarrow D. \)

An additional restriction in Section III-C ensures that the operator \( \## \) is transitive, i.e., that \( c_1 \# \# c_2 \) and \( c_2 \# \# c_3 \) imply \( c_1 \## c_3. \)

7) \( I_{\text{isa}} \) gives meaning to class membership. It is a total mapping of the form \( D_{\text{ind}} \times D_{\text{ind}} \rightarrow D. \)

An additional restriction in Section III-C ensures that the relationships \( \# \) and \( \# \# \) have the usual property that all members of a subclass are also members of the superclass, i.e., that \( o \# c_1 \) and \( c_1 \## c \ldots \## c_3 \) imply \( o \# c_3 \).

8) \( I_{\text{a}} \) is a mapping of the form \( D_{\text{ind}} \times D_{\text{ind}} \rightarrow D. \) It gives meaning to the equality operator.

9) \( I_{\text{truth}} \) is a mapping of the form \( D \rightarrow TV. \) It is used to define truth valuation for formulas.

10) \( I_{\text{external}} \) is a mapping that is used to give meaning to External terms. It maps symbols in \( \text{Const} \) designated as external to fixed functions of appropriate arity. Typically, external terms are invocations of built-in functions or calls to external sources, and their fixed interpretations are determined by the specification of those built-ins and external sources.

We also define the following generic mapping from terms to \( D \), which we denote by \( I \).

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

Here we use \{...\} to denote a set of attribute-value pairs.

- \( I[o[a_1->v_1 \ldots a_k->v_k]] \)
- \( I_{\text{frame}}(I(<o>(v_1), ..., I(<o>(v_n)) \}) \)
- \( I_{\text{frame}}(I(<o>(v_1), ..., I(<o>(v_n)) \}) \)

Here \{...\} denotes a bag of attribute-value pairs. Jumping ahead, we note that duplicate elements in such a bag do not affect the value of \( I_{\text{frame}}(I(<o>)) \)---see Section III-C. For instance, \( I(o[a->b a->b]) \) is defined.

- \( I(c_1##c_2) = I_{\text{sub}}(I(c_1), I(c_2)) \)
- \( I(c_1+c_2) = I_{\text{isa}}(I(c_1), I(c_2)) \)
- \( I(x=y) = I_{\text{sub}}(I(x), I(y)) \)
- \( I(\text{External}(p(s_1 \ldots s_n))) = I_{\text{external}}(p)(I(s_1), ..., I(s_n)). \)

In addition, RIF-BLD imposes certain restrictions on datatypes so that they would be interpreted as intended (for instance, that the constants in the symbol space \( \text{xs:integer} \) are interpreted by integers). Details can be found in [2].

B. RIF-BLD Annotations in the Semantics

RIF-BLD annotations are stripped before the mappings that constitute RIF-BLD semantic structures are applied. Likewise, they are stripped before applying the truth valuation, \( TVal_L \), defined in the next section. Thus, annotations have no effect on the formal semantics of this Horn logic variant.

Note that, although identifiers and metadata associated with RIF-BLD formulas are ignored by the semantics, they can be extracted by XML tools. The frame terms used to represent RIF-BLD metadata can then be combined with RIF-BLD rules and fed into rule reasoners. In this way, applications can be built to reason about metadata. However, RIF itself defines no particular semantics for metadata.

C. Interpretation of Non-document Formulas

This section establishes how semantic structures determine the truth value of RIF-BLD formulas other than document formulas. Truth valuation of document formulas is defined in the next section. Here we define a mapping, \( TVal_L \), from the set of all non-document formulas to \( TV \).

Observe that in case of atomic formulas, \( TVal_L(\phi) \) is defined essentially as \( I_{\text{truth}}(I(\phi)) \). Recall that \( I(\phi) \) is just an element of the domain \( D \) and \( I_{\text{truth}} \) maps \( D \) to truth values in \( TV \). This might seem strange to the reader who is used to textbook-style definitions, since normally the mapping \( I \) is defined only for terms that occur as arguments to predicates, not for atomic formulas. Similarly, truth valuations are usually defined via mappings from instantiated formulas to \( TV \), not from the interpretation domain \( D \) to \( TV \). This HiLog-style definition [6] is inherited from RIF-FLD [4] and is equivalent to a standard one for first-order languages such as RIF-BLD. In RIF-FLD, this style of definition is a provision for enabling future RIF dialects that support higher-order features, such as those of HiLog [6], FLORA-2 [25], [26], and others.
Definition 3.2 (Truth valuation): Truth valuation for well-formed formulas in RIF-BLD is determined using the following function, denoted $\textit{Val}_T$:

1) **Positional atomic formulas:** $\textit{Val}_T(x(t_1 \ldots t_n)) = I_{\text{truth}}(I(x(t_1 \ldots t_n)))$

2) **Atomic formulas with named arguments:**
   
   $\textit{Val}_T(p(s_1 \rightarrow v_1 \ldots s_k \rightarrow v_k)) = I_{\text{truth}}(I(p(s_1 \rightarrow v_1 \ldots s_k \rightarrow v_k)))$

3) **Equality:** $\textit{Val}_T(x = y) = I_{\text{truth}}(I(x) = I(y))$.
   - To ensure that equality has precisely the expected properties, it is required that:
     
     $I_{\text{truth}}(I(x) = I(y)) = t$ if $I(x) = I(y)$ and that
     
     $I_{\text{truth}}(I(x) = I(y)) = f$ otherwise.
   - This is tantamount to saying that $\textit{Val}_T(x = y) = t$ if and only if $I(x) = I(y)$.

4) **Subclass:** $\textit{Val}_T(c_1 \# c_2) = I_{\text{truth}}(I(c_1 \# c_2))$.
   - To ensure that $\#$ is transitive, i.e., $c_1 \# c_2$ and $c_2 \# c_3$ imply $c_1 \# c_3$, the following is required:
     
     - For all $c_1, c_2, c_3 \in D$, if $\textit{Val}_T(c_1 \# c_2) = \textit{Val}_T(c_2 \# c_3) = t$ then $\textit{Val}_T(c_1 \# c_3) = t$.

5) **Membership:** $\textit{Val}_T(o \# c_1) = I_{\text{truth}}(I(o \# c_1))$.
   - To ensure that all members of a subclass are also members of its superclasses, i.e., $o \# c_1$ and $c_1 \# c_2$ imply $o \# c_2$, the following restriction is imposed:
     
     - For all $o, c_1, c_2 \in D$, if $\textit{Val}_T(o \# c_1) = \textit{Val}_T(c_1 \# c_2) = t$ then $\textit{Val}_T(o \# c_2) = t$.

6) **Frame:** $\textit{Val}_T(o[a_1 \rightarrow v_1 \ldots a_k \rightarrow v_k]) = I_{\text{truth}}(I(o[a_1 \rightarrow v_1 \ldots a_k \rightarrow v_k]))$.
   - Since the bag of attribute-value pairs represents a conjunction of all the pairs, the following restriction is used, if $k > 0$:
     
     - $\textit{Val}_T(o[a_1 \rightarrow v_1 \ldots a_k \rightarrow v_k]) = t$ if and only if
     
     $\textit{Val}_T(o[a_1 \rightarrow v_1]) = \ldots = \textit{Val}_T(o[a_k \rightarrow v_k]) = t$.

7) **Externally defined atomic formula:**
   
   $\textit{Val}_T(\text{External}(t)) = I_{\text{truth}}(I(\text{external}(t)))$.

8) **Conjunction:** $\textit{Val}_T(\text{And}(c_1 \ldots c_n)) = t$ if and only if $\textit{Val}_T(c_1) = \ldots = \textit{Val}_T(c_n) = t$. Otherwise, $\textit{Val}_T(\text{And}(c_1 \ldots c_n)) = f$. The empty conjunction is treated as a tautology: $\textit{Val}_T(\text{And}(\)) = t$.

9) **Disjunction:** $\textit{Val}_T(\text{Or}(c_1 \ldots c_n)) = f$ if and only if $\textit{Val}_T(c_1) = \ldots = \textit{Val}_T(c_n) = f$. Otherwise, $\textit{Val}_T(\text{Or}(c_1 \ldots c_n)) = t$. The empty disjunction is treated as a contradiction: $\textit{Val}_T(\text{Or}(\)) = f$.

10) **Quantification:**
    - $\textit{Val}_T(\exists y \phi) = t$ if and only if for some $\exists^*$, described below, $\textit{Val}_T.(\phi) = \exists^* (\exists^* \phi)$.
    - $\textit{Val}_T(\forall y \phi) = \exists^* (\forall \phi) = \exists^*$ if and only if for every $\exists^*$, described below, $\textit{Val}_T.(\phi) = \exists^*$.

Here $\exists^*$ is a semantic structure of the form $<TV, DTS, D, D_{\text{ind}}, D_{\text{func}}, IC, I^\text{V}, I^\text{F}, I^{\text{IF}}, I_{\text{frame}}, I_{\text{sub}}, I_{\text{inst}}, I_{\text{fs}}, I_{\text{external}}, I_{\text{truth}},>$, which is exactly like $I$, except that the mapping $I^*\text{v}$, is used instead of $I_{\text{v}}$. $I^*\text{v}$ is defined to coincide with $I_{\text{v}}$ on all variables except, possibly, on $?v_1, \ldots, ?v_n$.

11) **Rule implication:**
    - $\textit{Val}_T(\text{conclusion} :- \text{condition}) = t$, if either
      
      $\textit{Val}_T(\text{conclusion}) = t$ or $\textit{Val}_T(\text{condition}) = f$.
    - $\textit{Val}_T(\text{conclusion} :- \text{condition}) = f$ otherwise.

12) **Groups of rules:**
    - If $\Gamma$ is a group formula of the form $\text{Group}(\phi_1 \ldots \phi_n)$ then
      
      - $\textit{Val}_T(\Gamma) = t$ if and only if $\textit{Val}_T(\phi_1) = t, \ldots, \textit{Val}_T(\phi_n) = t$.
      
      - $\textit{Val}_T(\Gamma) = f$ otherwise.

In other words, rule groups are treated as conjunctions.

D. Interpretation of Documents

RIF documents are sets of rules and are similar to Group formulas in that respect. However, they can also import other documents. This feature requires special care, since the imported documents can have rif:local constants and there might be name clashes. This would be a problem, if we recall that rif:local constants are completely encapsulated within the documents they appear in, so the same local constant may mean different things in different documents. This property of local constants brings the notion of semantic multi-structures into the picture. Semantic multi-structures are essentially similar to regular semantic structures but, in addition, they allow to interpret rif:local symbols that belong to different documents differently.

Definition 3.3 (Semantic multi-structure): A semantic multi-structure $I$ is a set $\{I, I; I^{\phi_1}, \ldots, I^{\phi_n}, \ldots\}$ of semantic structures adorned with distinct RIF-BLD formulas $\phi_1, \ldots, \phi_n$ (adorned structures can be thought of as formula-structure pairs). These structures must be identical in all respects except that the mappings $J_C, I_C, I^{\phi_1}, \ldots, I_C^{\phi_n}, \ldots$ which interpret the constants in Const by domain elements in $D$, may differ on the constants that belong to the rif:local symbol space.

As will become clear shortly, the above $I$ serves to interpret the main document formulas. The adorned structures of the form $I^{\phi_1}$ are used to interpret imported documents, and $J$ is used in defining entailment for non-document formulas. We can now define the semantics of RIF formulas.

Definition 3.4 (Truth valuation for document formulas): Let $\Delta$ be a document formula and let $\Delta_1, \ldots, \Delta_k$ be all the RIF-BLD document formulas that are imported directly or indirectly$^5$ into $\Delta$. Let $\Gamma, \Gamma_1, \ldots, \Gamma_k$ denote the respective group formulas associated with these documents. Let $I = \{I, I; I^{\Delta_1}, \ldots, I^{\Delta_k}, \ldots\}$ be a semantic multi-structure that contains semantic structures adorned with at least the documents $\Delta_1, \ldots, \Delta_k$. Then we define:

- $\textit{Val}_T(\Delta) = t$ if and only if $\textit{Val}_T(\Gamma) = \textit{Val}_T(\Delta_{\Gamma_1}(\Gamma_1) = \ldots = \textit{Val}_T(\Delta_{\Gamma_k}(\Gamma_k) = t$.

Note that this definition considers only those document formulas that are reachable via the one-argument import

$^5$That is, through another imported document.
directives. Two argument import directives are not covered here. Their semantics is defined by the document RIF RDF and OWL Compatibility [10].

Also note that some of the $\Gamma_i$ above may be missing since all parts in a document formula are optional. In this case, we assume that $\Gamma_i$ is a tautology, such as $a = a$, and every $TVal$ function maps such a $\Gamma_i$ to the truth value $t$.

For non-document formulas, we extend $TVal$ from regular semantic structures to multi-structures as follows: if $I = \{\mathcal{J}, \mathcal{I}, \ldots \}$ is a multi-structure and $\phi$ a non-document formula then $TVal_I(\phi) = TVal_J(\phi)$.

The above definitions make the intent behind rif:local constants clear: occurrences of such constants in different documents can be interpreted differently even if they have the same name. Therefore, each document can choose the names for its rif:local constants freely and without any risk of clashes with the names of local constants used in the imported documents. Hence rif:local constants are just a syntactic convenience introduced to facilitate rule interchange. This language feature, does not increase the expressive power of RIF-BLD, as such constants can be simulated in any multi-document rule language by simply renaming apart certain designated constants in different documents.

E. Logical Entailment

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

Definition 3.5 (Models): A multi-structure $I$ is a model of a formula, $\phi$, written as $I \models \phi$, iff $TVal_I(\phi) = t$. Here $\varphi$ can be a document or a non-document formula. □

Definition 3.6 (Logical entailment): Let $\varphi$ and $\psi$ be (document or non-document) formulas. We say that $\varphi$ entails $\psi$, written as $\varphi \models \psi$, if and only if for every multi-structure $I$ for which both $TVal_I(\varphi)$ and $TVal_I(\psi)$ are defined, $I \models \varphi$ implies $I \models \psi$. □

One consequence of the multi-document semantics of RIF-BLD is that local constants specified in one document cannot be queried from another document. For instance, if one document, $\Delta'$, has the fact

\[ \text{http://eg.com/p"rif:iri("abc"~rif:local)} \]

while another document formula, $\Delta$, imports $\Delta'$ and has the rule

\[ \text{http://eg.com/q"rif:iri(?X) :- \text{http://eg.com/p"rif:iri(?X)}} \]

then $\Delta \models \text{http://eg.com/q"rif:iri("abc"~rif:local)}$ does not hold. This is because the symbol "abc"~rif:local in $\Delta'$ and in $\Delta$ is treated as occurrences of different constants by semantic multi-structures.

The behavior of local symbols should be contrasted with that of rif:iri symbols, which are global. For instance, in the above scenario, let $\Delta'$ also have the fact

\[ \text{http://eg.com/p"rif:iri("http://cde"~rif:iri)} \]

Then $\Delta \models \text{http://eg.com/q"rif:iri("http://cde"~rif:iri) does hold.}

IV. XML Serialization

The goal of XML serialization of any RIF dialect, including BLD, is to provide a standard way of encoding the presentation syntax in XML so that the rules could be transmitted over the wire and exchanged among processors. The serialization of RIF-BLD includes the following components [2]:

- An XML schema document for the XML syntax.
- A mapping $\chi_{bld}$ from the RIF-BLD presentation syntax to XML.

The purpose of the XML schema document is obvious: as with any XML schema document, it defines which XML documents purporting to contain RIF rules are valid— a useful syntactic check. However, this is not enough. First, the semantics of RIF-BLD was specified using the presentation syntax because doing so directly with XML is infeasible. XML is too verbose and even the simplest of definitions of the kind we saw in Section III would span pages. The mapping $\chi_{bld}$ from the presentation syntax to XML is an indirect way to project the semantics on RIF-BLD’s XML documents. Second, recall that the syntax of RIF-BLD is not context-free and thus cannot be fully captured by EBNF. This limitation also applies to XML Schema. Again, the mapping $\chi_{bld}$ solves the problem, as the image of that mapping (which is a subset of all documents valid with respect to the XML schema) is precisely the set of all syntactically correct XML serializations of RIF-BLD.

To reflect this situation, we introduce two notions of syntactic correctness. The weaker notion checks correctness only with respect to XML Schema, while the stricter notion represents true syntactic correctness.

A valid RIF-BLD document in XML syntax is an XML document that is valid with respect to the XML schema of RIF-BLD as defined in [2].

An admissible RIF-BLD document in XML syntax is a valid RIF-BLD document in XML syntax that is the image of a well-formed RIF-BLD document in the presentation syntax under the presentation-to-XML syntax mapping $\chi_{bld}$.

Due to the lack of space, we cannot define here the XML schema of RIF-BLD or the mapping $\chi_{bld}$. Instead, we will explain the general principles of the XML serialization.

The XML serialization for RIF-BLD is based on an alternating (also known as striped) syntax [27]. A striped serialization views XML documents as objects and divides all XML tags into class descriptors, called type tags, and property descriptors, called role tags [11]. We follow the convention of using capitalized names for type tags and lowercase names for role tags.

The all-uppercase classes in the presentation syntax, such as FORMULA, become XML Schema groups. They are not visible in instance markup. The other classes as well as non-terminals and symbols (e.g., Exists or =) become XML elements with optional attributes, as shown below. Details can be found in the XML schema appendix of the RIF-BLD specification [2].
In the XML serialization of RIF-BLD some of the tag names directly reflect keywords of the presentation syntax. For instance, `And(...)` is serialized as `<And>...<And>`. Children of such `type tags` are wrapped into `role tags`. For instance, all children of the `And` type tag are `formula` role tags: `<And><formula>...</formula>...<formula>...</formula></And>`. Other tag names reflect the binary infix operators of the presentation syntax, such as `# =`. For instance, `...#...` is serialized as `<Member>...</Member>`. As before, child elements of the `Member` type tag are `role tags`—`instance` and `class` role tags in this case: `<Member><instance>...</instance><class>...</class></Member>`. The tags used in the serialization of the condition language are listed below.

- `And` (conjunction)
- `Or` (disjunction)
- `Exists` (quantified formula for 'Exists', containing declare and `formula` roles)
- `declare` (declare role, containing a `Var`
- `formula` (formula role, containing a `FORMULA`
- `Atom` (atom formula, positional or with named arguments)
- `External` (external call, containing a content role)
- `content` (content role, containing an Atom, for predicates, or `Expr`, for functions)
- `Member` (prefix version of member formula `#`)
- `Subclass` (prefix version of subclass formula `##`)
- `Frame` (Frame formula)
- `object` (Member/Frame role, containing a TERM or an object description)
- `op` (Atom/Expr role, containing a TERM or an object description)
- `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` (prefix version of Name/TERM `->` TERM pair as an Atom/Expr or Frame slot role, with fixed 'ordered' attribute)
- `Equal` (prefix version of term equation `='`)
- `Expr` (expression formula, positional or with named arguments)
- `left` (Equal left-hand side role)
- `right` (Equal right-hand side role)
- `Const` (individual, function, or predicate symbol, with optional 'type' attribute)
- `Name` (name of named argument)
- `Var` (serialized version of logic `?' variable)
- `id` (identifier role, containing `IRICONST`)
- `meta` (meta role, containing metadata as a Frame or Frame conjunction)

The `id` and `meta` elements, which are expansions of the `IRICOMMENT` element, can occur optionally as the first two children of any type (but not role) element. The XML syntax for symbol spaces uses the `type` attribute associated with the XML element `Const`. For instance, a literal in the `xs:dateTime` datatype is represented as `<Const type="&xs;dateTime">2007-11-23T03:55:44-02:30</Const>`.

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

The tags for the rule sublanguage also come in two flavors: those that correspond to the presentation syntax keywords (e.g., `Document`) and those corresponding to the infix operators (e.g., `:-` becomes `Implies`).

- `directive` (directive role, containing `Import`)
- `payload` (payload role, containing `Group`)
- `Import` (importation, containing location and optional profile)
- `location` (location role, containing `IRICOMMENT`)
- `profile` (profile role, containing `PROFILING`)
- `Group` (nested collection of `sentences`)
- `sentence` (sentence role, containing `RULE` or `Group`)
- `Forall` (quantified formula for `Forall`, containing declare and `formula` roles)
- `Implies` (prefix version of logic `:-` implication, containing if and then roles)
- `if` (antecedent role, containing `FORMULA`)
- `then` (consequent role, containing `ATOMIC` or `conjunct of ATOMICs`)

The following example illustrates XML serialization.

**Example 4.1 (Serialization of the introductory example):**

The following XML document is a serialization of Example 1.1 from the introductory section.

```xml
<!DOCTYPE Document [ 
<!ENTITY xs "http://www.w3.org/2001/XMLSchema#” 
<!ENTITY rif "http://www.w3.org/2007/rif#” 
<!ENTITY bks "http://eg.com/books#” ] > 
<Document 
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" 
  xmlns="http://www.w3.org/2007/rif#” 
  xml:base="http://eg.com/people#” > 
  <payload> 
    <Group> 
      <sentence> 
        <Forall> 
          <declare> <Var>Borrower</Var> </declare> 
          <declare> <Var>Lender</Var> </declare> 
          <declare> <Var>Item</Var> </declare> 
          <formula> 
            <Implies> 
              <if> 
                <Atom> 
                  <op> 
                    <Const type="&rif;iri">&cpt;lend</Const> 
                  </op> 
                </Atom> 
              </if> 
              <then> 
                <Atom> 
                  <op> 
                    <Const type="&rif;iri">&cript;borrow</Const> 
                    <args ordered="yes"> 
                      <Var>Borrower</Var> 
                      <Var>Item</Var> 
                      <Var>Lender</Var> 
                    </args> 
                  </op> 
                </Atom> 
              </then> 
            </Implies> 
          </formula> 
        </Forall> 
      </sentence> 
    </Group> 
  </payload> 
</Document>
```
V. CONFORMANCE

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

Let $T$ be a set of datatypes that includes the mandatory RIF datatypes defined in [15], and suppose $E$ is a set of external terms that includes the mandatory built-ins listed in [15]. We say that a formula $\varphi$ is a $BLD_{T,E}$ formula iff

- it is a well-formed RIF-BLD formula,
- all the datatypes used in $\varphi$ are in $T$, and
- all the externally defined terms used in $\varphi$ are in $E$.

A RIF processor is a software program that can produce admissible RIF documents (a RIF producer) or consume them (a RIF consumer). Intuitively, a conformant RIF-BLD processor is one that can interpret $\varphi$ correctly. The formal definition is as follows.

**Definition 5.1 (Conformant processor):** A RIF processor is a conformant $BLD_{T,E}$ consumer if and only if it implements a semantics-preserving mapping, $\mu$, from the set of all $BLD_{T,E}$ formulas to the language $L$ of the processor ($\mu$ does not need to be an “onto” mapping).

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

A RIF processor is a conformant $BLD_{T,E}$ producer if and only if it implements a semantics-preserving mapping, $\nu$, from the language $L$ of the processor to the set of all $BLD_{T,E}$ formulas ($\nu$ does not need to be an “onto” mapping).

Specifically, this means that for any pair $\varphi, \psi$ of formulas in $L$ for which $\varphi \models_L \psi$ is defined, $\varphi \models_L \psi$ if and only if $\nu(\varphi) \models_{BLD} \psi$.

Thus, compliance with RIF-BLD means implementation of a conformant RIF consumer, producer, or both. In addition to the above, RIF-BLD imposes the following requirements:

- Conformant BLD producers and consumers are required to support only the entailments of the form $\varphi \models_{BLD} \psi$, where $\psi$ is a closed RIF condition formula, i.e., a RIF condition in which every variable, $?V$, is in the scope of a quantifier of the form $\exists ?V$. Conformant BLD producers and consumers are also expected to preserve annotations wherever possible.

- Two other conformance requirements are still under discussion in the Working Group and might be removed in the final W3C recommendation. To understand them, recall that in Definition 5.1 the set of supported datatypes and built-ins must include the set of mandatory types and built-ins of RIF-BLD. The requirements under discussion state that conformant processors must also have a “strict” mode in which documents that refer to additional, non-mandatory datatypes and built-ins are rejected.

The first restriction above simply says that rule engines are required to support querying but not more general types of logical entailment. The strictness requirement is intended to allow RIF-BLD consumers to reject documents that contain datatypes and built-ins that are not defined by RIF and thus might not be understood by conformant processors.

RIF-BLD supports a wide variety of syntactic forms for terms and formulas, which creates infrastructure for exchanging syntactically diverse rule languages. The conformance clauses enable systems that do not support some of the syntax directly to still support it through syntactic transformations. For instance, disjunctions in rules premises can be eliminated through a standard transformation, such as replacing $p :- \lor(q,r)$ with a pair of rules $p :- q$, $p :- r$. Likewise, terms with named arguments can be reduced to positional terms by ordering the arguments lexicographically by their names and incorporating the ordered argument names into the predicate name. For instance, $p(bb->a aa->2)$ could be represented as $p_a_{bb}(2 1)$.

VI. CONCLUSIONS

RIF-BLD was the first specification that the W3C RIF Working Group approved for the “Candidate Recommendation” status. Prior to reaching that point, the specification had evolved with often dramatic twists and turns, e.g., inclusion and later exclusion of a sorted logic. RIF-BLD served as a basis for generalizations (RIF-FLD [4]) and specializations (RIF-Core). It has also been the focus of the RDF and OWL Compatibility document [10]. Other documents that have reached the Candidate Recommendation stage include RIF-PRD, the Production Rules Dialect [3], and RIF Datatypes and Built-ins [15].

A logically complete implementation of RIF-BLD consumer and producer processors requires the handling of unrestricted equality in rule conclusions. Theorem provers implementing the RIF-BLD superset of first-order logic with equality can be used as initial mapping targets (e.g., VampirePrime [28]). For better performance, they can then be specialized toward RIF-BLD. Conversely, implementations of subsets of RIF-BLD that restrict equality in various ways are promising, as several systems already support much of the needed functionality (e.g., FLORA-2 [26] and Ontobroker [29]). Missing are mostly the actual mappings from those systems to and from RIF-BLD’s XML.

Extensions of RIF-BLD and RIF-Core of the highest practical value are expected to be logic programming dialects with negation as failure. Such dialects, based on the well-founded [30] and answer-set [31] semantics for negation, are expected to be submitted to W3C by RIF user groups before the end of 2009. These new dialects are being defined as specializations of the general framework for logic dialects, RIF-FLD, which contains most of the requisite machinery and thus greatly simplifies the task. Since negation-as-failure constructs are widespread in industry and many rule engines provide them, this is expected to contribute to wider adoption of RIF.

More importantly, it is hoped that the foundation provided by BLD as the first RIF dialect and FLD as an extensible RIF framework will spur the creation of a general infrastructure for exchanging rule languages through dialects specified in a
consistent manner with a presentation syntax, a formal semantics, an XML serialization, and conformance clauses. The RIF Working Group currently lists VampirePrime [28] and SILK\(^6\) as two of the six on-going BLD implementations, along with three PRD, five DTB, and one Core-only implementation.\(^7\)

VII. ACKNOWLEDGEMENTS

This article is based on the document [2] by the Rules Interchange Format (RIF) Working Group. The authors extend special thanks to Jos de Bruijn, Gary Hallmark, Sandro Hawke, David Hirtle, Stella Mitchell, Leora Morgenstern, Igor Mozetic, Axel Polleres, Dave Reynolds, Christian de Sainte-Marie, and Chris Welty for contributing ideas, feedback, and insightful discussions. Our thanks go also to the IEEE TKDE reviewers, whose suggestions led to improvements of both this article and the RIF-BLD document.

REFERENCES


\(^6\)http://silk.semwebcentral.org
\(^7\)http://www.w3.org/2005/rules/wiki/Implementations


Harold Boley is Adjunct Professor at the Faculty of Computer Science, University of New Brunswick, and Leader of the Semantic Web Laboratory at the Institute for Information Technology, National Research Council Canada. His specification of Web rules through RuleML has found broad uptake, been combined with OWL to SWRL, and become the main input to RIF. His work on Rule Responder has enabled deployed distributed applications for the Social Semantic Web.

Michael Kifer is a Professor with the Computer Science Department, Stony Brook University. His work on knowledge representation, particularly on F-logic and HiLog, is among the most widely cited in Semantic Web research. He was twice recipient of the prestigious ACM-SIGMOD “Test of Time” award for his work on object-oriented database languages. Recently he also received SUNY Chancellor’s Award for Excellence in Scholarship.