W3C Working Group Note

RIF Primer (Second Edition)

W3C Working Group Note 5 February 2013

This version:
http://www.w3.org/TR/2013/NOTE-rif-primer-20130205/
Latest version:
http://www.w3.org/TR/rif-primer/
Previous version:
http://www.w3.org/TR/2012/NOTE-rif-primer-20121211/
Editors:
Leora Morgenstern, SAIC
Chris Welty, IBM Research
Harold Boley, National Research Council Canada
Gary Hallmark, Oracle

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

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

Abstract

This document is a primer on the Rule Interchange Format (RIF). The primer provides a practical introduction to specifying declarative rules and production rules in RIF, in particular for the RIF BLD and PRD dialects. Examples of RIF specifications are developed in a stepwise manner.

Status of this Document

May Be Superseded

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

Set of Documents

This document is being published as one of a set of 13 documents:

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

W3C Working Group Note

Document Unchanged

There have been no changes to the body of this document since the previous version. For details on earlier changes, see the change log.

Please Send Comments

Please send any comments to public-rif-comments@w3.org (public archive). Although work on this document by the Rule Interchange Format (RIF) Working Group is complete, comments may be addressed in the errata or in future revisions. Open discussion among developers is welcome at public-rif-dev@w3.org (public archive).

No Endorsement

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

Patents

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

Table of Contents

1 Introduction

1.1 What is a rule?

This document is meant to introduce computer professionals to basic techniques for writing declarative rules and production rules in the W3C Rule Interchange Format.

The dictionary defines a rule as a prescribed guide for conduct or a regulation governing behavior such as "Keep off the grass" or "A driver receiving a traffic ticket must appear in traffic court on the assigned date." In the computer science and logic programming communities, though, there are two different, but closely related ways to understand rules. One is closely related to the idea of an instruction in a computer program: If a certain condition holds, then some action is carried out. Such rules are often referred to as *production rules*. An example of a production rule is "If a customer has flown more than 100,000 miles, then upgrade him to Gold Member status."

Alternately, one can think of a rule as stating a fact about the world. These rules, often referred to as *declarative rules*, are understood to be sentences of the form "If P, then Q." An example of a declarative rule is "If a person is currently president of the United States of America, then his or her current residence is the White House."

Declarative rules do not specify an action that is to be carried out. For example, the rule above makes a statement about the relation between the U.S. president and the White House, but doesn't specify any action (say, to move the president into the White House). In other words, a declarative rule describes how the world is, rather than prescribing how things ought to be.

Production rules are frequently used for business applications. They typically incorporate an explicit notion of control to specify which rules are applied first. Declarative rules, on the other hand, are useful for writing down large amounts of information about a particular domain, or area of knowledge, independent of knowing, 'a priori', how these rules will eventually be used.

For example, one could write down rules in the medical domain:

- *If John's rapid-strep test is positive, then he is infected with strep.*
- *If John is infected with strep, then he needs to take antibiotics.*

Rules (whether declarative or production rules) can also be used to reason with other information that we have. In the example above, we can infer another rule:

- *If John's rapid-strep test is positive, then he needs to take antibiotics.*

It is particularly useful to reason with rules and facts, pieces of concrete information that do not make use of the if-then construct. Examples of facts are:

- *Elizabeth II became Queen of England in 1952.*
- *The Bronx is a borough of New York City.*
- *John's rapid-strep test is positive.*

To continue the example above, the fact

- *John's rapid-strep test is positive*

can be combined with the rules above to yield the conclusion:

- *John needs to take antibiotics.*

Rules and facts must be written in some sort of formal language in order for computer programs to reason with and draw conclusions from them. Such a computer program is often called a *rule engine*.

1.2 What is RIF?

There are many rule languages including SILK [GR09], OntoBroker [DEFS99], Eye [EULERSHARP10], VampirePrime [RZ09], N3-Logic [BCKSH08], and SWRL [SWRL-Ref] (declarative rule languages); and Jess [FH03], Drools [BA09], IBM ILog [ILOG10], and Oracle Business Rules [ORACLE10] (production rule languages). Many languages incorporate features of both declarative and production rule language. For example, Prolog [CM03], which is generally considered to be a declarative rule language, provides a cut operator for controlling the application of rules. Moreover, all production rule languages have a core subset that is declarative.

The abundance of rule sets in different languages can create difficulties if one wants to integrate rule sets, or import information from one rule set to another. How can a rule engine work with rule sets of different languages?

The W3C Rule Interchange Format (RIF) [RIF-Overview] is a standard that was developed to facilitate ruleset integration and synthesis. It comprises a set of interconnected dialects representing rule languages with various features. This document focus on the most basic RIF language, RIF-Core [RIF-Core], augmented by a set of datatypes and built-in functions and predicates that can be used when writing rules [RIF-DTB]. All RIF dialects are an extension of RIF-Core plus DTB; we focus on two dialects, BLD and PRD.

1.3 What this document does and doesn't cover

This document focuses on the Basic Logic Dialect [RIF-BLD], and the Production Rule dialect [PRD], both of which are extensions of RIF-Core plus DTB. [RIF-FLD] is not discussed in this document.

This document does not include a detailed discussion of the semantics of any RIF dialect In particular, it does not discuss the model-theoretic semantics of BLD or the operational semantics of PRD. However, the document discusses the notions of assumption, consequence, and pattern matching. An intuitive understanding of these notions should enable a computer professional to understand the Primer and to write rules in RIF. Reading the BLD document [BLD] and the PRD document [PRD] is recommended for those who wish to learn more about, respectively, the model-theoretic and operational semantics used in RIF.

While this Primer is targeted at getting computer professionals to quickly learn how to write rules in RIF, it does not provide a complete specification of syntax. There are some details of RIF syntax, specifically those that are not necessary for writing most rules in RIF, which are not covered in this document. All syntactic details of RIF's logic dialects are covered in BLD, PRD, DTB, and FLD.

1.4 A note on syntax

The standard syntax for RIF is a verbose XML syntax, designed so that programs can easily generate and parse it. For human readers and writers, we generally use terser syntaxes that have a simple 1-1 correspondence with the XML. Different dialects have introduced different compact syntaxes: for example, BLD uses a compact syntax called Presentation Syntax, while PRD uses a compact syntax called Abstract Syntax.

One of the major stylistic differences between Presentation Syntax and Abstract Syntax is the way in which implications are written. In Presentation Syntax, an implication *If A then B* is written as

```
B :- A
```

while in Abstract Syntax, this implication is written as

```
If A Then B
```

We will use this syntax style, referred to as Mixed Presentation Syntax (MPS) whether presenting examples of BLD or PRD.

2 A Simple Example in RIF

2.1 Example Background

The RIF examples in this document concern the integration of data about films and plays across the Semantic Web. Suppose, for example, that one wants to combine data about films from IMDb, the Internet Movie Data Base (at http://imdb.com) with DBpedia (at http://dbpedia.org). Both resources contain facts about actors being in the cast of films, but DBpedia expresses these facts as a binary relation (aka predicate or RDF property).

In DBpedia, for example, one can express the fact that an actor is in the cast of a film:

- `starring(?Film ?Actor)`

where we use '?'-prefixed variables as placeholders. The names of the variables used in this example are meaningful to human readers, but not to a machine. These variable names are intended to convey to readers that the first argument of the DBpedia `starring` relation is a film, and the second an actor who stars in the film. (Variables are discussed in more detail in Section 2.2.2.)

In IMDb, however, one does not have an analogous relation. Rather, one can state facts of the following form about actors playing roles:

- `playsRole(?Actor ?Role)`

and one can state facts of the following form about roles (characters) being in films:

- `roleInFilm(?Role ?Film)`

Thus, for example, in DBpedia, one represents the information that Vivien Leigh was in the cast of *A Streetcar Named Desire*, as a fact

- `starring(Streetcar VivienLeigh)`

In IMDb, however, one represents two pieces of factual information, that Vivien Leigh played the role of Blanche DuBois:

- `playsRole(VivienLeigh BlancheDubois)`

and that Blanche DuBois was a character in *A Streetcar Named Desire*:

- `roleInFilm(BlancheDubois Streetcar)`

2.2 Developing the Example

The challenge in combining this data should be obvious: not only do the two data sources (IMDb and DBpedia) use different *vocabulary* (the relation names *starring, playsRole, roleInFilm*), but the *structure* is different. To combine this data, we essentially want to say something like the following:

If there are two facts in the IMDb database, saying that an actor plays a role/character, and that the character is in a film, then there is a single fact in the DBpedia database, saying that the actor is in the film.

This will be referred to as the *Basic Combination Rule*. We develop the Basic Combination Rule as a RIF rule in a stepwise manner in sections 2.2.1 through 2.3.

In this document, we incrementally introduce elements of RIF syntax and semantics, eventually building up to an example of valid RIF syntax. Often --- as is the case in the text boxes below, in sections 2.2.1 through 2.2.6 --- a single element or set of elements, that by itself is not a valid RIF rule set, is introduced. The red-tinted background in which these preliminary examples are displayed indicates that they are not fully valid RIF.

2.2.1 Atomic Formulas: Basic Facts

To prepare the Basic Combination Rule example, we first show how to write atomic formulas (atoms) in RIF.

Atoms can be formed from predicates applied to zero, one, two, or more arguments. Atoms can be used to state facts. For example:

- the nullary (0-argument) predicate application `itRains()` could stand for the proposition *it rains*.
- the unary (1-argument) predicate application `rapidStrepPos(John)` could stand for the proposition *John's rapid-strep test is positive*.
- the binary (2-argument) predicate application `playsRole(VivienLeigh BlancheDubois)` could stand for the proposition *Vivien Leigh played the role of Blanche DuBois*.

2.2.2 Constants and Variables

Like the atoms that make up molecules in the material world, logical atoms are not truly indivisible entities: they are, rather, composed of parts. The basic parts of an atom are *predicates* and *constants* (which are closely related, as discussed below), as well as *variables* and some other syntax, such as parentheses.

A constant is a term, or a symbol, used to refer to some specific (real or imagined) individual in the world (e.g. the constant `BlancheDubois` may refer to the character Blanche Dubois), or to some specific set of individuals (e.g. the constant `Actors` may refer to the set of all actors) or to some specific set of related individuals (e.g. the constant `starring` can refer to the set of all pairs of <x,y> where x stands for an actor, y stands for a film, and the starring relation holds between the individuals in each pair; the constant `producedFilm` can refer to the set of all triples <x,y,x> where x stands for a film producer, y stands for the film, and z stands for the year in which the film was produced). A set of related individuals, more commonly known as a set of ordered tuples, is known as a *predicate*.

Constant symbols mean nothing to machines; it is up to the humans who write and implement rules to interpret them in ways that make sense. As discussed below, one can write rules to help ensure the machine doesn't use the symbols in ways that violate their intended meaning.

A variable is a symbol, prefixed by a ?, that does not refer to anything specific in itself, but rather serves as a placeholder for writing general rules that range over larger sets of individuals. Variables in rules are similar, in this sense, to variables in computer programs. For example, the variable ?Actor could stand for any actor or actress.

As is the case with constants, the names of variables mean nothing to the machine, except to distinguish one variable from another. One may name the variable ?Actor in order to convey to human readers that the variable is intended to be a placeholder for Actors, but this does not yet convey that information to the machine.

2.2.3 Conjunctions and Implications

The general structure of the Basic Combination Rule is:

- *If firstatom and secondatom, then thirdatom*.

The rule contains a *conjunction*, a formula of the form *A and B* or, generally, *A1 and A2 and .... and An*.

In RIF, a conjunction is rewritten in prefix notation, e.g.

- the binary *A and B* is written as `And(A B)`.

Generally,

- the n-ary *A1 and A2 and ... and An* is written as `And(A1 A2 .... An)`.

The Basic Combination Rule also contains an *implication*, a statement of the form *if A then B*.

This implication is written almost unchanged in our notation, Mixed Presentation Syntax, as

- `If A Then B`.

Thus, one would write an implication of the form *If firstatom and secondatom, Then thirdatom* as

```
If And(firstatom secondatom) Then thirdatom
```

or in a pretty-print format with indentation indicating levels of a formula:

```
If   And(firstatom secondatom)
Then thirdatom
```

Note that in contrast, in RIF Presentation Syntax (PS) --- used, e.g., in [RIF-BLD] --- this is written in infix notation as B `:-` A, where the antecedent A and consequent B are reversed. The implication expresses the exact same meaning.

2.2.4 Quantifiers

The syntax that we have introduced so far allows us to write a rule that says:

*If IMDb contains the facts that **Vivien Leigh** played the role of **Blanche Dubois**, and that **Blanche Dubois** is a character role in **A Streetcar Named Desire**, then conclude the DBpedia fact that **Vivien Leigh** acted in **A Streetcar Named Desire** .*

This rule could be represented as

```
If   And(playsRole(VivienLeigh BlancheDubois)
         roleInFilm(BlancheDubois Streetcar))
Then starring(Streetcar VivienLeigh)
```

(Note how indentation is used to facilitate reading. **Bold-facing** is only used in this example to emphasize the correspondence between individual constants in the English and RIF versions of the rule.)

But this doesn't, of course, represent the Basic Combination Rule. The Basic Combination Rule says something about *all* actors, *all* roles, and *all* films.

To express this, we need to use variables with *quantifiers*. There are two sorts of quantifiers, *Forall*, known as the *universal quantifier*, and *Exists*, known as the *existential quantifier*. The universal and existential quantifiers can be used to form facts and rules.

For example, to say that all people like the film Casablanca, one can say:

```
Forall ?Person (likesFilm (?Person Casablanca) )
```

Note that while this is technically a rule, it is not a valid RIF rule yet.

To say that at least one person likes the film Casablanca, one can say (this, too, is not a valid RIF rule):

```
Exists ?Person (likesFilm (?Person Casablanca) )
```

Now it is possible to write a version of the Basic Combination Rule:

```
 Forall ?Actor ?Film ?Role (
  If And(playsRole(?Actor ?Role) roleInfilm(?Role ?Film))
  Then starring(?Film ?Actor)
)
```

There are still a few more steps needed to make this a valid RIF rule, as will be discussed in the next section.

Note also that in Presentation Syntax, this would read as:

```
Forall ?Actor ?Film ?Role (
  starring(?Film ?Actor) :-
    And(playsRole(?Actor ?Role) roleInFilm(?Role ?Film))
)
```

2.2.5 IRIs (Internationalized Resource Identifiers)

The formula shown in BCR-v0.1 is still not a correct formula in RIF. Individual constants like *VivienLeigh* and predicate constants like *playsRole* cannot be just used 'as is' but need to be disambiguated. This

disambiguation addresses the issue that the constants used in this rule come from more than one database and may have different meanings --- that is, refer to different entities --- in each.

In RIF, disambiguation is effected using IRIs, Internationalized Resource Identifiers. (IRIs are a generalization of the concept of URIs; the primary distinction between the two is that IRIs allow characters from more alphabets than do URIs.) As with URIs, an IRI is a web address that typically includes some information about where the constant comes from (e.g. http://www.imdb.com/constants or http://dbpedia.org/resource).

Since using the full IRI notation can be cumbersome for every constant, RIF Mixed Presentation Syntax and other compact syntaxes of RIF allow an abbreviated form through namespace declarations. The specification of abbreviated form is explained in full in [RIF-BLD]. The general form of a prefix declaration can be quite complex. The basic idea is taht one can declare a namespace *ns* stands for the concept or entity described by the IRI *thisIRI* by writing the prefix declaration *Prefix(ns<ThisIRI>)*. Then the constant *name* can be disambiguated in rules using the string *ns:name*.

Consider the following example, in which, as is standard, http://example.com is an IRI reserved to demonstrate examples. Assume we are given the following declarations:

```
Prefix(imdbrel <http://example.com/imdbrelations#>)
Prefix(dbpedia <http://dbpedia.org/ontology/>)
```

The constant `imdbrel:playsRole` would be interpreted by RIF as the entity referred to by `http://example.com/imdbrelations#playsRole`, and the constant `dbpedia:starring` would be interpreted as the entity referred to by `http://dbpedia.org/ontology/starring`.

Using these prefixes, we can write our BCR with URIs for constants as follows:

```
Forall ?Actor ?Film ?Role (
  If   And(imdbrel:playsRole(?Actor ?Role) imdbrel:roleInFilm(?Role ?Film))
  Then dbpedia:starring(?Film ?Actor)
)
```

(Note that in Presentation Syntax, this would read as:)

```
Forall ?Actor ?Film ?Role (
  dbpedia:starring(?Film ?Actor) :-
    And(imdbrel:playsRole(?Actor ?Role) imdbrel:roleInFilm(?Role ?Film))
)
```

Namespace prefixes are not strictly required; however, they can greatly improve the readability of rules. Consider, for example, how the rule above would look like without them:

```
Forall ?Actor ?Film ?Role (
  If   And(<http://example.com/imdbrelations#playsRole>(?Actor ?Role)
           <http://example.com/imdbrelations#roleInFilm>(?Role ?Film))
  Then <http://dbpedia.org/ontology/starring>(?Film ?Actor)
)
```

Note that our examples v0.2 and v0.2.1 are now valid RIF syntax. However, they are not yet complete RIF documents, and are therefore still shown in red.

2.2.6 Structuring Operators

Two more operators, *Group* and *Document*, are needed to write rules in RIF. *Group* is used to delimit, or group together, a set of rules within a RIF document. It is semantically equivalent to the operation of conjunction (the *And* operator). That is, a set of rules *Rule1* and *Rule2* can be written as

```
Group(
  Rule1
  Rule2
)
```

This is semantically equivalent to writing *And(Rule1 Rule2)*.

A document may contain many groups or just one group. Similarly, a group can consist of a single rule, although they are generally intended to group multiple rules together. For example, if there were multiple rules that concluded the same predicate, they might be put in the same group to help keep them organized. This organization is more for the sake of the humans who write, read, and use these rules than for the software systems that process them. A software system that processes RIF does not care whether every group has only one rule, or if all rules are in one group.

It is necessary to have an explicit *Document* operator because a RIF document can import other documents and can thus itself be a multi-document object. For practical purposes, it is sufficient to know that the *Document* operator is generally used at the beginning of a document, followed by a prefix declaration and one or more groups of rules.

2.3 Putting the pieces together: example rule in RIF

Using the elements introduced in Sections 2.2.1 --- 2.2.6, it is now possible to write the complete Basic Combination Rule in RIF Core:

```
Document(
  Prefix(rdfs <http://www.w3.org/2000/01/rdf-schema#>)
  Prefix(imdbrel <http://example.com/imdbrelations#>)
  Prefix(dbpedia <http://dbpedia.org/ontology/>)

  Group(
    Forall ?Actor ?Film ?Role (
      If   And(imdbrel:playsRole(?Actor ?Role) imdbrel:roleInFilm(?Role ?Film))
      Then dbpedia:starring(?Film ?Actor)
    )
  )
)
```

As has been demonstrated in these sections, it is natural to develop RIF rule documents from the inside out, focusing first on the logical structure of the rules; next, grounding all terms using IRIs; and finally, using structuring operators to create a RIF document.

In this document, text boxes with a gray background, like those in the above example, are fully valid RIF documents in RIF Mixed Presentation Syntax.

3 Two Additional Examples

3.1 Second Example: Disjunctions, Existentials, Overloading Predicate Names

Much of the previous discussion is relevant to the second example in the DBpedia domain. Suppose one wants to say that an actor can be described as an award-winner in DBpedia if he has won an award either for acting in a major film or a Broadway play. Such a rule could make reference to the data in IBDB, the Internet Broadway Database.

Using the syntax developed in the previous section, plus a few additions that will be explained below, the rule could be written as:

```
Document(
  Prefix(rdfs <http://www.w3.org/2000/01/rdf-schema#>)
  Prefix(imdbrelf <http://example.com/fauximdbrelations#>)
  Prefix(dbpediaf <http://example.com/fauxdbpediarelations>)
  Prefix(ibdbrelf <http://example.com/fauxibdbrelations#>)

  Group(
    Forall ?Actor (
```

```
        If   Or(Exists ?Film (imdbrelf:winAward(?Actor ?Film))
                Exists ?Play (ibdbrelf:winAward(?Actor ?Play)) )
        Then dbpediaf:awardWinner(?Actor)
      )
    )
  )
```

Most of the syntax used in this rule --- e.g., the *Document* and *Group* operators, the conditionals --- has been introduced in the previous section. This example introduces the disjunction operator *Or*.

A disjunction is a formula of the form *A or B* or more generally, *A1 or A2 or ... or* An.

In RIF MPS,

- A disjunction *A or B* is written as `Or(A B)`.

More generally

- a disjunction *A1 or A2 or ... or An* can be written as `Or(A1 A2 .... An)`.

Note that two different instances of the *winAward* predicate are used in this rule: the *winAward* predicate that would be used by IMDb and the *winAward* predicate that would be used by IBDB. While these are both binary predicates in this example, it could also be the case that one of these predicates has greater arity (that is, has more arguments, or variables) than another.

Note also that this is a fabricated example, in these sense that there is no relation *awardWinner* used by DBpedia. For that reason, we use the domain http://example.com which is used for hypothetical examples, rather than the actual DBpedia website, http://dbpedia.org, and make clear from the prefix declarations that these relations are fabricated.

3.2 Third Example: Using Guards

As discussed above, the use of `?Actor` as the name of a variable means nothing to the rule engine. Rather, such meaningful names --- that is, meaningful to a human --- are used to make rules more readable. However, there are various strategies, used by many linked data sources, to convey information about meaning to a rule engine.

For example, in RDF, a special property `rdf:type` is reserved to represent the relation between an individual and a class: a string of the general form `rdf:type (a B)` states that individual a is an instance of class B. For example `rdf:type(VivienLeigh Actor)` states that the individual `VivenLeigh` is an instance of the class `Actor`.

By convention, names of individuals and properties in linked data sources usually start with a lowercase letter; while names of classes start with an uppercase letter. If a data source has such information, we can make use of it in rules to enforce that, for example, the `?Actor` variable only has values that are instances of the class *Actor*. These kinds of restrictions on the values of variables are called *guards*.

For this example, we will extend the Basic Combination Rule example with guards on the values of the variables.

```
Document(
  Prefix(rdf <http://www.w3.org/1999/02/22-rdf-syntax-ns#>)
  Prefix(rdfs <http://www.w3.org/2000/01/rdf-schema#>)
  Prefix(imdbrel <http://example.com/imdbrelations#>)
  Prefix(dbpedia <http://dbpedia.org/ontology>)

  Group(
    Forall ?Actor ?Film ?Role (
      If   And(rdf:type(?Actor imdbrel:Actor)
               rdf:type(?Film imdbrel:Film)
               rdf:type(?Role imdbrel:Character)
               imdbrel:playsRole(?Actor ?Role)
               imdbrel:roleInilm(?Role ?Film))
      Then dbpedia:starring(?Film ?Actor)
```

```
      )
    )
  )
```

Guards are so common that RIF introduces a special syntax for the rdf:type predicate, denoted in the Mixed Presentation Syntax by the binary operator #. With this, the rule becomes slightly more concise:

```
Document(
  Prefix(rdf <http://www.w3.org/1999/02/22-rdf-syntax-ns#>)
  Prefix(rdfs <http://www.w3.org/2000/01/rdf-schema#>)
  Prefix(imdbrel <http://example.com/imdbrelations#>)
  Prefix(dbpedia <http://dbpedia.org/ontology/>)

  Group(
    Forall ?Actor ?Film ?Role (
      If   And(?Actor # imdbrel:Actor
               ?Film # imdbrel:Film
               ?Role # imdbrel:Character
               imdbrel:playsRole(?Actor ?Role)
               imdbrel:roleInFilm(?Role ?Film))
      Then dbpedia:starring(?Film ?Actor)
    )
  )
)
```

4 Reasoning in RIF

4.1 Using RIF Rulesets

The examples thus far demonstrate how to write certain types of rules in RIF. So far, however, we have not demonstrated how and in what circumstances these rules can be used. Intuitively, we know that rules are used in combination, and that combinations of if-then rules and facts are especially important for reeasoning. The following discussion makes this intuition more precise.

Using a rule, or set of rules, can mean different things depending on the rule engine that processes the rule set. In general, in a declarative rule engine, one is primarily concerned with using a set of if-then rules, combined with a set of facts, to infer new conclusions about what is true. In a production rule engine, one is primarily concerned with using a set of if-then rules, combined with a set of facts, to *fire* some rules and to make certain types of changes, perhaps to a system knowledge base.

For example, assume one is given, in a declarative rule language, the Basic Combination Rule introduced in Section 2.3 and the following *Document* with *Prefix* declarations for IMDb and DBpedia database concepts and names (of actors, roles, films), as well as a *Group* of facts (interpreted as a conjunction):

```
Document(
  Prefix(rdfs <http://www.w3.org/2000/01/rdf-schema#>)
  Prefix(imdbrel <http://example.com/imdbrelations#>)
  Prefix(imdbname <http://example.com/imdbnames#>)
  Prefix(dbpedia <http://dbpedia.org/ontology/>)

   Group(
     imdbrel:playsRole(imdbname:VivienLeigh imdbname:BlancheDubois)
     imdbrel:roleInFilm(imdbname:BlancheDubois imdbname:Streetcar)
   )
 )
```

One should then be able to conclude that `dbpedia:starring(imdbname:Streetcar imdbname:VivienLeigh)`.

The precise conditions that allow one to draw a conclusion from a set of rules and facts in RIF is discussed in detail in [RIF-BLD], which gives a model-theoretic semantics for inference in RIF. The precise conditions that allow rules to be fired and an action to be performed is discussed in [RIF-PRD], which gives an operational semantics for performing actions in production rule languages of RIF.

We do not discuss these semantics in detail: instead, we give an overview in sections 4.2 and 4.3, and discuss some basic reasoning principles that should suffice to enable a computer professional to understand the consequences of the rules that (s)he writes.

4.2 Reasoning in Declarative Rule Languages

The model-theoretic semantics for a declarative rule language seeks to answer the question: Given a world where a certain set of if-then rules and facts is true, what else is true in that world? The crux of the answer is that what is true in that world follows from certain rules that specify how one can combine simple if-then rules, facts, and logical operators.

It generally suffices to think about reasoning in RIF as being a combination of

1. *instantiation* --- especially *universal instantiation*
2. *Modus Ponens* --- enabling simple deduction
3. basic principles for evaluating conjunctions and disjunctions

These principles are explained below:

1. *Universal instantiation* is the process whereby one can take a universally quantified rule, such as *All humans are mortal*

```
Document(
  Prefix(bio <http://example.com/biology#>)

  Group(
    Forall ?X (
      If   bio:human(?X)
      Then bio:mortal(?X)
    )
  )
)
```

and instantiate it for any individual constant in the language. Since the rule is true of anyone, it is certainly true, for example, of an individual named Socrates. Thus, the rule above entails that

```
Document(
  Prefix(bio <http://example.com/biology#>)
  Prefix(phil <http://example.com/philosophers#>)

  Group(
    If   bio:human(phil:Socrates)
    Then bio:mortal(phil:Socrates)
  )
)
```

Note in particular that this universally instantiated rule *does not say Socrates is human*, nor that Socrates is mortal. It says that IF Socrates is human THEN Socrates is mortal. All we have done is to *instantiate* the predicates by replacing the variables in the previous example with constants.

2. *Modus Ponens* is the principle that allows concluding the consequent of a rule if one knows that the antecedent is true. That is, if one has a rule of the form

```
If P Then Q
```

and one also knows that `P`, it is obviously also the case that `Q`.

Thus, if one knows Aristotle's famous rule and also knows that `bio:human(phil:Socrates)`, one also ought to know that `bio:mortal(phil:Socrates)`. E.g. we would expect a RIF rule engine, when given the following input document, to be able to conclude that bio:mortal(phil:Socrates).

```
Document(
  Prefix(bio <http://example.com/biology#>)
  Prefix(phil <http://example.com/philosophers#>)

  Group(
    If   bio:human(?x)
    Then bio:mortal(?x)
  )
  Group(
    bio:human(phil:Socrates)
  )
)
```

3. The *basic principles of evaluating conjunctions and disjunctions* are simply these:

- A conjunction *A1 and A2 and ... and An* is true if each of its conjuncts is true.
- A disjunction *A1 or A2 or ... or An* is true if at least one of its disjuncts is true.

That is, assuming declarations for bio, ppl (people), soc (society), and ct (cities),

```
And(bio:fatherOf(ppl:John ppl:Bill) bio:fatherOf(ppl:John ppl:Mary))
```

is true if John is the father of both Bill and Mary;

```
Or(soc:livesIn(ppl:John ct:London) soc:livesIn(ppl:John ct:Paris))
```

is true if John lives in London or in Paris.

4.3 Production Rule Languages: Operational Semantics

The operational semantics for a production rule language is primarily concerned with the question: Given a working memory consisting of a set of if-then rules of the form *if condition then action* and a set of facts, what rules will fire, and what actions will take place? The crux of the answer is that an action *A* will take effect if there is a rule of the form *if C then A* that fires.

The somewhat more substantive answer is that the operational semantics specifies the following steps for determining the firing of rules and the triggering of actions. Note that these steps comprise a cycle which may repeat arbitrarily many times.

- **1.** A rule *potentially fires* if one can match a fact with the rule's condition; moreover, the rule has not previously fired due to a match with that specific fact.
- **2.** If several rules *potentially fire*, they are all in a conflict set.
- **3.** If several rules are in a conflict set, a rule is chosen to fire based on a predefined conflict strategy. The conflict strategy may mandate choosing a rule that has a higher priority, or that has been used more recently, for example. Section 6.2 gives some examples of how priorities may be used.
- **4.** If a rule of the form *if C then A* fires, the action *A* is carried out, causing a change in the working memory.
- **5.** This cycle repeats until there are no more changes in the working memory; that is, a fixpoint is reached.

For example, consider a rule set in PRD consisting of the following if-then rule and fact:

```
Document(
  Prefix(rdfs <http://www.w3.org/2000/01/rdf-schema#>)
  Prefix(imdbrelf <http://example.com/fauximdbrelations#>)
  Prefix(dbpediaf <http://example.com/fauxibdbrelations>)
  Prefix(ibdbrelf <http://example.com/fauxibdbrelations#>)

  Group(
    Forall ?Actor (
      If   Or(Exists ?Film (imdbrelf:winAward(?Actor ?Film))
              Exists ?Play (ibdbrelf:winAward(?Actor ?Play)) )
      Then assert(dbpediaf:awardWinner(?Actor))
```

```
      )
    imdbrelf:winAward(RobertoBenigni LifeIsBeautiful)
    )
  )
```

As this example demonstrates, churning through this step effects a kind of inference. The pattern matching mechanism incorporates the basic principles for evaluating connectives such as conjunctions and disjunctions; thus, the fact *imdbrelf:winAward(RobertoBenigni LifeIsBeautiful)* matches the condition in the if-then rule. Since there is only one rule and one fact, there is no conflict set; thus, the action *assert(dbpediaf:awardWinner(RobertoBenigni))* is carried out.

5 Datatypes and Built-ins

5.1 Datatypes and Built-ins in RIF

It is expected that all RIF dialects will support a set of commonly used datatypes, predicates, and functions. This list, specified in [RIF-DTB], includes such datatypes as `integer`, `boolean`, `string`, `date`, and `datetime`; as well as built-in predicates and functions that range over these datatypes. Examples follow:

- Examples of built-in numeric functions are arithmetic operators like:
  - `numeric-add`
  - `numeric-integer-divide`
- Examples of built-in numeric predicates:
  - `numeric-less-than`
  - `numeric-not-equal`
- Examples of built-in string functions:
  - `lower-case`
  - `concat`
  - `replace`
- Examples of built-in string predicates:
  - `starts-with`
  - `matches`
- Examples of built-in date/time functions:
  - `hours-from-dateTime`
  - `hours-from-time`
- Examples of built-in date/time predicates:
  - `dateTime-less-than`
  - `date-less-than`
  - `time-less-than`

5.2 Fourth RIF Example: Using datatypes and built-ins

As an example of a RIF rule in which datatypes and built-ins are used, again in the DBpedia movies domain, consider how one might characterize a movie star. A movie star will be someone who has starred in more than three successful films, made over a span of at least 5 years (this is to avoid including flash-in-the-pan stars from consideration as movie stars). A successful film will be characterized as one that received critical acclaim (for the purposes of this example, this will mean a critics' rating higher than 8) or was financially successful (for the purposes of this example, this will mean that the film grossed more than $100 million in ticket sales).

The following RIF document implements these rules. Note the use of built-ins such as `numeric-greater-than`, `numeric-subtract`, and `literal-not-identical`:

```
Document(
    Prefix(func <http://www.w3.org/2007/rif-builtin-function#> )
    Prefix(pred <http://www.w3.org/2007/rif-builtin-predicate#> )
    Prefix(rdfs <http://www.w3.org/2000/01/rdf-schema#>)
    Prefix(imdbrel <http://example.com/imdbrelations#>)
    Prefix(dbpedia <http://dbpedia.org/ontology/>)
    Prefix(ibdbrel <http://example.com/ibdbrelations#>)
  Group(
    Forall ?Actor ?Film ?Year (
```

```
        If  And( dbpedia:starring(?Film ?Actor)
                 dbpedia:dateOfFilm(?Film ?Year))
        Then  dbpedia:starredInYear(?Film ?Actor ?Year)
      )
      Forall ?Actor (
        If  ( Exists  ?Film1 ?Film2 ?Film3  ?Year1 ?Year2 ?Year3
                 And( dbpedia:starredInYear(?Film1 ?Actor ?Year1)
                      dbpedia:starredInYear(?Film2 ?Actor ?Year2)
                      dbpedia:starredInYear(?Film3 ?Actor ?Year3)
                      (External( pred:numeric-greater-than(
                                    External( func:numeric-subtract ?Year1 ?Year3)
                                    5)))
                      dbpedia:successful(?Film1)
                      dbpedia:successful(?Film2)
                      dbpedia:successful(?Film3)
                      External( pred:literal-not-identical(?Film1 ?Film2))
                      External( pred:literal-not-identical(?Film1 ?Film3))
                      External( pred:literal-not-identical(?Film3 ?Film2))
                    )
        Then dbpedia:movieStar(?Actor)
      )
      Forall ?Film (
        If Or( External(pred:numeric-greater-than(dbpedia:criticalRating(?Film) 8))
               External(pred:numeric-greater-than(dpbedia:boxOfficeGross(?Film) 100000000)))
        Then dpbedia:successful(?Film)
      )
    )
  )
```

6 Extensions to RIF Core: Constructs for BLD and PRD

We discuss below examples of rules in dialects that are extensions of RIF Core. The RIF dialect BLD extends RIF-Core by allowing logically-defined functions. The RIF dialect PRD extends RIF-Core by allowing prioritization of rules, negation, and explicit statement of knowledge-base modification.

6.1 BLD extensions to Core: Functions, Equality

So far, the arguments of predicates have been variables like `?Actor` or constants like `vivienleigh`. These are the only arguments to predicates that are allowed in RIF-Core. RIF-BLD is more expressive, and allows predicate arguments to be *function* applications. (For the purposes of this Primer, it suffices to know that a function is a mapping between a set of elements known as the *domain*, and a set of elements known as the *range*, and to note that there is a close connection between predicates and functions, in that both can be characterized by a set of ordered tuples.) For example, if `mainActor` is a function denoting the actor who plays the main character of a film, and `pottersorcerer` is a constant denoting the film *Harry Potter and the Sorcerer's Stone*, then `mainActor(pottersorcerer)` denotes the actor playing the main character of *Harry Potter and the Sorcerer's Stone*. For example,

- the application `seriesStar(mainActor(pottersorcerer))` of the unary predicate `seriesStar` to the unary function `mainActor` applied to `pottersorcerer` could stand for the proposition *the main actor in Harry Potter and the Sorcerer's Stone is a star of a movie series*.

In general, functions are applied to zero or more arguments, which themselves can be variables, constants, or function applications. A function application denotes an individual. This allows us to refer to an individual without coining a new constant (symbol) for it.

Moreover, RIF-BLD has a distinguished equality predicate, '=', which enables user-defined functions. For example, `mainActor(pottersorcerer) = danielradcliffe` could be used to equate the main actor of *Harry Potter and the Sorcerer's Stone* with *Daniel Radcliffe*. Another function, `mainChar`, could be defined to return the main characters of films. For example, `mainChar(pottersorcerer) = harrypotter` would equate the main character of *Harry Potter and the Sorcerer's Stone* with *Harry Potter*.

Besides such definitions of functions applied to arguments in a 'pointwise' fashion, functions can be also defined generally, using conditional equations, i.e. rules with '=' in the consequent. For example, based on the

function `MainChar` and the predicate `playsRole`, the function `mainActor` can be defined as in the following fragment:

```
If And( mainChar(?Film) = ?Char
        playsRole(?Actor ?Char))
Then mainActor(?Film) = ?Actor
```

Similarly, there can be a function `villainActor`, returning the actor playing the primary villain in a film.

One could then define a predicate combinedMainCharVillainActor that is true of someone who has played the main character in a film `?Film1` and the villain in a (possibly different) film `?Film2`, as shown in this RIF fragment:

```
Forall ?Actor ?Film1 ?Film2
  If And( ?Actor = mainActor(?Film1)
          ?Actor = villainActor(?Film2))
  Then combinedMainCharVillain(?Actor)
```

For example, Ralph Fiennes, who has played the primary villain, Lord Voldemort, in most of the Harry Potter films, played the main character, Count Laszlo de Almasy, in *The English Patient*.

6.2 PRD Extensions to RIF Core: Priorities, Negation, and Knowledge Base Modification

The RIF dialect PRD is targeted to users writing rules for production systems. For any rule 'if P then Q', 'P' is called the 'condition' and 'Q' is called the 'action'.

Below, we discuss three features that PRD adds to RIF-Core. Two of these, 'priorities' and 'knowledge base modification', are features which are closely associated with production systems; the other feature, 'negation', is a construct that is present in most declarative rule languages. As of this date, the only RIF dialect that supports negation is PRD.

6.2.1 Priorities

The concept of control is absent in RIF Core. That is, there is no way to specify that an implementation ought to consider one rule before another.

Often, however, one constructs a rule set with the intention that some rules are considered before others. This capability is generally present in production systems, in which there is an explicit notion of rules "firing" in a particular sequence. PRD supports specifying that certain rules have higher priority than others. That is, when there are two rules whose antecedents are satisfied, one can specify that one rule should be fired before the other.

The example in the following subsection demonstrates how priorities are used.

6.2.2 Negation

PRD is currently the only dialect of RIF to allow negation. Specifically, it allows adding negation to rule conditions. For example, the following rule set computes awardless film actors.

```
Document(
  Prefix(rdfs <http://www.w3.org/2000/01/rdf-schema#>)
  Prefix(imdbrel <http://example.com/imdbrelations#>)
  Prefix(dbpedia <http://dbpedia.org/ontology/>)
 Group( 2
   Forall ?Actor ?Film ?Role (
     If   And(imdbrel:playsRole(?Actor ?Role)
              imdbrel:roleInFilm(?Role ?Film))
     Then dbpedia:starring(?Film ?Actor)
   )
 )
  Group( 1
    Forall ?Actor (
      If   Not(Exists ?Film ( And(
```

```
                    dbpedia:starring(?Actor ?Film)
                    imdbrel:winAward(?Actor ?Film)) ))
          Then dbpedia:awardlessFilmActor(?Actor)
        )
      )
    )
```

Note that priorities are used here as well. One must first compute the set of all actors who have won awards; only then can one compute the set of actors who have not won awards. If one fires the rules in the reverse order, all actors would be considered awardless.

6.2.3 Knowledge Base Modification

For any PRD rule of the form 'If Condition Then Action', the 'Action' portion may correspond to an action that modifies a knowledge base. Specific examples of such actions are 'Assert', which adds a statement to a knowledge base; 'Retract', which removes a statement from the knowledge base; and 'Modify' which first retracts a statement, and then asserts a statement.. See 7.2 for an example of the use of `Assert` and `Modify` .

7 Using Frames in RIF

7.1 Example Using Frames

It is often convenient to use a frame construct when writing rules in RIF. For example, IMDb could decide to organize information about films in a frame that includes, among other things, information about the name of the film and the year in which the film was made. This information could be stored in predefined *slots*.

The use of slots is widespread in object-oriented programming languages such as C++ and Java, as well as knowledge-representation languages such as CLASSIC and OWL. Incorporating slots into RIF thus facilitates the interchange of information among languages that support slots, although a caveat is discussed in the next subsection.

In a variant of the data combination example discussed above, consider how DBpedia could import information about award winners from IMDb, if the latter used slots. In this example, the notation *frame[slot1->value1 ... slotn->valuen]* is used. Note the quantification over the particular type of frame (that is, the frame that holds information about film names and release years), as well as the slot values.

```
Document(
  Prefix(rdfs <http://www.w3.org/2000/01/rdf-schema#>)
  Prefix(imdbrelf <http://example.com/fauximdbrelations#>)
 Prefix(imdbnamef <http://example.com/fauximdbnames#>)
  Prefix(dbpediaf <http://example.com/fauxdbpediarelations>)
  Group(
    Forall ?Actor ?Film ?Year ?FF (
      If   And (?FF#imdbnamef:filmframe
                ?FF[imdbnamef:filmName -> ?Film
                    imdbname:filmYear -> ?Year]
               imdbrelf:winAward(?Actor ?FF))
      Then dbpediaf:winAward(?Actor ?Film ?Year) )
    )
  )
 )
```

Note that in this fabricated example, both IMDb and DBpedia use a predicate named *winAward*; these instances of the predicate are distinguished by their IRIs. Note also that DBpedia's *winAward* predicate has three arguments, while IMDb's has only two arguments (the second of which being a frame with two slots); the use of frames allows IMDb to reduce the number of predicates.

7.2 Distinguishing Slots in Object-Oriented Languages and RIF

The user of RIF should be aware that slots in RIF are treated differently than slots in object-oriented languages such as C++ or Java.

Consider the following fragment of Java:

```
Class Example {
  int a;

  public static void main(String args[]) {
    Example e1 = new Example();

    e1.a = 1;
    e1.a = 2;

    System.out.println(e1.a);
  }
}
```

The output of this program will be 2. The singleton slot e1.a is first assigned 1, but that value is then *replaced* by 2. In contrast, RIF frames can be considered as multimaps, supporting multiple assignments to the same attribute:

```
Document(
  Prefix(ex <http://example.com/exampleconcepts#>)
  Group (
    ex:e1#Example
    ex:e1[ex:a -> 1]
    ex:e1[ex:a -> 2]
  )
)
```

Under RIF's semantics, *a* has both the value 1 and 2; that is, both *(e1,1)* and *(e1,2)* are ordered pairs in the relation named by *a*. In other words, *a* functions as a predicate, which can comprise multiple ordered pairs. This is a direct consequence of the fact that RIF has the semantics of first-order logic.

Object-oriented languages, on the other hand, have the semantics of programming languages. The slot a is therefore understood as a variable which can be overwritten.

For applications in which the overwriting facility is desired, RIF-PRD has a modify action with operational semantics that can overwrite slot values. Consider the following example:

```
Document(
 Prefix(ex <http://example.com/exampleconcepts#>)
 Group (
   Do (
     (?e1 new())
     Assert(?e1 # ex:Example)
     Assert(?e1[ex:a -> 1])
     Modify(?e1[ex:a -> 2])
   ))
)
```

At the end of the action block (identified using the keyword "Do"), the slot a has the value 2.

8 Compatibility with Other Standards

RIF has been designed so that it is intercompatible with other Web Standards. Specifically, it is interoperable with both RDF [RDF-CONCEPTS] and OWL [OWL]. In practice, this means that one can reason from a combination of RIF documents and RDF and/or OWL documents. The practical import of this is that RIF does not just allow interchange of different rule sets; it also facilitates interchange of RDF triples and/or OWL 2 axioms

with rule sets in RIF. Specifically, RIF supports interchange of RDF triples and OWL axioms with RDF frame formulas.

8.1 Intercompatibility of RIF and RDF

The following example shows how one can reason from a combination of RIF and RDF documents. Consider, first, the following rule, which expresses the fact that the Hollywood Production Code, which at one point regulated the moral content of films produced in the United States, was in force between 1930 and 1968.

This rule can be expressed as follows: (The practitioner should also note the use of frames and built-in predicates in this example.)

```
Document(
  Prefix(rdfs <http://www.w3.org/2000/01/rdf-schema#>)
  Prefix(imdbrelf <http://example.com/fauximdbrelations#>)
 Prefix(imdbnamef <http://example.com/fauximdbnames#>)
  Prefix(dbpediaf <http://example.com/fauxdbpediarelations>)
 Group(
    Forall ?Film ?Year ??FF  (
      If   And ( ?FF#imdbnamef:filmframe
                  ?FF[imdbnamef:filmName -> ?Film
                      imdbnamef:filmYear -> ?Year]
                  External(pred:date-greater-than(?Year, 1930))
                  External(pred:date-less-than(?Year, 1968)))
      Then dbpediaf:enforcedProductionCode(?Film)
    )
  )
 )
```

Consider, now, the fact that *Gone With the Wind* was produced in 1939.

Naturally, one can represent this as the following fact in RIF:

```
imdbrel:FF[imdbnamef:filmName -> GoneWiththeWind imdbnamef:filmYear -> 1939]
```

Moreover, under RIF's semantics, one can conclude

```
dbpediaf:enforcedProductionCode(GoneWiththeWind)
```

The fact that RDF is compatible with RIF means that one can equally well represent this fact as an RDF triple. For example, an RDF graph might contain a triple such as

```
ex:GoneWiththeWind ex:filmYear ex:1939
```

which would contain the same information as the RIF fact above. Moreover, one can still use the RIF semantics to conclude, as above, that

```
dbpediaf:enforcedProductionCode(GoneWiththeWind)
```

The complete specification of using combinations of RIF and RDF (along with proofs of correctness) is given in [RIF-RDF+OWL].

Note that there is another issue, separate from intercompatibility, that concerns the ability to map a document from one language to another. For example, one may wish to map a RIF syntactic structure into an RDF graph.

[RIF in RDF] demonstrates how and to what extent this can be done. A reversible mapping is given.

8.2 Intercompatibility of RIF and OWL

OWL [OWL] is an ontology language that has been incorporated as a web standard. The current accepted version of OWL is OWL 2, an extension of the original OWL (now known as OWL 1). OWL 2 has several standard subsets (profiles) and two similar semantics, called the Direct Semantics and the RDF-Based-Semantics.

One can use techniques similar to those used to show that RDF and RIF are intercompatible to show that OWL and RIF are intercompatible, in the following sense:

- The combined use of RIF and OWL is well defined;
- One can infer conclusions from certain combinations of OWL axioms and RIF rules and facts.
- There is a subset of OWL, [RIF-OWLRL], that can be implemented with a RIF ruleset.

One often has a choice as to whether one can do something in RIF or OWL; many sentences of logic can be expressed in either language. However, there are many sentences of RIF that cannot be expressed in OWL. For example, although OWL 2 (unlike OWL 1) allows arbitrary composition of relations/properties, it is still not possible to have an OWL property chain that ends in a data value. One would need RIF for such purposes.

More details on the intercompatibility of RIF and OWL are given in [RIF-OWLRL] and [RIF-RDF+OWL].

9 The Test Suite

The RIF Working Group has put together a test suite of rules [TESTS] that have been designed to test certain special cases of RIF semantics in order to verify implementations. As a side effect, this suite also serves as a large body of examples of RIF syntax and semantics.

The RIF Test Cases document [TEST] describes the organization and format of the test cases. The first item in each test case is of particular importance: it indicates what *type* of test it is. Each type of test is documented: readers can click on the test type to see its description. Two important types of tests are *Positive Entailment Tests* and *Negative Entailment Tests*. Positive Entailment Tests, the most common in the test repository, each contain a set of rules listed as *Premises* and a set of *Conclusions*. If a correctly implemented RIF Rule Engine takes the premises as input, then the statements in the conclusion should be among its outputs. The conclusions are not necessarily exhaustive: it may be correct for a rule engine to output other statements as well.

Each Negative Entailment Test also has a set of premises. Instead of a conclusion, however, it has a *Non-conclusion* section that shows a set of statements that should *not* be output by a RIF rule engine. These test cases are typically used to demonstrate aspects of RIF semantics that may not be obvious.

All test cases are provided in standard RIF/XML syntax as well as in RIF Presentation Syntax for readability. As discussed above, RIF PS is quite close to the RIF Mixed Presentation Syntax used here: what mainly distinguishes the two syntaxes is that RIF PS inverts the arguments to IF and THEN and separates them with the **:-** symbol.

10 Learning More about RIF: Next Steps

After reading this Primer, the reader interested in using RIF is encouraged to learn more about RIF by

- reading the Frequently Asked Questions, and their answers, at http://www.w3.org/2005/rules/wiki/RIF_FAQ ;
- reading the [RIF-OVERVIEW], which gives a very brief description of RIF documents;
- consulting the full specifications for BLD and PRD at [RIF-BLD] and [RIF-PRD] ;
- joining the developers' mailing list at http://lists.w3.org/Archives/Public/public-rif-dev/

The best way to learn a language is to use it. We encourage the readers of this Primer to start using RIF.

11 Acknowledgements

This document is the product of the Rules Interchange Format (RIF) Working Group (see below), all of whom have contributed to this document through their work in developing RIF. The editors extend special thanks to:

Gary Hallmark, Sandro Hawke, and Stella Mitchell, for their thorough reviews, insightful discussions, and substantive suggestions for changes in content and wording; the working group chairs, Chris Welty and Christian de Sainte Marie, for their invaluable technical help and inspirational leadership; and W3C staff contact Sandro Hawke, a constant source of ideas, help, and feedback.

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

12 References

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

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

[RIF-BLD]
RIF Basic Logic Dialect (Second Edition) Harold Boley, Michael Kifer, eds. W3C Recommendation, 5 February 2013, http://www.w3.org/TR/2013/REC-rif-bld-20130205/. Latest version available at http://www.w3.org/TR/rif-bld/.

[RIF-Core]
RIF Core Dialect (Second Edition) Harold Boley, Gary Hallmark, Michael Kifer, Adrian Paschke, Axel Polleres, Dave Reynolds, eds. W3C Recommendation, 5 February 2013, http://www.w3.org/TR/2013/REC-rif-core-20130205/. Latest version available at http://www.w3.org/TR/rif-core/.

[RIF-DTB]
RIF Datatypes and Built-Ins 1.0 (Second Edition) Axel Polleres, Harold Boley, Michael Kifer, eds. W3C Recommendation, 5 February 2013, http://www.w3.org/TR/2013/REC-rif-dtb-20130205/. Latest version available at http://www.w3.org/TR/rif-dtb/.

[RIF in RDF]
RIF In RDF (Second Edition) Sandro Hawke, Axel Polleres, eds. W3C Working Group Note, 5 February 2013, http://www.w3.org/TR/2013/NOTE-rif-in-rdf-20130205/. Latest version available at http://www.w3.org/TR/rif-in-rdf/.

[RIF+XML-Data]
RIF Combination with XML data (Second Edition) Christian de Sainte Marie, editor. W3C Working Group Note, 5 February 2013, http://www.w3.org/TR/2013/NOTE-rif-xml-data-20130205/. Latest version available at http://www.w3.org/TR/rif-xml-data/.

[RIF-Overview]
RIF Overview, Kifer M. and Boley H. (Editors), W3C Rule Interchange Format Working Group Note. Latest Version available at http://www.w3.org/TR/rif-overview/.

[RIF-OWLRL]
OWL 2 RL in RIF (Second Edition) Dave Reynolds, editor. W3C Working Group Note, 5 February 2013, http://www.w3.org/TR/2013/NOTE-rif-owl-rl-20130205/. Latest version available at http://www.w3.org/TR/rif-owl-rl/.

[RIF-FLD]
RIF Framework for Logic Dialects (Second Edition) Harold Boley, Michael Kifer, eds. W3C Recommendation, 5 February 2013, http://www.w3.org/TR/2013/REC-rif-fld-20130205/. Latest version available at http://www.w3.org/TR/rif-fld/.

[RIF-PRD]
RIF Production Rule Dialect (Second Edition) Christian de Sainte Marie, Gary Hallmark, Adrian Paschke, eds. W3C Recommendation, 5 February 2013, http://www.w3.org/TR/2013/REC-rif-prd-20130205/. Latest version available at http://www.w3.org/TR/rif-prd/.

[RIF-RDF+OWL]
RIF RDF and OWL Compatibility (Second Edition) Jos de Bruijn, Chris Welty, eds. W3C Recommendation, 5 February 2013, http://www.w3.org/TR/2013/REC-rif-rdf-owl-20130205/. Latest version available at http://www.w3.org/TR/rif-rdf-owl/.

[RIF-Test]
RIF Test Cases (Second Edition) Stella Mitchell, Leora Morgenstern, Adrian Paschke, eds. W3C Working Group Note, 5 February 2013, http://www.w3.org/TR/2013/NOTE-rif-test-20130205/. Latest version available at http://www.w3.org/TR/rif-test/.

[RIF-UCR]
RIF Use Cases and Requirements (Second Edition) Adrian Paschke, Leora Morgenstern, David Hirtle, Allen Ginsberg, Paula-Lavinia Patranjan, Frank McCabe, eds. W3C Working Group Note, 5 February 2013, http://www.w3.org/TR/2013/NOTE-rif-ucr-20130205/. Latest version available at http://www.w3.org/TR/rif-ucr/.

[RIF-Test-Suite]
RIF Approved Test Cases, Mitchell, S. (Editor), RIF Approved Test Cases. Latest Version available at http://www.w3.org/2005/rules/wiki/Category:Approved.

[BA09]
Drools JBoss Rules 5.0 Developer's Guie", M. Bali, Packt, 2009.

[FH03]
Jess in Action: Java Rule-Based Systems, E. Friedman-Hill, Manning, 2003.

[SPARQL]
SPARQL Query Language for RDF, E. Prud'hommeaux, A. Seaborne (Editors), W3C Recommendation, World Wide Web Consortium, 12 January 2008, http://www.w3.org/TR/2008/REC-rdf-sparql-query-20080115/. Latest version available at http://www.w3.org/TR/rdf-sparql-query/.

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

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

[CM03]
Programming in Prolog: Using the ISO Standard, W. F. Clocksin and C. S. Mellish. Springer, 2003.

[DEFS99]
Ontobroker:Ontology Based Access to Distributed and Semi-Structured Information, S. Decker, M. Erdmann, D. Fensel, and R. Studer, DS-8, 1999.

[GR09]
SILK: Higher Level Rules with Defaults and Semantic Scalability, B. Grosof, "Web Reasoning and Rule Systems", 24--25, 2009.

[RZ09]
Vampire Reasoner, A. Riazanov, http://www.vprover.org/ , 2009.

[BCKSH08]
N3Logic: A logical framework for the World Wide Web, T. Berners-Lee, D. Connolly, L. Kagal, Y.Scharf, and J. Hendler, "Theory and Practice of Logic Programming", 8(3), 249-269, 2008.

[SWRL-Ref]
SWRL: A Semantic Web Rule Language Combining OWL and Rule-ML, I. Horrocks, P. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, M. Dean, http://www.w3.org/Submission/SWRL/, 2004.

[EULERSHARP10]
Euler Proof Mechanism, J. de Roo. http://eulersharp.sourceforge.net/ , 2010.

[ILOG10]
ILOG Website,http://www-01.ibm.com/software/websphere/ilog-migration/ , 2010.

[ORACLE10]
ORACLE Website, http://www.oracle.com/technetwork/middleware/business-rules/overview/index.html , 2010.

13 Appendix: Change Log (Informative)

First publication: 11 December 2012