RIF Core Design

W3C Editor's Draft @@not-published

This version:
@@not-published
Latest version:

Previous version:

Editors:
Harold Boley
Michael Kifer

Abstract

This document, developed by the Rule Interchange Format (RIF) Working Group, specifies the core design for a format that allows rules to be translated between rule languages and thus transferred between rule systems.

In Phase 1, the RIF Working Group is first defining a Core Condition Language. These conditions are then used as rule bodies to define a Core Horn Language. A human-oriented syntax, an XML syntax, and the semantics of the condition language and of the Horn rule language are given.

Status of this Document

May Be Superseded

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

Please Comment By @@no-due-date

Send comments to public-rif-wg@w3.org (assuming you're in the WG)

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 under the 5 February 2004 W3C Patent Policy. The Working Group maintains a public list of patent disclosures relevant to this document; that page also includes instructions for disclosing [and excluding] a patent. An individual who has actual knowledge of a patent which the individual believes contains Essential Claim(s) with respect to this specification should disclose the information in accordance with section 6 of the W3C Patent Policy.

Table of Contents

1. RIF Condition Language

(Editor's Note: This text is maintained on wiki page RIF Condition Language).

This proposal develops a set of fundamental concepts shared by the rule languages of interest to the RIF WG as outlined in the Design Roadmap http://lists.w3.org/Archives/Public/public-rif-wg/2006Feb/0255.html. Here we focus on the one part that is shared by Logic Programming rules, Production (Condition-Action) rules, Reactive (Event-Condition-Action) rules, Normative rules (Integrity Constraints), and queries. We call this part 'Conditions' and the proposed sublanguage 'RIF Condition Language' (as a working title).

Comment [AG1]: Are these official dialects at this time?
Should the notions of "dialect" and "sublanguage" be defined or elaborated in some way?

The RIF Condition Language is common ground for specifying syntactic fragments in several different dialects of RIF:

- Rule bodies in a declarative logic programming dialect (LP)

- Rule bodies in a first-order dialect (FO)
- Conditions in the bodies in production rules dialects (PR)
- The event and condition parts of the rule bodies in reactive rules dialects (RR)
- Integrity constraints (IC)
- Queries in LP dialects (QY)

Note that for rules this sublanguage is intended to be used only in the bodies, not their heads. The various RIF dialects diverge in the way they specify rule heads and other parts of their rules. We believe that by focusing on the condition part of the rule bodies we can achieve maximum syntactic (and some semantic) reuse among RIF dialects.

Since different semantics are possible for syntactically identical rule sets, we propose that the semantics of a RIF rule set be specified by a predefined attribute. We elaborate on this proposal in A.6 Semantics, but the idea is to organize the most important known semantic and syntactic features into a taxonomy (which could be extended to accommodate new semantics/syntax), and the values of the aforementioned attribute would come from that taxonomy.

The assumptions underlying the RIF Condition Language are explicated in A. Assumptions.

1.1. Positive Conditions

(Editor's Note: This text is maintained on wiki page Positive Conditions).

Introduction

The basis of the language is formed by conditions that can appear in the bodies of Horn-like rules with equality -- conjunctions and disjunctions of atomic formulas and equations (such rules reduce to pure Horn). We later extend our proposal to include builtins. As indicated in RIF Condition Language, the motivation is that this sublanguage can be shared among the bodies of the rules expressed in the RIF dialects LP, FO, PR, RR, and can also be used to uniformly express the body-like IC and QY sublanguages.

This document proposes a positive-condition syntax and semantics to account for webizing (the use of URIs for constants, predicates, and functions) and primitive data types (integers, floats, time, date, etc.). The main novelty is that there is no longer a wall between the domains of constants, functions, and predicates. Instead, all these symbols are drawn from the ***same*** domain. Separation between the different kinds of symbols is introduced through the mechanism of ***sorts***. For instance, we can introduce a sort **URI** for URIs, and the sorts **integer**, **float**, **time**, **string**, etc. for the corresponding data types. We can decide that certain sorts must be disjoint (integers, time) and others are not (eg,

Comment [AG2]: In formal logic the notion of "domain" is equivalent to "universe of discourse" which is a) specified as part of an interpretation and b) includes everything that exists for that interpretation. I don't think that is the use of the term you have in mind here. Do you want to say that there isonly one namespace for the non-logical vocabulary, as opposed to the classical procedure of requiring different syntactic elements for individual constants, predicates, and functions?

Formatted: Font: Bold, Italic

URI can be a subsort of **string**). We can control what sorts can be used for relation symbols (predicates), function symbols, etc. For example, we can decide that only strings can be predicates (which includes URIs, if **URI** is a subsort of **string**). Or we can decide that only **URI** and **localSymbol** (where **localSymbol** is some kind of a subsort of **string**) can be predicates and functions.

The first few sections on the syntax and semantics use a single domain for constants, functions, and predicates. The multisorted extension is described in the section "Multisorted Extensions".

SYNTAX

The following BNF syntax is for illustration/explanation purposes only. This is the essential BNF for a human-readable syntax, where Expressions and Atoms abstract from the underlying data model in that they can take a Herbrand (positional) or another form:

```
Var         ::= '?' NAME?
TERM        ::= Con | Var | Expr
Expr        ::= Con '(' TERM* ')'
Atom        ::= Expr
LITFORM     ::= Atom | TERM '=' TERM
QUANTIF     ::= 'Exists' Var+ '(' CONDIT ')'
CONJ        ::= 'And' '(' CONDIT* ')'
DISJ        ::= 'Or' '(' CONDIT* ')'
CONDIT      ::= LITFORM | QUANTIF | CONJ | DISJ
```

Comment [AG3]: I don't understand this production. What is the role of the 2nd question mark? Given what you say about anonymous variables below, this would make more sense to me as Var ::= '?' | ?NAME

Here the Herbrand form (Expr or Atom) applies its operator (Con) to positional TERM arguments using round parentheses, (... TERM ...). Notice that LITFORM stands for Literal Formula and anticipates the introduction of negated atoms later on. QUANTIF stands for Quantified Formula, which for Horn-like conditions can only be 'Exists' Formulas (Var+ variables should occur free in the scoped CONDIT, so 'Exists' can quantify them; free variables are discussed below). More explicitly than in logic programming, CONJ expresses formula conjunctions, and DISJ expresses disjunctions. Finally, CONDIT combines everything and defines RIF conditions, which can later be extended beyond LITFORM, QUANTIF, CONJ, or DISJ.

We initially assume that all constants (Con) belong to one logical sort: the sort of elementary entities. Relation names and function symbols are also in Con. Likewise, variables are initially not sorted and thus can range over all constants, expressions, etc. Both of these assumptions will be subsequently refined to allow multiple sorts for different data types. See section "Multisorted Extensions".

At this point we do not commit to any particular vocabulary for the names of variables and for constants. For instance, NAME could be any alphanumeric string and a variety of options could be used for Con. We leave the decision till

later time. A stand-alone '?' denotes an anonymous variable, each of whose occurrences is equivalent to a variable with a fresh NAME.

Comment [AG4]: See comment above.

Note that there are two uses of variables in the RIF Condition Language: free and quantified. All quantified variables are quantified explicitly, existentially (and also universally, later). We adopt the usual scoping rules for quantification from first-order logic. Variables that are not explicitly quantified are free.

The free variables are needed because we are dealing with conditions that occur in rule bodies only. When a condition occurs in such a rule body, the free variables in the condition are precisely those that also occur in the rule head. Such variables are quantified universally outside of the rule, and the scope of such quantification is the entire rule. For instance, the variable ?X in the rule below is free in the condition that occurs in the rule body, but it is universally quantified outside of the rule.

Comment [AG5]: Not that it is necessarily a bad thing, but the specs given allow the same variable to appear bound and free in the same expression, for example,
And (Exists ?X (Foo ?X) (Bar ?X)).
Scoping rules would give the result that ?X is bound in (Foo ?X) and free in (Bar ?X). In practice this is something that should be avioided.

```
Condition with a free variable ?X:
                              ... Exists ?Y (condition(..?X..?Y..))
...

Rule using the condition in its body:
Forall ?X (head(...?X...) :- ... Exists ?Y (condition(..?X..?Y..))
...)
```

When conditions are used as queries, their free variables are to be bound to carry the answer bindings back to the caller.

The semantics of conditions is defined in the section "SEMANTIC STRUCTURES".

```
Example 1 (A Herbrand RIF condition in human-readable syntax):

  In this condition, ?Buyer is quantified existentially, while
?Seller
  and ?Author are free:

  And ( Exists ?Buyer (purchase(?Buyer ?Seller book(?Author LeRif)
$49))
        ?Seller=?Author )
```

Comment [AG6]: I am not sure what this means.

This syntax is similar in style, and compatible to, the OWL Abstract Syntax http://www.w3.org/TR/owl-semantics/syntax.html.

The following XML syntax is for illustration purposes only. It can be obtained from the above BNF as shown below. The XML syntax is stripe-skipped in that it omits role (or property) elements since the Herbrand terms to be serialized have positional arguments and the positional information of the left-to-right ordered subelements of an XML element is unique. The only place in the Condition Language where role elements ('declare' and 'formula') are provided is within the

'Exists' element. Notice that Java's familiar case convention for distinguishing methods and classes is adopted, using lower-cased role elements and upper-cased class elements.

The non-terminals in all-uppercase such as CONDIT become XML entities, which act like macros and will not be visible in instance markups. The other non-terminals as well as symbols ('Exists' etc. as well as '=') become XML elements, which are adapted from RuleML as shown below.

```
- Con (constant individual, function, or relation)
- Var (logic variable, empty for anonymous variable)
- Expr   (expression formula)
- Atom   (atomic formula)
- Equal  (prefix version of term equation '=')
- Exists (quantified formula for 'Exists')
- declare (declare role, containing a Var)
- formula (formula role, containing a CONDIT formula)
- And    (conjunction)
- Or     (disjunction)
```

This can be directly rewritten as a DTD (adapting PositiveConditions.dtd) or an XML Schema.

Comment [AG7]: Needs to updated to be consistent with text.

The condition formula in Example 1 can be serialized in XML as shown below.

```
Example 2 (A Herbrand RIF condition in XML syntax):

  <And>
    <Exists>
      <declare><Var>Buyer</Var></declare>
      <formula>
        <Atom>
          <Con>purchase</Con>
          <Var>Buyer</Var>
          <Var>Seller</Var>
          <Expr>
            <Con>book</Con>
            <Var>Author</Var>
            <Con>LeRif</Con>
          </Expr>
          <Con>$49</Con>
        </Atom>
      </formula>
    </Exists>
    <Equal>
      <Var>Seller</Var>
      <Var>Author</Var>
    </Equal>
  </And>
```

Using the DTD spec, Richard Goerwitz' STG Validator succeeds with the conjunction of Example 2:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE And SYSTEM
"http://www.jdrew.org/rif/PositiveConditions.dtd">
<And>
  ... content of Example 2 ...
</And>
```

This XML version can be derived from a 'fully striped' version in the abstract syntax notation (cf. asn06), which permits metasyntax interoperation with OWL, RDF, etc.

SEMANTIC STRUCTURES (a.k.a. INTERPRETATIONS)

The first step in defining a model-theoretic semantics for a logic-based language is to define the notion of a *semantic structure*, also known as an *interpretation*, and then to define the notion of truth valuation for the formulas in the language.

Comment [AG8]: This is a little confusing, since below you equate a semantic structure with a mapping to truth values.

In case of the first-order semantics, the setting given here is one of the standard common definitions. Although it is not as frequently used as some other well-known definitions of semantic structures, it has the advantage of being easy to generalize to non-first-order cases --- for instance, rule sets with negation as failure (NAF), some of which (e.g., well-founded negation) use three-valued semantic structures, and settings, such as the Web, where information can be uncertain or contradictory. In the latter case, four-valued and other multi-valued semantic structures are used. (See, for example, M. Fitting, Fixpoint Semantics for Logic Programming A Survey, Theoretical Computer Science, 1999.)

Deleted: s

A semantic structure is a mapping of the form

I: Set of formulas → ***TV***

where ***TV*** is the set of all truth values. Thus, if Φ a formula then ***I***(Φ) is its truth value.

The set of truth values ***TV*** typically has only two values, **t** and **f**. However, some versions of NAF have three, **t**, **u** (undefined), and **f**, and, as we remarked, treatment of contradictions and uncertainty requires at least four: **t**, **u**, **f**, and **i** (inconsistent).

The set ***TV*** is assumed to have a total or partial order, called the *truth order*; it is denoted $<_t$. For instance, in the first-order case, **f** $<_t$ **t**, and it is a total order. In the well-founded semantics, **f** $<_t$ **u** $<_t$ **t**, and it is again a total order. But in Belnap-style four-valued logics, which are suitable for dealing with uncertain or inconsistent information, the truth order is partial: **f** $<_t$ **u** $<_t$ **t** and **f** $<_t$ **i** $<_t$ **t**.

As a side remark, Belnap-style logics also have another order, called the *knowledge order* $<_k$: **u** $<_k$ **t** $<_k$ **i**; and **u** $<_k$ **f** $<_k$ **i**. Under the knowledge order, true

and false are incomparable, and facts that are both true and false receive the truth value **i**, which is the least upper bound of **f** and **t** in the knowledge order.

Comment [AG9]: Why introduce this?

More formally, let us define the following sets:

- D - a non-empty set (of domain elements),
- Con - the set of syntax elements recognized by the Con / entity production,
- Var - the set of syntax elements recognized by the Var / ?name production

Comment [AG10]: see comment 2 above. I would just delete the word "domain" here, or put in an intrepretive remark like "D is the universe of discourse"

Comment [AG12]: See comment 3 above

Comment [AG11]: No such production exists in the BNF given above

An interpretation ***I*** consists of four mappings:

- I_C from Con to elements of D
- I_V from Var to elements of D
- I_F from Con to functions from D* into D (D* is a set of all tuples over domain D)
- I_R from Con to truth-valued mappings D* -> ***TV***

Using these mappings, we can define a more general mapping, *I*, as follows:

- $I(k) = I_C(k)$ if k is a constant
- $I(?v) = I_V(?v)$ if v is a variable
- $I(f(t1,...,tn)) = I_F(f)(I(t1),...,I(tn))$

Comment [AG13]: According to the syntax above, there shouldn't be any commas here. Should say "if 'f' is a constant followed by n terms"

As explained earlier, an interpretation is supposed to map formulas to truth values. We define this mapping now:

- Atomic formulas: $I(r(t1,...,tn)) = I_R(r)(I(t1),...,I(tn))$
- Equality: $I(t1=t2)$ = **t** iff $I(t1) = I(t2)$ and it is **f** otherwise.
- Conjunction: $I(And(c1,...,cn)) = min_t(I(c1),...,I(cn))$, where min_t is minimum with respect to the truth order.
- Disjunction: $I(Or(c1,...,cn)) = max_t(I(c1),...,I(cn))$, where max_t is maximum with respect to the truth order.
- Quantification: $I(\text{Exists v1 ... vn (c)}) = max_t(I^*(c))$, where max_t is taken over all interpretations I^* of the form $<I_C, I^*_V, I_F, I_R>$, where I^*_V is the same as I_V except possibly on the variables v1,...,vn (i.e., I^* agrees with I everywhere except possibly in its interpretation of the mappings of variables v1 ... vn).

Comment [AG14]: same as 13

Comment [AG15]: ditto

Comment [AG16]: ditto

Multisorted Extensions

The classical idea of sorted logic can easily account for the ideas of primitive data types, URIs as identifiers of objects and concepts, and more. Many logic languages (e.g., Prolog, HiLog, F-logic, RDF) allow the same symbol to play multiple roles. For instance, the same symbol *foo* can be used as a constant, a predicate of several different arities, and as a function symbol of different arities. To account for such languages, we will use a *multisorted* logic.

In a multisorted RIF core, each constant from Con is associated with one or more sorts. A sort can be ***primitive***, an ***arrow sort***, or a ***Boolean sort***. Arrow sorts are also known as *function* sorts and Boolean sorts are also known as *predicate* sorts.

Primitive sorts are drawn from a fixed collection of sorts PS_1, ..., PS_n. These sorts are intended to model primitive data types. For instance, we could have the sorts *integer*, *strings*, *time*, *dates*, etc. The same constant can be associated with more than one primitive sort, so it is possible that the sort of short integers will be a subsort of the sort of long integers (i.e., every constant that is associated with the sort *short* will also be associated with the sort *long*). It is a common practice to distinguish the constants of different primitive sorts syntactically. For instance, constants of the primitive sort *integer*, would have a different syntax from constants of sort *string*, and constants of primitive type URI would have yet another syntax.

Comment [AG17]: I find this a bit confusing. I would rewrite it, for example, as "Primitive sorts can form a logical heirarchy. For example the sort *short intergers* is a subsort of the sort *integers*. In that case any constant that is associate with the former sort will automatically be associate with the latter sort."

An ***arrow sort*** is a statement of the form $s_1 \times ... \times s_k \rightarrow s$, where s_1, ..., s_k, s are names of primitive sorts (i.e., one of the PS_1, ..., PS_n). A ***Boolean sort*** is a statement of the form $s_1 \times ... \times s_k$, where, again, s_1, ..., s_k are names of primitive sorts.

Comment [AG18]: In what language are these statements? And what is their meaning?

Recall that RIF core uses the symbols from Con to denote constants, predicates, and function symbols alike, so the same symbol can occur in multiple contexts. However, it is useful to restrict the contexts in which various symbols are allowed to occur. For instance, Prolog or RDF don't place any such restrictions, but OWL-DL has a unique role for each symbol. This restriction of the context is accomplished by controlling the sorts that are associated with each constant. For instance, if one doesn't want integers to occur as predicate and function symbols then we don't associate any arrow or Boolean sorts with the constants that are associated with primitive sort *integer*. On the other hand, we do want URIs to denote concepts and other predicates. In that case, we would associate every Boolean sort with every constant that has a primitive sort URI. If we want to also allow *local* names for concepts and other predicates, then we might introduce a separate primitive sort, *localPred*, and endow it with every Boolean type.

Multisorted Syntax

MULTISORTED SYNTAX OF PRIMITIVE SORTS

The human-readable syntax for primitive sorts (types) can use an infix operator between a term and its type. For example, reusing the `^^` infix of N3/Turtle, the term `6` can be given the type `#Perfect_number` by writing `6^^#Perfect_number`. A correspondingly typed variable `xyz` is written as `xyz^^#Perfect_number`.

The XML syntax can be obtained by using a 'type' attribute on XML term elements such as Con. Thus, the above example becomes `<Con`

`type="#Perfect_number">6</Con>`. A correspondingly typed variable `xyz` is written as `<Var type="#Perfect_number">xyz</Var>`.

Formalization of Multisorted Extensions

Formally, the syntax of RIF core needs the following adjustments. We introduce new functions:

- PSort: Con → powerset(Primitive_Sorts)
- ASort: Con → powerset(Arrow_Sorts)
- BSort: Con → powerset(Boolean_Sorts)

Each of these functions associates a (possibly empty) set of sorts (primitive, arrow, or Boolean) with every constant c ∈ Con.

PSort is also defined on variables:

PSort: Var → powerset(Primitive_Sorts)

The intended meaning is that if ?v ∈ Var and PSort(?v) = {s_1, ..., s_k} then ?v can be bound only to function terms that are simultaneously of sorts s_1, ..., s_k (we define what it means for a function term to belong to a primitive sort below). In theory, PSort(?v) can be an empty set. However, such a variable would be useless, since it cannot be bound to anything.

Well-formed function terms. If c ∈ Con is a constant and s ∈ PSort(c) then we say that c (and c(), which we identify with c) is a ***well-formed function term of sort s***. Note that the same constant can be a well-formed term of several different sorts because we allow several primitive sorts to be associated with the same constant. The informal meaning of such a happenstance is that the term belongs to the "intersection" of all the sorts with which it is associated.

By induction, if f(t_1, ..., t_k) is a function term then it is a ***well-formed function term of sort s*** if there is an arrow sort s_1, ..., s_k → s ∈ ASort(f) such that t_1, ..., t_k are well-formed function terms of sorts s_1, ..., s_k, respectively.

Comment [AG19]: commas again

It is convenient to extend the mapping PSort from constants to function terms as follows:

- PSort(t) = { s | t is a well-formed term of sort s }

Well-formed atomic formula. We can now say that an atomic formula $p(t_1, ..., t_k)$ is ***well-formed*** if and only if $t_1, ..., t_k$ are well-formed function terms and there is a Boolean sort $s_1 \times ... \times s_k \in BSort(p)$ such that $s_1 \in PSort(t_1), ..., s_k \in PSort(t_k)$.

The only other modification to the definition of the RIF syntax is that we must require that all atomic formulas that occur in RIF conditions and rules must be well-formed.

Semantics of the Multisorted RIF Core

The semantics of RIF core needs the following adjustments in order to be compatible with the multisorted syntax:

- The domain D of an interpretation is now split into several subdomains:
  - $D = D_{s1} \cup ... \cup D_{sn}$, where each D_{si} is the domain of interpretation of the primitive sort s_i.
- If $c \in Con$ or $?v \in Var$ is a constant or a variable of
  - primitive sort s then $I_C(c) \in D_s$ and $I_V(?v) \in D_s$.
- If f has an arrow type $s_1, ..., s_k \rightarrow s \in ASort(f)$ then
  - $I_F(f)$ should be a (possibly polymorphic) function of type $D_{s1} ... D_{sk} \rightarrow D_s$, i.e., if $d_1 \in D_{s1}, ..., d_k \in D_{sk}$ then $I_F(d_1,...,d_k)$ must be in D_{sk} (if the arguments are not in $D_{s1} ... D_{sk}$ then the result does not need to be in D_s, but $I_F(f)$ might have other types, which restrict its behavior).
- The definition of I_P requires no adjustments.

Comment [AG20]: Shouldn't this be D_s ?

Comment [AG21]: Shouldn't this be I_R ?

2. RIF Rule Language

(Editor's Note: This text is maintained on wiki page RIF Rule Language).

This proposal develops a RIF Rule Language as an extension of the RIF Condition Language, where conditions become rule bodies. The Rule Language starts with rules having Horn logic expressiveness (positive conditions) and then proceeds to increased expressiveness, mainly by reusing generalized conditions of the Condition Language as rule conditions.

2.1. Horn Rules

(Editor's Note: This text is maintained on wiki page Horn Rules).

Based on RIF's positive conditions, this section defines Horn rules for RIF (Phase 1).

SYNTAX

The following BNF syntax is for illustration/explanation purposes only. To specify Horn clauses we just need to add these productions to the human-readable syntax of Positive Conditions:

```
  . . .
  HEAD     ::= LITFORM
  BODY     ::= CONDIT
  Implies  ::= HEAD ':-' BODY
  CLAUSE   ::= 'Forall' Var* '(' Implies ')' | 'Forall' Var* '('
HEAD ')'
```

where `CONDIT` and `LITFORM` are defined in Positive Conditions, and `:-` (pronounced 'IF') is an implication connective.

Rules are generated by the `Implies` production. Facts are generated by the `HEAD` production, where such a factual HEAD is regarded as a shorthand for a rule with an empty, hence true, conjunctive BODY condition of the form

```
Implies  ::= HEAD ':-' And '(' ')'
```

Finally, a `CLAUSE` generates a universally closed rule or fact.

By 'inheritance' from Positive Conditions through `CONDIT` and `LITFORM`, all Expressions and Atoms in Horn rules have a Herbrand (positional) form.

Note also that, since `CONDIT` permits disjunction and existential quantification, the rules defined by the `Implies` production look more general than Horn. However, it is well-known that such extended rules reduce to Horn via a simple syntactic transformation.

The document RIF Use Cases and Requirements describes the use case "Negotiating eBusiness Contracts Across Rule Platforms", containing this first rule proposed in a hypothetical negotiation by some agent John:

```
If an item is perishable and it is delivered more than 10 days
after the scheduled delivery date
then the item will be rejected.
```

This can be formalized in equivalent human-readable syntax as the following three ways: 1) *assuming* the universal closure when generated directly by the `Implies` production (Example 3a), 2) *showing* the universal closure when generated through the `CLAUSE` production (Example 3b), and 3) *transforming* BODY-only variables to existentials as introduced in the language of Positive Conditions (Example 3c):

Deleted: equivalently

Deleted: s

Deleted: ,

```
Example 3a (A RIF rule in human-readable syntax using implicit
quantification):
```

```
  In this rule, the HEAD is a user-defined relation of two
arguments, one a constant.
  The BODY is a conjunction of five Atoms, the first three user-
defined relations, the
  fourth and fifth user-defined or built-in relations.

  reject(John ?item) :-
    And ( perishable(?item)
          delivered(?item ?deliverydate)
          scheduled(?item ?scheduledate)
          timediff(?diffdate ?deliverydate ?scheduledate)
          greaterThan(?diffdate 10) )
```

Example 3b (A RIF rule in human-readable syntax using explicit top-level quantification):

```
  In this rule, the HEAD is a user-defined relation of two
arguments, one a constant.
  The BODY is a conjunction of five Atoms, the first three user-
defined relations, the
  fourth and fifth user-defined or built-in relations.

  Forall ?item ?deliverydate ?scheduledate ?diffdate
        (
          reject(John ?item) :-
             And ( perishable(?item)
                   delivered(?item ?deliverydate)
                   scheduled(?item ?scheduledate)
                   timediff(?diffdate ?deliverydate ?scheduledate)
                   greaterThan(?diffdate 10) )
        )
```

Example 3c (A RIF rule in human-readable syntax using existential BODY quantification):

```
  In this rule, the HEAD is a user-defined relation of two
arguments, one a constant.
  The BODY is a conjunction of five Atoms, the first three user-
defined relations, the
  fourth and fifth user-defined or built-in relations.

  Forall ?item
        (
          reject(John ?item) :-
             Exists ?deliverydate ?scheduledate ?diffdate
                    (
                      And ( perishable(?item)
                            delivered(?item ?deliverydate)
                            scheduled(?item ?scheduledate)
                            timediff(?diffdate ?deliverydate
?scheduledate)
                            greaterThan(?diffdate 10) )
                    )
        )
```

The following XML syntax is for illustration purposes only. It can be obtained from the above BNF and from Positive Conditions as shown below. The 'Forall'

element uses the role elements ('declare' and 'formula') introduced for the 'Exists' element in the Condition Language. The 'Implies' element uses role elements ('head' and 'body') to get rid of any order information, permitting serializations in both the consequent-first style of logic programming and the antecedent-first style of production rules.

```
  . . .
- head (consequent role, containing LITFORM)
- body (antecedent role, containing CONDIT)
- Implies (implication, containing head and body roles in any
order)
- Forall (quantified CLAUSE formula with 'Forall')
```

This can be directly rewritten as a DTD (adapting HornRules.dtd) or an XML Schema.

Comment [AG22]: This DTD also needs to be updated to be in accord with the text.

The rule in Example 3b can be serialized in XML, e.g. again having the head first, as shown below (since serializations are to be explicit, no direct serialization of Example 3a is offered, while the serialization of Example 3c is offered but not shown here).

```
Example 4 (A RIF rule in XML syntax using consequent-first
serialization):

<Forall>
  <declare><Var>item</Var></declare>
  <declare><Var>deliverydate</Var></declare>
  <declare><Var>scheduledate</Var></declare>
  <declare><Var>diffdate</Var></declare>
  <formula>
    <Implies>
      <head>
        <Atom>
          <Con>reject</Con>
          <Con>John</Con>
          <Var>item</Var>
        </Atom>
      </head>
      <body>
        <And>
          <Atom>
            <Con>perishable</Con>
            <Var>item</Var>
          </Atom>
          <Atom>
            <Con>delivered</Con>
            <Var>item</Var>
            <Var>deliverydate</Var>
          </Atom>
          <Atom>
            <Con>scheduled</Con>
            <Var>item</Var>
            <Var>scheduledate</Var>
```

```
            </Atom>
            <Atom>
              <Con>timediff</Con>
              <Var>diffdate</Var>
              <Var>deliverydate</Var>
              <Var>scheduledate</Var>
            </Atom>
            <Atom>
              <Con>greaterThan</Con>
              <Var>diffdate</Var>
              <Con>10</Con>
            </Atom>
          </And>
        </body>
      </Implies>
    </formula>
  </Forall>
```

Using the DTD spec, Richard Goerwitz' STG Validator succeeds with the Horn clause of Example 4:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE Forall SYSTEM "http://www.jdrew.org/rif/HornRules.dtd">
<Forall>
  ... content of Example 4 ...
</Forall>
```

SEMANTICS

Interpretation and Models of Rules

In Positive Conditions we defined the notion of *semantic structures* and what it means for such a structure to satisfy a RIF condition. Here we extend this notion and define what it means for such a structure to satisfy a rule.

While semantic structures can be multivalued, rules are typically two-valued even in logics that support inconsistency and uncertainty. Consider a rule of the form **Q** *head* :- *body*, where **Q** is a quantification prefix for all the variables in the rule. For the Horn subset, **Q** is a universal prefix, i.e., all variables in the rule are universally quantified outside of the rule. We first define the notion of rule satisfaction without the quantification prefix **Q**:

I |= *head* :- *body*

iff ***I***(*head*) $\geq$ ***I***(*body*).

We define ***I*** |= **Q** *head* :- *body* iff ***I**** |= *head* :- *body* for every ***I**** that agrees with ***I*** everywhere except possibly on some variables mentioned in **Q**. In this case we also say that *I* is a ***model*** of the rule. ***I*** is a ***model of a rule set*** **R** if it is a model

of every rule in the set, i.e., if it is a semantic structure such that ***I*** |= r for every rule r **R**.

Intended Models of Rules

The notion of a model is only the basic ingredient in the definition of a semantics of a rule set. In general, a ***semantics of a rule set*** **R** is the set of its ***intended models*** (see Y. Shoham. Nonmonotonic logics: meaning and utility. In: Proc. 10th International Joint Conference on Artificial Intelligence, Morgan Kaufmann, pp. 388--393, 1987). There are different theories of what the intended sets of models are supposed to look like depending on the features of the different rule sets.

For Horn rules, which we use in this section, the intended set of models of **R** is commonly agreed upon: it is the set of all models of **R**.

Deleted: agreed

However, when rules contain negation-as-failure (naf) literals in the body, only some of the models of a rule set are accepted as intended. This issue will be addressed in future extensions of RIF. The two most common theories of intended models are based on the so called ***well-founded models*** and ***stable models***. Here we will just illustrate the problem with an example.

Suppose **R** has a single rule `p:-naf q`. If naf is interpreted as classical negation, *not*, then this rule is simply `p\/q`, and so it has two kinds of models: one in which `p` is true and one where `q` is true. In contrast, most current rule-based systems do not consider `p` and `q` symmetrically. Instead, they view this as a specification of an intent that p must be true if it is not possible to establish the truth of `q`. Since it is, indeed, impossible to establish the truth of `q`, such theories will derive `p` even though it does not logically follow from `p:-not q`. The logic underlying rule-based systems also considers that only the *minimal* models as intended (minimality here means minimization of the set of true facts). Therefore, the intended models of our rule set must have the property that not only `p` is true but also that `q` is false.

Formatted: Font: Italic

3. RIF Compatibility

(Editor's Note: This text is maintained on wiki page RIF Compatibility).

The compatibility of RIF Core is currently focussed on the Semantic Web standards OWL and RDF, as explained in RIF-OWL Compatibility and RIF-RDF Compatibility.

3.1. RIF-OWL Compatibility

(Editor's Note: This text is maintained on wiki page RIF-OWL Compatibility).

RIF-OWL Compatibility will be described here on the basis of OWL Compatibility, A.5 Extension: Ontology Conditions, and (email) discussions.

3.2. RIF-RDF Compatibility

(Editor's Note: This text is maintained on wiki page RIF-RDF Compatibility).

RIF-RDF Compatibility will be described here on the basis of RDF Compatibility, A.4 Extension: Resource Conditions, and (email) discussions.