Rule Interchange on the Web

Harold Boley (NRC Canada)
Michael Kifer (State Univ. of NY at Stony Brook)
Paula-Lavinia Pătrânjan (REWERSE)
Axel Polleres (DERI Galway)

Reasoning Web 2007 - September 7, 2007

Outline

Motivation

The rule-based programming paradigm offers a

- flexible and adaptive approach towards application development
- executable specification: declarative model $\longrightarrow$ rapid prototyping
- high level means for deploying applications in various domains

For exploiting further the potential of the rule-based approach

- both the **Business Rules** and **Semantic Web** communities started to develop solutions to reuse and integrate knowledge
  - distributed over the Web
  - specified in different rule languages

(Some) rule-based systems and languages

... tailored (more or less) to the Web

- Rendering rules: CSS
- Prolog with XML support: Ciao Prolog, SWI Prolog
- Rules for XML: Xcerpt, XChange, ...
- Rules for RDF: TRIPLE, JenaRules, N3, F-Logic OntoBroker/OntoStudio (F-Logic), Fair Isaac Blaze Advisor (SRL),
  Oracle Business Rules, Prova (Prolog and Java), IRIS (WSML/WRL),
  FLORA-2 (F-Logic) ...

... and many more!

Application Examples

- Negotiating e-business contracts across rule platforms
  - reuse business documents made available online
- Access to business rules of supply chain partners
  - ease the integration of business processes
- Collaborative policy development for dynamic spectrum access
  - reuse protocols of wireless communication devices/services
- Ruleset integration for medical decision support
  - complex decision making systems using diff. data sources
- Vocabulary mapping for data integration
  - reuse rules implementing mappings between data models

(More) Application Examples

- Negotiating e-commerce transactions
  - by exchanging policies and credentials
  - an example rule

    ```
    Disclose Alice's credit card information only to
    online shops belonging to the Better Business Bureau.
    ```
  - such rules can elegantly be specified in Protune (recall Daniel's talk!)
- Publishing rules for interlinked metadata
  - specify and publish implicit data in form of rules
  - an example rule

    ```
    If a movie is listed at http://amdb.example.org but
    not listed at http://imd.example.org
    then it is an independent movie.
    ```

The multitude of such use cases drives the strong interest in rules and rule interchange technology!

(More) Application Examples

- Negotiating e-commerce transactions
  - by exchanging policies and credentials
  - an example rule

    ```
    Disclose Alice's credit card information only to
    online shops belonging to the Better Business Bureau.
    ```
  - such rules can elegantly be specified in Protune (recall Daniel's talk!)
- Publishing rules for interlinked metadata
  - specify and publish implicit data in form of rules
  - an example rule

    ```
    If a movie is listed at http://amdb.example.org but
    not listed at http://imd.example.org
    then it is an independent movie.
    ```

The multitude of such use cases drives the strong interest in rules and rule interchange technology!

Current Efforts

Efforts such as ...

- Rule Markup Initiative - RuleML
- OMG - PRR and SBVR
- REWERSE - Xcerpt, XChange, ..., R2ML
- W3C Member Submissions - SWRL, WRL, SWSL Rules

...led to the W3C Rule Interchange Format Working Group (RIF WG)

- 78 participants from industry and academia
- chaired by representatives of IBM and ILOG
- chartered to standardize a common format for interchanging rules
  - which is not a trivial task!

Current Efforts

Efforts such as . . .

- Rule Markup Initiative - RuleML
- OMG - PRR and SBVR
- REWERSE - Xcerpt, XChange, . . . , R2ML
- W3C Member Submissions - SWRL, WRL, SWSL Rules

. . . led to the W3C Rule Interchange Format Working Group (RIF WG)

- 78 participants from industry and academia
- chaired by representatives of IBM and ILOG
- chartered to standardize a common format for interchanging rules
  - which is not a trivial task!

Rule Types

PR vendors, database systems vendors, and Semantic Web researchers have different views on the notion of *rules*:

- *deduction rules* (derivation or constructive rules)
  - derive knowledge by means of logical inference
- *normative rules* (structural rules)
  - pose constraints on the data and the logic of applications
- *reactive rules* (dynamic rules)
  - automatically execute actions when events occur and/or conditions become true
  - for example
    - Production rules (PR)
    - Event-Condition-Action (ECA) rules

... and these rule types raise different requirements on an interchange format

Rule Types

PR vendors, database systems vendors, and Semantic Web researchers have different views on the notion of *rules*:

- *deduction rules* (derivation or constructive rules)
  - derive knowledge by means of logical inference
- *normative rules* (structural rules)
  - pose constraints on the data and the logic of applications
- *reactive rules* (dynamic rules)
  - automatically execute actions when events occur and/or conditions become true
  - for example
    - Production rules (PR)
    - Event-Condition-Action (ECA) rules

... and these rule types raise different requirements on an interchange format

Rule Types

PR vendors, database systems vendors, and Semantic Web researchers have different views on the notion of *rules*:

- *deduction rules* (derivation or constructive rules)
  - derive knowledge by means of logical inference
- *normative rules* (structural rules)
  - pose constraints on the data and the logic of applications
- *reactive rules* (dynamic rules)
  - automatically execute actions when events occur and/or conditions become true
  - for example
    - Production rules (PR)
    - Event-Condition-Action (ECA) rules

... and these rule types raise different requirements on an interchange format

Rule Types

PR vendors, database systems vendors, and Semantic Web researchers have different views on the notion of *rules*:

- *deduction rules* (derivation or constructive rules)
  - derive knowledge by means of logical inference
- *normative rules* (structural rules)
  - pose constraints on the data and the logic of applications
- *reactive rules* (dynamic rules)
  - automatically execute actions when events occur and/or conditions become true
  - for example
    - Production rules (PR)
    - Event-Condition-Action (ECA) rules

... and these rule types raise different requirements on an interchange format

Rule Types

Example (deductive) rule

IF movie ?M was produced before 1930
THEN ?M is a black and white movie

- IF-part
  - specifies a **condition** for retrieving data on movies
  - binds the variable ?M to data items
- THEN-part
  - constructs/derives new data by using the retrieved bindings
  - using relational database terminology you can say it creates a 'view' over movie data

Rule Types

Example (normative) rule

Each movie must have a single production year.

- specifies a **condition** which must not be violated by the data
- two different production years for the same movie is an indication of corrupted data

- derivation and dynamic rules can be used to implement normative rules
- implementation decision depends on the application and the available support for rules

Rule Types

Example (reactive) rule

`ON` request from customer ?C to book movie ?M
`IF` customer ?C is blacklisted
`DO` deny ?C's request for ?M

- ON-part waits for a request for a movie to come in (an **event**)
- IF-part checks a **condition** on the customer's data
- DO-part
  - specifies the **action** to be executed
  - on a request from a blacklisted customer

Rule Types

Condition part is common to all possible rule “dialects”, so

- let’s start with developing a format for interchanging rule conditions
- and then extend it!

Rule Types

Example rule variant implemented using XChange (recall Paula's talk!):

```
ON
  xchange:event {{
    xchange:sender { var S },
    order {{
      customer { var C }
    }}
  }}
FROM
  in { resource { "http://MoviShop.org/blacklisted.xml", XML },
    desc var C
  }
DO
  xchange:event {
    xchange:recipient { var S },
    message { "Your request can not be processed,
               since you are blacklisted" }
  }
END
```

Rule Types

Example rule variant implemented using ILOG JRules (recall Philippe's talk!):

```
rule denyBlacklistedCustomers {
     when {
       c: Customer (blacklisted == yes);
       m: MoviesCart (owner == c; value > 0);
     } then {
        out.println ("Customer " + c.name + " is blacklisted!");
        retract m;
     }
 }
```

Rule Types

The proposed classification of rules

- basis for discovering commonalities between rule languages
- however, they reveal also considerable differences regarding
  - *syntax*,
  - *supported features*, and
  - *semantics*

. . . a standard interchange format should be able to interchange rules

- not only with different structure
- but also intertranslatable constructs and semantics!

W3C RIF WG Charter

... i.e., what the W3C RIF WG should do[1]:

Phase I

- simple, but extensible interchange format for Horn-like rules (RIF Core)
- Dec 2005 - Nov 2007

Phase II

- extensions in form of RIF Dialects (e.g. FOL, PR)
- until June 2008

Emphasizes compatibility with

- Web technologies - XML
- Semantic Web technologies - RDF, OWL, SPARQL

[1] http://www.w3.org/2005/rules/wg/charter.html

The Web as Framework for Rule Interchange

- The Web is a success story in terms of **linking** data (HTML)
- Web formats, such as XML have made it to nowadays standard formats for also non-Web **data exchange**
- The next generation of the Web will allow to link and exchange data (RDF) and its **structure** (models/vocabularies, ontologies in RDF Schema, OWL) even more flexible
  $\longrightarrow$ this is often called the **Semantic** Web
- As an important facilitator for this flexibility, the Semantic Web will also allow to **exchange rules**!

ie.: The Semantic Web is about exchange of Data, Data/Domain Models and Rules (e.g., by RIF)!

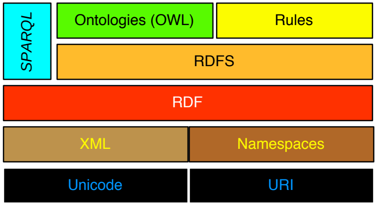
Let us talk about these foundations a bit, since they have some implications for RIF!

The Web as Framework for Rule Interchange

- The Web is a success story in terms of **linking** data (HTML)
- Web formats, such as XML have made it to nowadays standard formats for also non-Web **data exchange**
- The next generation of the Web will allow to link and exchange data (RDF) and its **structure** (models/vocabularies, ontologies in RDF Schema, OWL) even more flexible
  → this is often called the **Semantic** Web
- As an important facilitator for this flexibility, the Semantic Web will also allow to **exchange rules**!

ie.: The Semantic Web is about exchange of Data, Data/Domain Models and Rules (e.g., by RIF)!

Let us talk about these foundations a bit, since they have some implications for RIF!



The (Semantic) Web architecture stack

Semantic Web architecture 1/5: XML

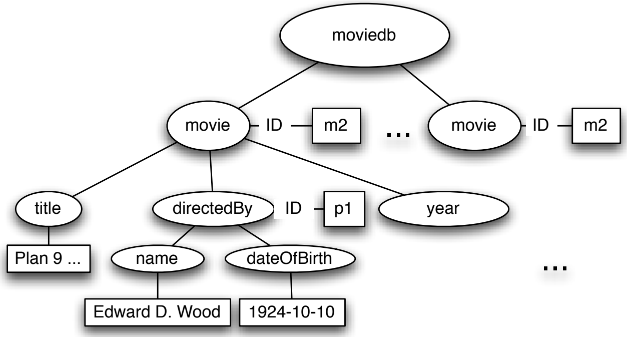
```
<?xml version="1.0"  encoding="UTF-8"?>
<moviedb xmlns="http://imd.example.org/ns/">
  <movie ID="m1">
    <title>Plan 9 from Outer Space</title>
    <directedBy ID="p1">
      <name>Edward D. Wood Jr.</name>
      <dateOfBirth>1924-10-10</dateOfBirth>
    </directedBy>
    ...
    <year>1959</year>
  </movie>

  ...
</moviedb>
```

- Tree to handle semi-structured data
- Unique identifiers to disambiguate formats (namespaces)
- Facilitates data exchange on a syntactical level
- Take-up in many applications which need common formats (ebXML, Web Services,. . .)

⇒ RIF will also have an XML syntax!

Semantic Web architecture 1/5: XML



- Tree to handle semi-structured data
- Unique identifiers to disambiguate formats (namespaces)
- Facilitates data exchange on a syntactical level
- Take-up in many applications which need common formats (ebXML, Web Services,. . .)

⇒ RIF will also have an XML syntax!

Semantic Web architecture 2/5: RDF

- Integrating **different** XML formats is still sometimes tricky (XSLT), due to the tree format of XML.
- The data model of the Semantic Web is **graphs** instead of trees.

An RDF graph is made up by a set of “**statements**” (i.e.simple triples) about **resources**:

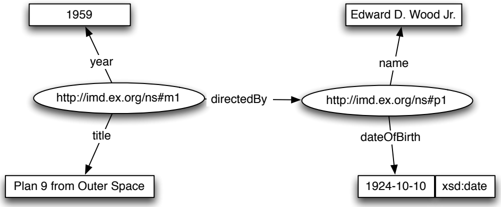
```
<http://imd.ex.org/ns#m1> rdf:type imd:Movie .
<http://imd.ex.org/ns#m1> imd:title       "Plan 9 from Outer Space" .
<http://imd.ex.org/ns#m1> imd:directedBy  <http://imd.ex.org/ns#p1> .
<http://imd.ex.org/ns#m1> imd:year "1959"


<http://imd.example.org/ns#p1> foaf:name        "Edward D. Wood Jr." .
<http://imd.example.org/ns#p1> bio:dateOfBirth "1924-10-10"^^xsd:date .
   ...
```

Semantic Web architecture 2/5: RDF

- Integrating **different** XML formats is still sometimes tricky (XSLT), due to the tree format of XML.
- The data model of the Semantic Web is **graphs** instead of trees.

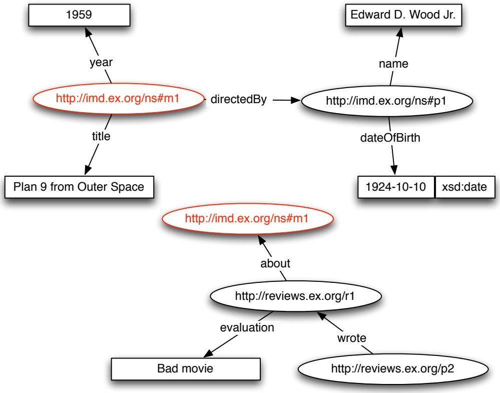
Sets of RDF statements may be viewed as directed, labelled Graphs:



Semantic Web architecture 2/5: RDF

- Integrating **different** XML formats is still sometimes tricky (XSLT), due to the tree format of XML.
- The data model of the Semantic Web is **graphs** instead of trees.

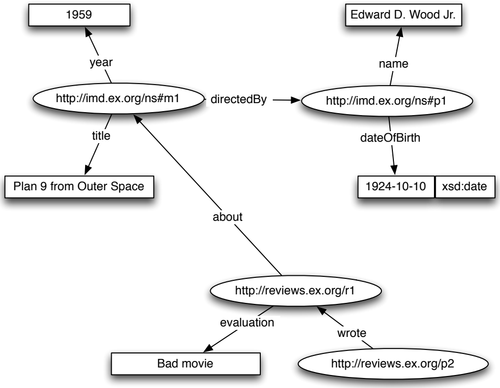
Sets of RDF statements may be viewed as directed, labelled Graphs:



Semantic Web architecture 2/5: RDF

- Integrating **different** XML formats is still sometimes tricky (XSLT), due to the tree format of XML.
- The data model of the Semantic Web is **graphs** instead of trees.

Sets of RDF statements may be viewed as directed, labelled Graphs:

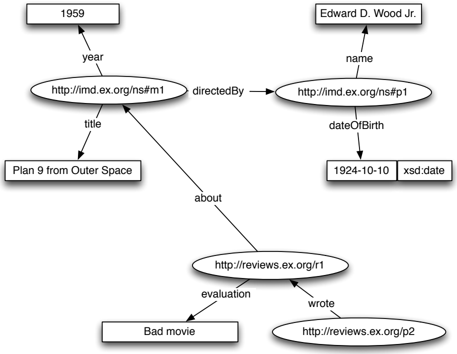


The flat data model of RDF is **easier to integrate** than XML!

Semantic Web architecture 3/5: RDFS/OWL

The Semantic Web architecture has defined more flexible ways to **exchange and integrate** not only data, but also **data/domain models**:

- RDFS (= RDF Schema) and OWL (= Web Ontology Language)
- allow to add classes and types to RDF
- allow to express subclass hierarchies, subproperty hierarchies, etc.



Semantic Web architecture 3/5: RDFS/OWL

The Semantic Web architecture has defined more flexible ways to **exchange and integrate** not only data, but also **data/domain models**:

- RDFS (= RDF Schema) and OWL (= Web Ontology Language)
- allow to add classes and types to RDF
- allow to express subclass hierarchies, subproperty hierarchies, etc.

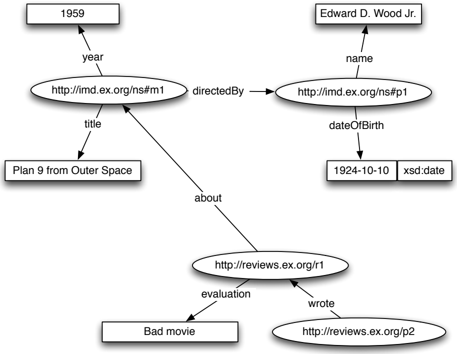
OWL and RDFS can express additional relations among types and properties, e.g.:

- *each Director is a Person (subclass)*
- *each Reviewer is a Person (subclass)*
- *somebody who directed a Movie is a Director (range restriction)*
- *somebody who wrote a Review is a Reviewer (domain restriction)*
- etc.

Semantic Web architecture 3/5: RDFS/OWL

The Semantic Web architecture has defined more flexible ways to **exchange and integrate** not only data, but also **data/domain models**:

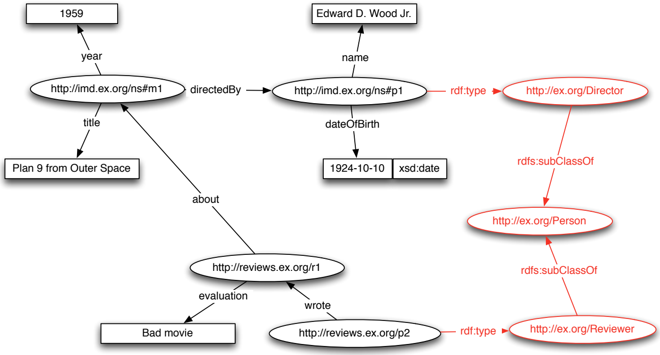
- RDFS (= RDF Schema) and OWL (= Web Ontology Language)
- allow to add classes and types to RDF
- allow to express subclass hierarchies, subproperty hierarchies, etc.



Semantic Web architecture 3/5: RDFS/OWL

The Semantic Web architecture has defined more flexible ways to **exchange and integrate** not only data, but also **data/domain models**:

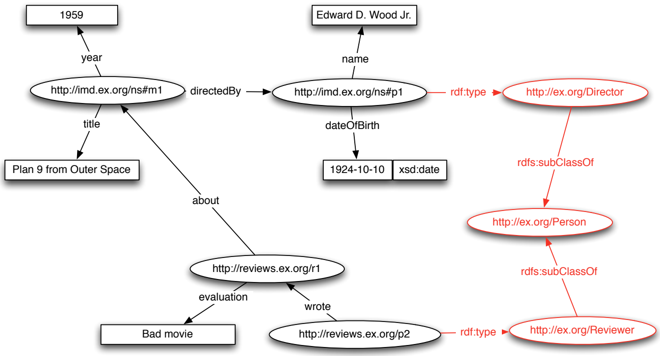
- RDFS (= RDF Schema) and OWL (= Web Ontology Language)
- allow to add classes and types to RDF
- allow to express subclass hierarchies, subproperty hierarchies, etc.



Semantic Web architecture 3/5: RDFS/OWL

The Semantic Web architecture has defined more flexible ways to **exchange and integrate** not only data, but also **data/domain models**:

- RDFS (= RDF Schema) and OWL (= Web Ontology Language)
- allow to add classes and types to RDF
- allow to express subclass hierarchies, subproperty hierarchies, etc.



Real power of common domain models reveals in **sharing**, **exchanging** and **reusing** them!

Semantic Web architecture 4/5: XML vs. RDF(S)+OWL

	XML	RDF
Data Model:	Tree	Graph
Identifiers:	element, attribute names	everything identified by URIs
Data:	in the leaves	in the nodes
Relations	in the nodes	in the edges
Data structure :	XML Schema (syntax)	RDFS/OWL (semantics)

Implication for “general” Web rule interchange:

- RIF shall support both XML and RDF as data formats

Semantic Web architecture 4/5: XML vs. RDF(S)+OWL

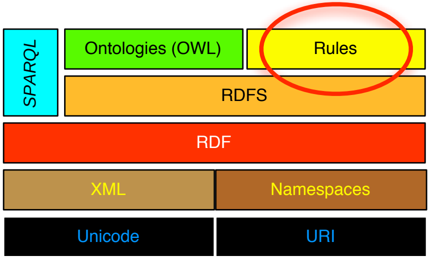
	XML	RDF
Data Model:	Tree	Graph
Identifiers:	element, attribute names	everything identified by URIs
Data:	in the leaves	in the nodes
Relations	in the nodes	in the edges
Data structure :	XML Schema (syntax)	RDFS/OWL (semantics)

Implication for “general” Web rule interchange:

- RIF shall support both XML and RDF as data formats

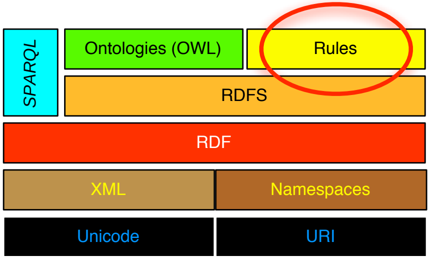
Semantic Web architecture 5/5: Rules

- After exchanging Data and Domain Models on the Web has been enabled, Rules are the next step! ⇒ RIF



Semantic Web architecture 5/5: Rules

- After exchanging Data and Domain Models on the Web has been enabled, Rules are the next step! $\Rightarrow$ RIF



Support for lower layers of the SW Arch in RIF:

Implications:

- RIF will use URIs as identifiers (for predicates, constants, etc.)
- RIF will allow both RDF and XML as data formats.
- RIF shall allow to take RDFS, OWL (and XSD?) domain models into account

This is not a trivial goal to achieve:

- Vertical Compatibility/exchange not even solved on the lower layers of the SW stack:
  - How to get from XML to RDF? W3C is working on it: GRDDL, RDFa, etc.
  - How to get from XML Schema to RDFS
  - Tricky issues around mixing OWL DL with arbitrary RDF
  - We also want to reuse/integrate other W3C specs (XQuery/XPath, SPARQL, etc.)

. . . We will get to some of these issues later on!

Support for lower layers of the SW Arch in RIF:

Implications:

- RIF will use URIs as identifiers (for predicates, constants, etc.)
- RIF will allow both RDF and XML as data formats.
- RIF shall allow to take RDFS, OWL (and XSD?) domain models into account

This is not a trivial goal to achieve:

- Vertical Compatibility/exchange not even solved on the lower layers of the SW stack:
  - How to get from XML to RDF? W3C is working on it: GRDDL, RDFa, etc.
  - How to get from XML Schema to RDFS
  - Tricky issues around mixing OWL DL with arbitrary RDF
  - We also want to reuse/integrate other W3C specs (XQuery/XPath, SPARQL, etc.)

. . . We will get to some of these issues later on!

Support for lower layers of the SW Arch in RIF:

Implications:

- RIF will use URIs as identifiers (for predicates, constants, etc.)
- RIF will allow both RDF and XML as data formats.
- RIF shall allow to take RDFS, OWL (and XSD?) domain models into account

This is not a trivial goal to achieve:

- Vertical Compatibility/exchange not even solved on the lower layers of the SW stack:
  - How to get from XML to RDF? W3C is working on it: GRDDL, RDFa, etc.
  - How to get from XML Schema to RDFS
  - Tricky issues around mixing OWL DL with arbitrary RDF
  - We also want to reuse/integrate other W3C specs (XQuery/XPath, SPARQL, etc.)

. . . We will get to some of these issues later on!

Some example Rules on top of RDF data 1/2

Given the RDF Data from above. . .

```
<http://imd.ex.org/ns#m1> rdf:type imd:Movie .
<http://imd.ex.org/ns#m1> imd:title       "Plan 9 from Outer Space" .
<http://imd.ex.org/ns#m1> imd:directedBy  <http://imd.ex.org/ns#p1> .
<http://imd.ex.org/ns#m1> imd:year "1959"


<http://imd.example.org/ns#p1> foaf:name        "Edward D. Wood Jr." .
<http://imd.example.org/ns#p1> bio:dateOfBirth "1924-10-10"^^xsd:date .
   ...
```

Some example Rules on top of RDF data 1/2

Given the RDF Data from above. . .

. . . how would we write (and exchange) rules? For instance:

`IF` movie ?M was produced before 1930
`THEN` ?M is a black and white movie

Even writing it as a Horn rule, there are several possibilities to embed RDF:

Some example Rules on top of RDF data 1/2

Given the RDF Data from above. . .

. . . how would we write (and exchange) rules? For instance:

IF movie ?M was produced before 1930
THEN ?M is a black and white movie

Even writing it as a Horn rule, there are several possibilities to embed RDF:

unary/binary predicate style:

```
∀ ?M "moviShop:BWMovie"( ?M ) ←
  ( ∃ ?Y
    "imd:Movie"( ?M ) ∧ "imd:Year"( ?M, ?Y ) ∧
    ?Y < "1930" )
```

We assume that we can use IRIs (QNames) as predicate/constant names here, variables are denoted by question marks.

Some example Rules on top of RDF data 1/2

Given the RDF Data from above. . .

. . . how would we write (and exchange) rules? For instance:

IF movie ?M was produced before 1930
THEN ?M is a black and white movie

Even writing it as a Horn rule, there are several possibilities to embed RDF:

one designated predicate `triple` for RDF triples:

```
∀ ?M triple( ?M,"rdf:type","moviShop:BWMovie")←
  ( ∃ ?Y
    triple( ?M,"rdf:type","imd:Movie" ) ∧ triple( ?M,"imd:Year",?Y )
    ?Y < "1930" )
```

This notion is more verbose, but has advantages as we will see. . .

Some example Rules on top of RDF data 1/2

Given the RDF Data from above…
…how would we write (and exchange) rules? For instance:

IF movie ?M was produced before 1930
THEN ?M is a black and white movie

Even writing it as a Horn rule, there are several possibilities to embed RDF:

slotted notation, i.e. FRAMES for RDF triples:

```
∀ ?M ?M#moviShop:BWMovie←
  ( ∃ ?Y
    ?M#imd:Movie[ imd:Year → ?Y ] ∧
    ?Y < "1930" )
```

Logic languages like F-Logic (Kifer et al. 1995) support this while still staying in a first-order semantics. '#' (class membership), '##' (is-A), and '[ ]' are basically syntactic sugar for the verbose notation that we used in the last slide.

Some example Rules on top of RDF data 1/2

Given the RDF Data from above. . .

. . . how would we write (and exchange) rules? For instance:

IF movie ?M was produced before 1930
THEN ?M is a black and white movie

Even writing it as a Horn rule, there are several possibilities to embed RDF:

unary/binary predicate style:

```
∀ ?M "moviShop:BWMovie"( ?M ) ←
  ( ∃ ?Y
    "imd:Movie"( ?M ) ∧ "imd:Year"( ?M, ?Y ) ∧
    ?Y < "1930" )
```

Some example Rules on top of RDF data 1/2

Given the RDF Data from above...

...how would we write (and exchange) rules? For instance:

IF movie ?M was produced before 1930
THEN ?M is a black and white movie

Even writing it as a Horn rule, there are several possibilities to embed RDF:

unary/binary predicate style:

```
∀ ?M "moviShop:BWMovie"( ?M )←
  ( ∃ ?Y
    "imd:Movie"( ?M ) ∧ "imd:Year"( ?M, ?Y ) ∧
    "op:date-less-than"( ?Y , "1930-01-01T00:00:00Z"^^dateTime ) )
```

Alternative: How about built-in functions like '<'?
We could/should reuse XPath/XQuery standard functions here, we could/should allow typed literals (primitive datatypes) as present in RDF.

Some example Rules on top of RDF data 2/2

. . . So, we see that some design decisions need to be made on how to embed different data models such as for instance RDF.

Let's consider another prominent example rule: the RDFS entailment rule (rdfs3) from semantics (Hayes 1999):

`IF` an RDF graph contains triples `(P rdfs:range C)` and `(S P O)`
`THEN` the triple `O rdf:type C` is entailed

Some example Rules on top of RDF data 2/2

. . . So, we see that some design decisions need to be made on how to embed different data models such as for instance RDF.

Let's consider another prominent example rule: the RDFS entailment rule (rdfs3) from semantics (Hayes 1999):

> `IF` an RDF graph contains triples `(P rdfs:range C)` and `(S P O)`
> `THEN` the triple `O rdf:type C` is entailed

Some example Rules on top of RDF data 2/2

. . . So, we see that some design decisions need to be made on how to embed different data models such as for instance RDF.

Let's consider another prominent example rule: the RDFS entailment rule (rdfs3) from semantics (Hayes 1999):

> `IF` an RDF graph contains triples `(P rdfs:range C)` and `(S P O)`
> `THEN` the triple `O rdf:type C` is entailed

Can be written as a Horn rule as follows (using the triple predicate notation):

```
∀ ?S,?P,?O,?C triple(?O,"rdf:type",?C) ←
              ( triple(?P,"rdf:range",?C) ∧ triple(?S,?P,?O) )
```

Some example Rules on top of RDF data 2/2

...So, we see that some design decisions need to be made on how to embed different data models such as for instance RDF.

Let's consider another prominent example rule: the RDFS entailment rule (rdfs3) from semantics (Hayes 1999):

> `IF` an RDF graph contains triples `(P rdfs:range C)` and `(S P O)`
> `THEN` the triple `O rdf:type C` is entailed

Note: The unary/binary predicate version would go outside first-order:

```
∀ ?S,?P,?O,?C "rdf:type"(?O,?C) ←
              ( "rdf:range"(?P,?C) ∧ ?P(?S,?O) )
```

Some example Rules on top of RDF data 2/2

. . . So, we see that some design decisions need to be made on how to embed different data models such as for instance RDF.

Let's consider another prominent example rule: the RDFS entailment rule (rdfs3) from semantics (Hayes 1999):

`IF` an RDF graph contains triples `(P rdfs:range C)` and `(S P O)`
`THEN` the triple `O rdf:type C` is entailed

Slotted/F-Logic version works as well:

```
∀ ?S,?P,?O,?C ?O#?C ←
               ( ?P[rdf:range->?C] ∧ ?S[?P->?O] )
```

Some example Rules on top of RDF data 2/2

...So, we see that some design decisions need to be made on how to embed different data models such as for instance RDF.

Let's consider another prominent example rule: the RDFS entailment rule (rdfs3) from semantics (Hayes 1999):

`IF` an RDF graph contains triples `(P rdfs:range C)` and `(S P O)`
`THEN` the triple `O rdf:type C` is entailed

Slotted/F-Logic version works as well:

```
∀ ?S,?P,?O,?C ?O#?C ←
              ( ?P[rdf:range->?C] ∧ ?S[?P->?O] )
```

Let's see how this looks in several existing rules systems for RDF!

Some SW Rules Language Systems: TRIPLE

TRIPLE:

- M.Sintek, S.Decker, A.Harth, 2002
- Frame syntax, similar to F-Logic
- Special syntax to import RDF, define namespaces, etc.

```
rdf:= 'http://www.w3.org/1999/02/22-rdf-syntax-ns#'.
rdfs:= 'http://www.w3.org/2000/01/rdf-schema#'.
type := rdf:type.
range := rdfs:range.

FORALL O,C O[type->C] <- EXISTS  S,P (S[P->O] AND P[range->C]).
```

Some SW Rules Systems: JENA

JENA:

- HP Labs Bristol
- proprietary syntax
- natively dealing with RDF, rules as add-on part of Jena API.

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.

[rdfs3: (?s ?p ?o) (?p rdfs:range ?c) -> (?o rdf:type ?c)]
```

Some SW Rules Systems: N3

N3:

- W3C people, Dan Connolly, TimBL
- syntax extends N-Triples RDF syntax by rules
- natively extension of RDF, implemented in a prototype system (cwm).

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix log: <http://www.w3.org/2000/10/swap/log#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .

{ <#p> rdfs:range <#c>. <#s> <#p> <#o> . }
               log:implies { <#o> rdf:type <#c> }.
```

Some SW Rules Systems: FLORA-2

FLORA-2:

- M. Kifer et al.
- A reference implementation for F-Logic with RDF support
- Additional support for higher-order modeling via HiLog

```
:- iriprefix rdf = 'http://www.w3.org/2000/01/rdf-schema#'.

?O[rdf#type->?C] :- ?S[?P->?O], ?P[rdf#range->?C].
```

Some SW Rules Systems: `dlvhex`

dlvhex:

- R. Schindlauer et al., developed within REWERSE
- SW rules engine on top of the `dlv` system, stable model semantics
- Prolog-style syntax, special predicates for RDF import, namespaces, etc.

```
#namespace("rdf","http://www.w3.org/1999/02/22-rdf-syntax-ns#")
#namespace("rdfs","http://www.w3.org/2000/01/rdf-schema#")

triple(O,rdf:type,C) :-  triple(P,rdfs:range,C), triple(S,P,O).
triple(S,P,O) :-
       &rdf["http://UrlWithRdfData.example.org/data.rdf"](S,P,O).
```

Some SW “Rules” Systems: SPARQL engines! 1/2

SPARQL:

- upcoming W3C query language standard
- Actually, SPARQL’s `CONSTRUCT` queries may be viewed as rules as well
- Syntax a bit like merging SQL with N-Triples/Turtle.

```
CONSTRUCT { ?M rdf:type moviShop:BWMovie }
WHERE { ?M rdf:type imd:Movie . ?M imd:year ?Y .
        FILTER (?Y < 1930) }

CONSTRUCT { ?O rdf:type ?C }
WHERE { ?P rdf:range ?C . ?S ?P ?O . }

CONSTRUCT { ?P foaf:knows _:a }
WHERE { ?P rdf:type ex:socialPerson . }
```

Issues:

- No recursive/fixpoint evaluation in standard engines
- No combination of several CONSTRUCTs in standard engines
- BTW: Blank nodes in rule heads (last rule) would make things non-Horn.

Some SW “Rules” Systems: SPARQL engines! 1/2

SPARQL:

- upcoming W3C query language standard
- Actually, SPARQL’s `CONSTRUCT` queries may be viewed as rules as well
- Syntax a bit like merging SQL with N-Triples/Turtle.

```
CONSTRUCT { ?M rdf:type moviShop:BWMovie }
WHERE { ?M rdf:type imd:Movie . ?M imd:year ?Y .
        FILTER (?Y < 1930) }

CONSTRUCT { ?O rdf:type ?C }
WHERE { ?P rdf:range ?C . ?S ?P ?O . }

CONSTRUCT { ?P foaf:knows _:a }
WHERE { ?P rdf:type ex:socialPerson . }
```

Issues:

- No recursive/fixpoint evaluation in standard engines
- No combination of several CONSTRUCTs in standard engines
- BTW: Blank nodes in rule heads (last rule) would make things non-Horn.

Some SW “Rules” Systems: SPARQL engines! 1/2

SPARQL:

- upcoming W3C query language standard
- Actually, SPARQL’s `CONSTRUCT` queries may be viewed as rules as well
- Syntax a bit like merging SQL with N-Triples/Turtle.

```
CONSTRUCT { ?M rdf:type moviShop:BWMovie }
WHERE { ?M rdf:type imd:Movie . ?M imd:year ?Y .
        FILTER (?Y < 1930) }

CONSTRUCT { ?O rdf:type ?C }
WHERE { ?P rdf:range ?C . ?S ?P ?O . }

CONSTRUCT { ?P foaf:knows _:a }
WHERE { ?P rdf:type ex:socialPerson . }
```

Issues:

- No recursive/fixpoint evaluation in standard engines
- No combination of several CONSTRUCTs in standard engines
- BTW: Blank nodes in rule heads (last rule) would make things non-Horn.

Rule Exchange on top of RDF - Syntactical/Semantic Issues

Summary: Now what issues arise for Web Rule exchange?

- Different options for embedding RDF
- Different Syntax (slotted, unary/binary) in different existing systems
- How to embed RDF(S) semantics?
- (Even worse: How to refer to more complicated semantics such as OWL, how to combine/integrate different data/domain models (XML, UML))

But this is not all, also signatures are important...

Rule Exchange on top of RDF - Syntactical/Semantic Issues

Summary: Now what issues arise for Web Rule exchange?

- Different options for embedding RDF
- Different Syntax (slotted, unary/binary) in different existing systems
- How to embed RDF(S) semantics?
- (Even worse: How to refer to more complicated semantics such as OWL, how to combine/integrate different data/domain models (XML, UML))

But this is not all, also signatures are important. . .

Rule Exchange - Signatures/Namespaces

Recall: When hearing about first-order semantics on Monday, you learned about signatures, that is:
Every ruleset or first-order theory uses a particular signature: $\Sigma = (P, F, C, V)$

- P . . . predicate symbols
- F . . . function symbols
- C . . . constant symbols
- V . . . variables

Important for defining a semantics for rules and also for combination/exchange of rulesets!

Ruleset r1:

```
∀ ?X,?Y q(p(?X,?Y),?X) ← q(?Y,?X)
```

Ruleset r2:

```
p("1") ←
```

Could still exchange rules on first-order level, if we know that p in ruleset 1 is something else than p in ruleset 2.

Rule Exchange - Signatures/Namespaces

Recall: When hearing about first-order semantics on Monday, you learned about signatures, that is:
Every ruleset or first-order theory uses a particular signature: $\Sigma = (P, F, C, V)$

P . . . predicate symbols
F . . . function symbols
C . . . constant symbols
V . . . variables

Important for defining a semantics for rules and also for combination/exchange of rulesets!

Ruleset r1:

```
∀ ?X,?Y q(p(?X,?Y),?X) ← q(?Y,?X)
```

Ruleset r2:

```
p("1") ←
```

Could still exchange rules on first-order level, if we know that p in ruleset 1 is something else than p in ruleset 2.

Rule Exchange - Signatures/Namespaces

Recall: When hearing about first-order semantics on Monday, you learned about signatures, that is:
Every ruleset or first-order theory uses a particular signature: $\Sigma = (P, F, C, V)$

P ... predicate symbols
F ... function symbols
C ... constant symbols
V ... variables

Important for defining a semantics for rules and also for combination/exchange of rulesets!

Ruleset r1:

```
∀ ?X,?Y q(p(?X,?Y),?X) ← q(?Y,?X)
```

Ruleset r2:

```
p("1") ←
```

Could still exchange rules on first-order level, if we know that p in ruleset 1 is something else than p in ruleset 2.

Rule Exchange - Signatures/Namespaces

Recall: When hearing about first-order semantics on Monday, you learned about signatures, that is:
Every ruleset or first-order theory uses a particular signature: $\Sigma = (P, F, C, V)$

- P ... predicate symbols
- F ... function symbols
- C ... constant symbols
- V ... variables

Important for defining a semantics for rules and also for combination/exchange of rulesets!

Ruleset r1:

```
∀ ?X,?Y r1:q(r1:p(?X,?Y),?X) ← r1:q(?Y,?X)
```

Ruleset r2:

```
r2:p("1") ←
```

Could still exchange rules on first-order level, if we know that p in ruleset 1 is something else than p in ruleset 2, IRIs/namespaces partially solve that problem.

Current Status of RIF

Now you got an idea of issues which need to be solved for Web rule exchange

. . . Let's finally talk about RIF's current state . . .

2 working drafts produced so far:

- Use Cases and Requirements
- RIF Core Design (now being renamed to "RIF Basic Logic Dialect")

Use Cases and Requirements

- almost 50 use cases for a rule interchange format submitted
- 2 Public Working Drafts of *'RIF Use Cases and Requirements'*
  - use cases from various application domains
  - requirements mainly for Phase I
- a refined Working Draft underway
- we gather Phase II requirements at the moment

RIF Core

- 1st Public Working Draft of *'RIF Core Design'*
- published end of March 2007

Current Status of RIF

Now you got an idea of issues which need to be solved for Web rule exchange

. . . Let’s finally talk about RIF’s current state . . .

2 working drafts produced so far:

- Use Cases and Requirements
- RIF Core Design (now being renamed to “RIF Basic Logic Dialect”)

Use Cases and Requirements

- almost 50 use cases for a rule interchange format submitted
- 2 Public Working Drafts of *’RIF Use Cases and Requirements’*
  - use cases from various application domains
  - requirements mainly for Phase I
- a refined Working Draft underway
- we gather Phase II requirements at the moment

RIF Core

- 1st Public Working Draft of *’RIF Core Design’*
- published end of March 2007

Current Status of RIF

Now you got an idea of issues which need to be solved for Web rule exchange

. . . Let's finally talk about RIF's current state . . .

2 working drafts produced so far:

- Use Cases and Requirements
- RIF Core Design (now being renamed to "RIF Basic Logic Dialect")

Use Cases and Requirements

- almost 50 use cases for a rule interchange format submitted
- 2 Public Working Drafts of *'RIF Use Cases and Requirements'*
  - use cases from various application domains
  - requirements mainly for Phase I
- a refined Working Draft underway
- we gather Phase II requirements at the moment

RIF Core

- 1st Public Working Draft of *'RIF Core Design'*
- published end of March 2007

Current Status of RIF

Now you got an idea of issues which need to be solved for Web rule exchange

. . . Let's finally talk about RIF's current state . . .

2 working drafts produced so far:

- Use Cases and Requirements
- RIF Core Design (now being renamed to "RIF Basic Logic Dialect")

Use Cases and Requirements

- almost 50 use cases for a rule interchange format submitted
- 2 Public Working Drafts of *'RIF Use Cases and Requirements'*
  - use cases from various application domains
  - requirements mainly for Phase I
- a refined Working Draft underway
- we gather Phase II requirements at the moment

RIF Core

- 1st Public Working Draft of *'RIF Core Design'*
- published end of March 2007

Current Status of RIF

Now you got an idea of issues which need to be solved for Web rule exchange

. . . Let's finally talk about RIF's current state . . .

2 working drafts produced so far:

- Use Cases and Requirements
- RIF Core Design (now being renamed to "RIF Basic Logic Dialect")

Use Cases and Requirements

- almost 50 use cases for a rule interchange format submitted
- 2 Public Working Drafts of *'RIF Use Cases and Requirements'*
  - use cases from various application domains
  - requirements mainly for Phase I
- a refined Working Draft underway
- we gather Phase II requirements at the moment

RIF Core

- 1st Public Working Draft of *'RIF Core Design'*
- published end of March 2007

RIF Core Design

RIF Core shall cover the minimal overlap of different Rule dialects, that is

- an extensible formalism to express “basic” conditions
- a simple framework for “basic” rules

⇒ “basic” = **positive Horn** rules

- allow to define rulesets
- provide formal underpinning for
  - interoperation with the remaining Semantic Web architecture
  - extensible semantics for Horn rules and extending dialects

⇒ an extensible **architecture** to build RIF “dialects” around a common Core: This Core Horn dialect will be called RIF Basic Logic Dialect (BLD)

RIF Core Design

RIF Core shall cover the minimal overlap of different Rule dialects, that is

- an extensible formalism to express “basic” conditions
- a simple framework for “basic” rules

⇒ “basic” = **positive Horn** rules

- allow to define rulesets
- provide formal underpinning for
  - interoperation with the remaining Semantic Web architecture
  - extensible semantics for Horn rules and extending dialects

⇒ an extensible **architecture** to build RIF “dialects” around a common Core:
This Core Horn dialect will be called RIF Basic Logic Dialect (BLD)

RIF Core Design

RIF Core shall cover the minimal overlap of different Rule dialects, that is

- an extensible formalism to express “basic” conditions
- a simple framework for “basic” rules

$\Rightarrow$ “basic” = **positive Horn** rules

- allow to define rulesets
- provide formal underpinning for
  - interoperation with the remaining Semantic Web architecture
  - extensible semantics for Horn rules and extending dialects

$\Rightarrow$ an extensible **architecture** to build RIF “dialects” around a common Core:
This Core Horn dialect will be called RIF Basic Logic Dialect (BLD)

RIF Core Design

RIF Core shall cover the minimal overlap of different Rule dialects, that is

- an extensible formalism to express “basic” conditions
- a simple framework for “basic” rules

⇒ “basic” = **positive Horn** rules

- allow to define rulesets
- provide formal underpinning for
  - interoperation with the remaining Semantic Web architecture
  - extensible semantics for Horn rules and extending dialects

⇒ an extensible **architecture** to build RIF “dialects” around a common Core:
This Core Horn dialect will be called RIF Basic Logic Dialect (BLD)

RIF Core Design

RIF Core shall cover the minimal overlap of different Rule dialects, that is

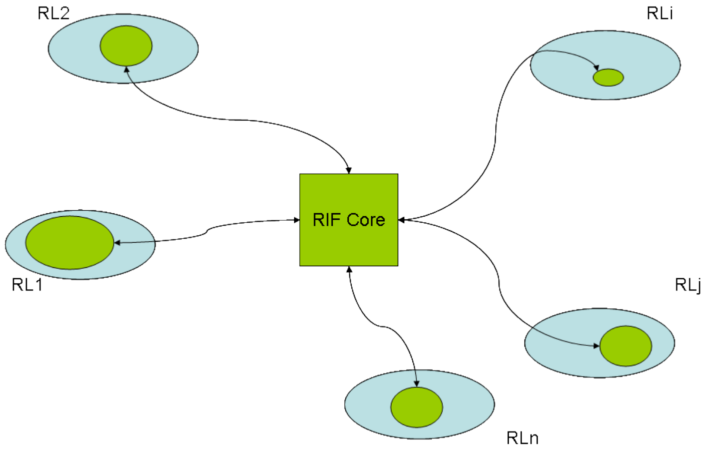
- an extensible formalism to express “basic” conditions
- a simple framework for “basic” rules

⇒ “basic” = **positive Horn** rules

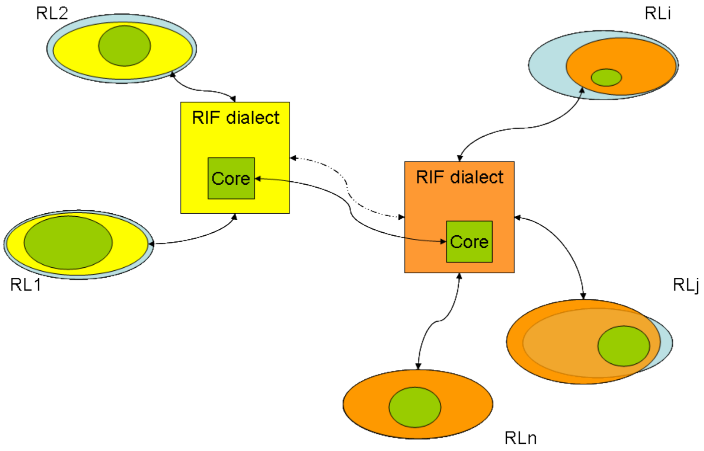
- allow to define rulesets
- provide formal underpinning for
  - interoperation with the remaining Semantic Web architecture
  - extensible semantics for Horn rules and extending dialects

⇒ an extensible **architecture** to build RIF “dialects” around a common Core:
This Core Horn dialect will be called RIF Basic Logic Dialect (BLD)

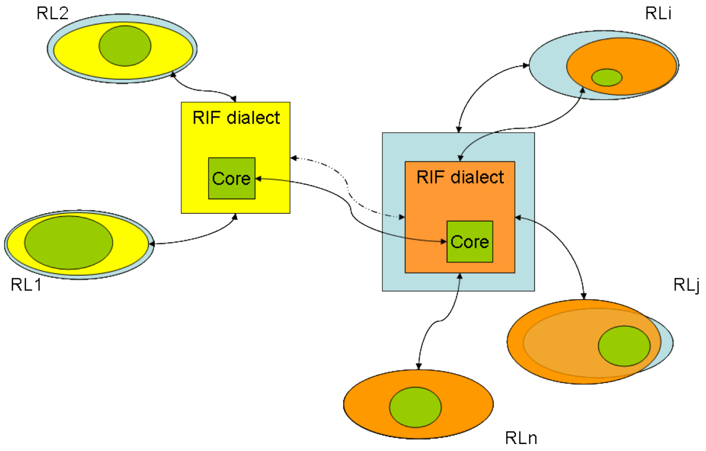
RIF Architecture 1/2



RIF Architecture 1/2



RIF Architecture 1/2



RIF Architecture 2/2

Required:

- Ruleset
  - Annotation: Semantics, Dialect, Name, Description, . . .
- Rule
  - Annotation: Name, Description, . . .
  - Event (`ON`)
  - Condition (`IF`)
  - Conclusion/Derivation (`THEN`)
  - Action (`DO`)
  - . . .

Start with positive Horn:

`IF`: conjunctions (and disjunctions) of atomic conditions

`THEN`: atomic formulae

RIF Architecture 2/2

Required:

- Ruleset
  - Annotation: Semantics, Dialect, Name, Description, ...
- Rule
  - Annotation: Name, Description, ...
  - Event (`ON`)
  - Condition (`IF`)
  - Conclusion/Derivation (`THEN`)
  - Action (`DO`)
  - ...

Start with positive Horn:

`IF`: conjunctions (and disjunctions) of atomic conditions

`THEN`: atomic formulae

RIF Architecture 2/2

Required:

- Ruleset
  - Annotation: Semantics, Dialect, Name, Description, . . .
- Rule
  - Annotation: Name, Description, . . .
  - Event (`ON`)
  - Condition (`IF`)
  - Conclusion/Derivation (`THEN`)
  - Action (`DO`)
  - . . .

Start with positive Horn:

`IF`: conjunctions (and disjunctions) of atomic conditions

`THEN`: atomic formulae

RIF Architecture 2/2

Required:

- Ruleset
    - Annotation: Semantics, Dialect, Name, Description, . . .
- Rule
    - Annotation: Name, Description, . . .
    - Event (`ON`)
    - Condition (`IF`)
    - Conclusion/Derivation (`THEN`)
    - Action (`DO`)
    - . . .

Start with positive Horn:

`IF`: conjunctions (and disjunctions) of atomic conditions

`THEN`: atomic formulae

RIF Architecture 2/2

Required:

- Ruleset
  - Annotation: Semantics, Dialect, Name, Description, . . .
- Rule
  - Annotation: Name, Description, . . .
  - Event (`ON`)
  - Condition (`IF`)
  - Conclusion/Derivation (`THEN`)
  - Action (`DO`)
  - . . .

Start with positive Horn:

`IF`: conjunctions (and disjunctions) of atomic conditions

`THEN`: atomic formulae

$$\text{IF } C_1 \text{ AND } C_2 \text{ AND } \ldots \text{AND } C_n \text{ THEN } A$$

RIF Architecture 2/2

Required:

- Ruleset
  - Annotation: Semantics, Dialect, Name, Description, . . .
- Rule
  - Annotation: Name, Description, . . .
  - Event (`ON`)
  - Condition (`IF`)
  - Conclusion/Derivation (`THEN`)
  - Action (`DO`)
  - . . .

Start with positive Horn:

`IF`: conjunctions (and disjunctions) of atomic conditions

`THEN`: atomic formulae

$$(\forall)\ C_1 \wedge C_2 \wedge \ldots \wedge C_n \rightarrow A$$

RIF Architecture 2/2

Required:

- Ruleset
  - Annotation: Semantics, Dialect, Name, Description, . . .
- Rule
  - Annotation: Name, Description, . . .
  - Event (`ON`)
  - Condition (`IF`)
  - Conclusion/Derivation (`THEN`)
  - Action (`DO`)
  - . . .

Start with positive Horn:

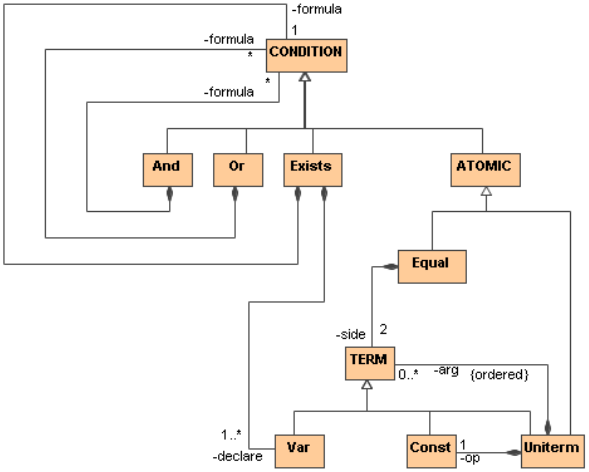
`IF`: conjunctions (and disjunctions) of atomic conditions

`THEN`: atomic formulae

$$(\forall)\ \neg C_1 \lor \neg C_2 \lor \ldots \lor \neg C_n \lor A$$

RIF Core Conditions

An extensible model to express basic conditions:



RIF Core Conditions

An extensible model to express basic conditions:

EBNF Syntax (in progress/under discussion):

```
CONDITION       ::= CONJUNCTION | DISJUNCTION | EXISTENTIAL | ATOMIC
CONJUNCTION     ::= 'And' '(' CONDITION* ')'
DISJUNCTION     ::= 'Or' '(' CONDITION* ')'
EXISTENTIAL     ::= 'Exists' Var+ '(' CONDITION ')'
ATOMIC          ::= Uniterm | Equal | CLASSIFICATION | Frame
Uniterm         ::= Const '(' TERM* ')' | Const '(' (Const '->' TERM)* ')'
Equal           ::= TERM '=' TERM
TERM            ::= Const | Var | Uniterm
Const           ::= CONSTNAME | '"'CONSTNAME'""^^'TYPENAME
Var             ::= '?'VARNAME
```

For instance under discussion: language labels for literals as in RDF (e.g. `"lecture"@en`, `"vorlesung"@de`)

RIF Core Conditions – Example 1/2

Example: `IF` movie ?M was produced before 1930

RIF “readable” version of this condition:

```
Exists ?Y (
    And ( "imd:Movie"( ?M ) "imd:Year"( ?M ?Y )
          "op:date-less-than"( ?Y "1930-01-01T00:00:00Z"^^dateTime )
```

- Names of predicates are “webized” (using URIs and namespaces like in XML and RDF)
- Builtin predicates, like `op:date-less-than` around XPath and XQuery functions and operators will be also standardized (in an extensible way)

RIF Core Conditions – Example 2/2

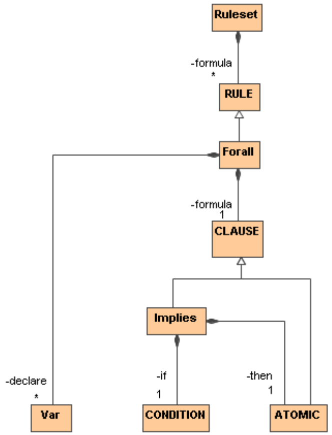
Mock-up XML serialization (currently under discussion):

```
    <Exists>
      <declare><Var>Y</Var></declare>
      <formula>
       <And>
        <formula>
          <Uniterm>
            <Const>Movie</Const>
            <Var>M</Var>
          </Uniterm>
        </formula>
        <formula>
          <Uniterm>
            <Const>Year</Const>
            <Var>M</Var>
            <Var>Y</Var>
          </Uniterm>
        </formula>
        <formula>
          <Uniterm type="builtin">
            <Const>date-less-than</Const>
           <Var>M</Var>
           <Const type="&xsd;dateTime">"1930-01-01T00:00:00</Const>
          </Uniterm>
        </formula>
    </Exists>
  </And>
```

Discussed issues: How to markup typed constants, builtin functions, etc.

RIF Core Horn Rules

A basic model for Horn rules:



Current focus:

- only cover simple `IF-THEN` rules
- provide a clean formal underpinning (model theory)

RIF Core Horn Rule – Example

A rule “local” to a certain DVD shop:

IF dvd ?D shows movie ?M and ?M was produced before 1930

THEN ?M is a black and white movie

```
"moviShop:BWMovie" ( ?M ) :-
  Exists ?D ?Y (
    And ( "moviShop:Dvd"( ?D ) "imd:shows"( ?D ?M )
          "imd:Movie"( ?M ) "imd:Year"( ?M ?Y )
          "op:date-less-than"( ?Y "1930-01-01T00:00:00Z"^^dateTime )
```

- XML syntax similarly discussed
- Keep door open for later extensibility
- Discussions how to integrate with RDF/OWL data and also other data models!

RIF Core Horn Rule – Example

A rule “local” to a certain DVD shop:

`IF` dvd ?D shows movie ?M and ?M was produced before 1930

`THEN` ?M is a black and white movie

```
"moviShop:BWMovie" ( ?M ) :-
  Exists ?D ?Y (
    And ( "moviShop:Dvd"( ?D ) "imd:shows"( ?D ?M )
          "imd:Movie"( ?M ) "imd:Year"( ?M ?Y )
          "op:date-less-than"( ?Y "1930-01-01T00:00:00Z"^^dateTime )
```

- XML syntax similarly discussed
- Keep door open for later extensibility
- Discussions how to integrate with RDF/OWL data and also other data models!

Semantics of RIF Core

General picture

- model-theoretical semantics
- starts with defining the semantics of RIF conditions
- and extends it to RIF (Horn) rules

- RIF dialects are to further extend this semantics
- however, some dialects might not have a model theory (e.g. PR dialect)

Semantics of RIF Core

General picture

- model-theoretical semantics
- starts with defining the semantics of RIF conditions
- and extends it to RIF (Horn) rules = RIF Basic Logic Dialect

- RIF dialects are to further extend this semantics
- however, some dialects might not have a model theory (e.g. PR dialect)

Semantics - RIF Positive Conditions

From the first lecture of this summer school, we know about the notion of interpretation (or semantic structure).

We define a **basic semantic structure** I

- a tuple $<D, I_C, I_V, I_F, I_R>$ that determines the truth value of a formula (`CONDITION` or `CLAUSE` production of EBNF
- D - a non-empty set of elements called the domain of I
- $Const$ - the set of individuals, predicate names, and function symbols
- Var - the set of variables

We denote by TV the set of truth values

- for the RIF BLD it includes only t (true) and f (false)
- TV has a *truth order* $f <_t t$

Semantics - Positive Conditions

... and the mappings are as follows:

- I_C from $Const$ to elements of D
- I_V from Var to elements of D
- I_F from $Const$ to functions from D^* into D
- I_R from $Const$ to truth-valued mappings $D^* \rightarrow TV$

A more general mapping is defined as follows

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

Note that signatures do not appear in the definition of semantic structure!
...But they are important for keeping typical first-order restrictions of RIF's Basic Logic Dialect! more on that later ...

Semantics - Positive Conditions

... and the mappings are as follows:

- I_C from $Const$ to elements of D
- I_V from Var to elements of D
- I_F from $Const$ to functions from D^* into D
- I_R from $Const$ to truth-valued mappings $D^* \rightarrow TV$

A more general mapping is defined as follows

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

Note that signatures do not appear in the definition of semantic structure!

…But they are important for keeping typical first-order restrictions of RIF's Basic Logic Dialect! more on that later …

Semantics - Positive Conditions

... and the mappings are as follows:

- I_C from $Const$ to elements of D
- I_V from Var to elements of D
- I_F from $Const$ to functions from D^* into D
- I_R from $Const$ to truth-valued mappings $D^* \rightarrow TV$

A more general mapping is defined as follows

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

Note that signatures do not appear in the definition of semantic structure!
...But they are important for keeping typical first-order restrictions of RIF's Basic Logic Dialect! more on that later ...

Semantics - Positive Conditions

Truth valuation for formulas determined using I_{Truth}

- atomic formulas: $I_{Truth}(r(t1...tn)) = I_R(r)(I(t1),...,I(tn))$
- equality: $I_{Truth}(t1 = t2) = t$ iff $I(t1) = I(t2)$; $I_{Truth}(t1 = t2) = f$ otherwise
- conjunction: $I_{Truth}(And(c1...cn)) = min_t(I_{Truth}(c1),...,I_{Truth}(cn))$, where min_t is minimum with respect to the truth order
- disjunction: $I_{Truth}(Or(c1...cn)) = max_t(I_{Truth}(c1),...,I_{Truth}(cn))$, where max_t is maximum with respect to the truth order
- quantification: $I_{Truth}(Exists?v1...?vn(c)) = max_t(I^*_{Truth}(c))$, where max_t is taken over all interpretations I^* of the form $<D, I_C, I^*_V, I_F, I_R>$, and the mapping I^*_V has the same value as I_V on all variables except, possibly, on the variables $?v1,...,?vn$.

Semantics - Positive Conditions

Truth valuation for formulas determined using I_{Truth}

- atomic formulas: $I_{Truth}(r(t1 \ldots tn)) = I_R(r)(I(t1), \ldots, I(tn))$
- equality: $I_{Truth}(t1 = t2) = t$ iff $I(t1) = I(t2)$; $I_{Truth}(t1 = t2) = f$ otherwise
- conjunction: $I_{Truth}(And(c1 \ldots cn)) = min_t(I_{Truth}(c1), \ldots, I_{Truth}(cn))$, where min_t is minimum with respect to the truth order
- disjunction: $I_{Truth}(Or(c1 \ldots cn)) = max_t(I_{Truth}(c1), \ldots, I_{Truth}(cn))$, where max_t is maximum with respect to the truth order
- quantification: $I_{Truth}(Exists ?v1 \ldots ?vn(c)) = max_t(I^*_{Truth}(c))$, where max_t is taken over all interpretations I^* of the form $<D, I_C, I^*_V, I_F, I_R>$, and the mapping I^*_V has the same value as I_V on all variables except, possibly, on the variables $?v1, \ldots, ?vn$.

Semantics - Positive Conditions

Truth valuation for formulas determined using I_{Truth}

- atomic formulas: $I_{Truth}(r(t1...tn)) = I_R(r)(I(t1),...,I(tn))$
- equality: $I_{Truth}(t1 = t2) = t$ iff $I(t1) = I(t2)$; $I_{Truth}(t1 = t2) = f$ otherwise
- conjunction: $I_{Truth}(And(c1...cn)) = min_t(I_{Truth}(c1),...,I_{Truth}(cn))$, where min_t is minimum with respect to the truth order
- disjunction: $I_{Truth}(Or(c1...cn)) = max_t(I_{Truth}(c1),...,I_{Truth}(cn))$, where max_t is maximum with respect to the truth order
- quantification: $I_{Truth}(Exists ?v1...?vn(c)) = max_t(I^*_{Truth}(c))$, where max_t is taken over all interpretations I^* of the form $<D, I_C, I^*_V, I_F, I_R>$, and the mapping I^*_V has the same value as I_V on all variables except, possibly, on the variables $?v1,...,?vn$.

Semantics - Positive Conditions

Truth valuation for formulas determined using I_{Truth}

- atomic formulas: $I_{Truth}(r(t1...tn)) = I_R(r)(I(t1),...,I(tn))$
- equality: $I_{Truth}(t1 = t2) = t$ iff $I(t1) = I(t2)$; $I_{Truth}(t1 = t2) = f$ otherwise
- conjunction: $I_{Truth}(And(c1...cn)) = min_t(I_{Truth}(c1),...,I_{Truth}(cn))$, where min_t is minimum with respect to the truth order
- disjunction: $I_{Truth}(Or(c1...cn)) = max_t(I_{Truth}(c1),...,I_{Truth}(cn))$, where max_t is maximum with respect to the truth order
- quantification: $I_{Truth}(Exists?v1...?vn(c)) = max_t(I^*_{Truth}(c))$, where max_t is taken over all interpretations I^* of the form $<D, I_C, I^*_V, I_F, I_R>$, and the mapping I^*_V has the same value as I_V on all variables except, possibly, on the variables $?v1, ..., ?vn$.

Semantics - Positive Conditions

Truth valuation for formulas determined using I_{Truth}

- atomic formulas: $I_{Truth}(r(t1...tn)) = I_R(r)(I(t1),...,I(tn))$
- equality: $I_{Truth}(t1 = t2) = t$ iff $I(t1) = I(t2)$; $I_{Truth}(t1 = t2) = f$ otherwise
- conjunction: $I_{Truth}(And(c1...cn)) = min_t(I_{Truth}(c1),...,I_{Truth}(cn))$, where min_t is minimum with respect to the truth order
- disjunction: $I_{Truth}(Or(c1...cn)) = max_t(I_{Truth}(c1),...,I_{Truth}(cn))$, where max_t is maximum with respect to the truth order
- quantification: $I_{Truth}(Exists?v1...?vn(c)) = max_t(I^*_{Truth}(c))$, where max_t is taken over all interpretations I^* of the form <D, I_C, I^*_V, I_F, I_R>, and the mapping I^*_V has the same value as I_V on all variables except, possibly, on the variables $?v1,...,?vn$.

Semantics - RIF Horn Rules

General form of a RIF rule

$Q\ then\ :\!-\ if$, where Q is the quantification prefix (universal here)

- We first define rule satisfaction without Q

$$I \models then\ :\!-\ if \text{ iff } I_{Truth}(then) >_t I_{Truth}(if)$$

- We define

$$I \models Q\ then\ :\!-\ if \text{ iff } I^* \models then\ :\!-\ if \text{ for every } I^*$$

  where I^* agrees with I everywhere except possibly on some variables mentioned in Q. In this case I is a **model of the given rule.**

- I is a **model of a rule set** R

$$I \models R \text{ if } I \text{ is a semantic structure such that}$$
$$I \models r \text{ for every rule } r \in R$$

Semantics - RIF Horn Rules

General form of a RIF rule

$Q\ then\ :\!-\ if$, where Q is the quantification prefix (universal here)

- We first define rule satisfaction without Q

$$I \models then :\!- if \text{ iff } I_{Truth}(then) >_t I_{Truth}(if)$$

- We define

$$I \models Q\ then :\!- if \text{ iff } I^* \models then :\!- if \text{ for every } I^*$$

  where I^* agrees with I everywhere except possibly on some variables mentioned in Q. In this case I is a **model of the given rule.**
- I is a **model of a rule set** R

$$I \models R \text{ if } I \text{ is a semantic structure such that}$$
$$I \models r \text{ for every rule } r \in R$$

Semantics - RIF Horn Rules

General form of a RIF rule

$Q\ then\ :\!-\ if$, where Q is the quantification prefix (universal here)

- We first define rule satisfaction without Q

$$I \models then\ :\!-\ if \text{ iff } I_{Truth}(then) >_t I_{Truth}(if)$$

- We define

$$I \models Q\ then\ :\!-\ if \text{ iff } I^* \models then\ :\!-\ if \text{ for every } I^*$$

  where I^* agrees with I everywhere except possibly on some variables mentioned in Q. In this case I is a **model of the given rule**.

- I is a **model of a rule set** R

$$I \models R \text{ if } I \text{ is a semantic structure such that}$$
$$I \models r \text{ for every rule } r \in R$$

Semantics - RIF Horn Rules

General form of a RIF rule

$$Q\ then\ :\!-\ if$$, where Q is the quantification prefix (universal here)

- We first define rule satisfaction without Q

$$I \models then\ :\!-\ if \text{ iff } I_{Truth}(then) >_t I_{Truth}(if)$$

- We define

$$I \models Q\ then\ :\!-\ if \text{ iff } I^* \models then\ :\!-\ if \text{ for every } I^*$$

  where I^* agrees with I everywhere except possibly on some variables mentioned in Q. In this case I is a **model of the given rule**.

- I is a **model of a rule set** R

$$I \models R \text{ if } I \text{ is a semantic structure such that}$$
$$I \models r \text{ for every rule } r \in R$$

Semantics - RIF Horn Rules

Entailment of RIF conditions by rule sets

- let S be a RIF rule set and
- ϕ a closed RIF condition (i.e. no free variables)

S **entails** ϕ written as $S \models \phi$

- if for every semantic structure I, such that $I \models S$
- it is the case that $I_{Truth}(\phi) = t$

Semantics - RIF Horn Rules

Entailment of RIF conditions by rule sets

- let S be a RIF rule set and
- ϕ a closed RIF condition (i.e. no free variables)

S **entails** ϕ written as $S \models \phi$

- if for every semantic structure I, such that $I \models S$
- it is the case that $I_{Truth}(\phi) = t$

. . . This sign indicates issues which are being currently discussed and reflect partially personal opinions of WG members!

Slots + Frames

- As we've seen, several rule languages support slots and frames (e.g. F-Logic)
- Considered to model RDF or ontological data, relations with named attributes; often more intuitively than predicates
- Meta-modeling no problem (recall (rdfs3) rule from before!)
- Proposal in the WG from Michael Kifer, Harold Boley

EBNF Syntax:

```
CONDITION      ::= CONJUNCTION | DISJUNCTION | EXISTENTIAL | ATOMIC
CONJUNCTION    ::= 'And' '(' CONDITION* ')'
DISJUNCTION    ::= 'Or' '(' CONDITION* ')'
EXISTENTIAL    ::= 'Exists' Var+ '(' CONDITION ')'
ATOMIC         ::= Uniterm | Equal | CLASSIFICATION | Frame
Uniterm        ::= Const '(' TERM* ')' | Const '(' (Const '->' TERM)* ')'
CLASSIFICATION ::= TERM '#' TERM | TERM '##' TERM
Frame          ::=(TERM | CLASSIFICATION) '[' (TERM '->' (TERM | Frame))* ']'
Equal          ::= TERM '=' TERM
TERM           ::= Const | Var | Uniterm
Const          ::= CONSTNAME | '"'CONSTNAME'"^^'TYPENAME
Var            ::= '?'VARNAME
```

Slots + Frames

- As we've seen, several rule languages support slots and frames (e.g. F-Logic)
- Considered to model RDF or ontological data, relations with named attributes; often more intuitively than predicates
- Meta-modeling no problem (recall (rdfs3) rule from before!)
- Proposal in the WG from Michael Kifer, Harold Boley

EBNF Syntax:

```
CONDITION      ::= CONJUNCTION | DISJUNCTION | EXISTENTIAL | ATOMIC
CONJUNCTION    ::= 'And' '(' CONDITION* ')'
DISJUNCTION    ::= 'Or' '(' CONDITION* ')'
EXISTENTIAL    ::= 'Exists' Var+ '(' CONDITION ')'
ATOMIC         ::= Uniterm | Equal | CLASSIFICATION | Frame
Uniterm        ::= Const '(' TERM* ')' | Const '(' (Const '->' TERM)* ')'
CLASSIFICATION ::= TERM '#' TERM | TERM '##' TERM
Frame          ::=(TERM | CLASSIFICATION) '[' (TERM '->' (TERM | Frame))* ']'
Equal          ::= TERM '=' TERM
TERM           ::= Const | Var | Uniterm
Const          ::= CONSTNAME | '"'CONSTNAME'"''^^'TYPENAME
Var            ::= '?'VARNAME
```

Slots + Frames

- As we've seen, several rule languages support slots and frames (e.g. F-Logic)
- Considered to model RDF or ontological data, relations with named attributes; often more intuitively than predicates
- Meta-modeling no problem (recall (rdfs3) rule from before!)
- Proposal in the WG from Michael Kifer, Harold Boley

EBNF Syntax:

```
CONDITION       ::= CONJUNCTION | DISJUNCTION | EXISTENTIAL | ATOMIC
CONJUNCTION     ::= 'And' '(' CONDITION* ')'
DISJUNCTION     ::= 'Or' '(' CONDITION* ')'
EXISTENTIAL     ::= 'Exists' Var+ '(' CONDITION ')'
ATOMIC          ::= Uniterm | Equal | CLASSIFICATION | Frame
Uniterm         ::= Const '(' TERM* ')' | Const '(' (Const '->' TERM)* ')'
CLASSIFICATION  ::= TERM '#' TERM | TERM '##' TERM
Frame           ::=(TERM | CLASSIFICATION) '[' (TERM '->' (TERM | Frame))* ']'
Equal           ::= TERM '=' TERM
TERM            ::= Const | Var | Uniterm
Const           ::= CONSTNAME | '"'CONSTNAME'""^^'TYPENAME
Var             ::= '?'VARNAME
```

Slots + Frames – Semantics

A **semantic structure**, I, is a tuple of the form

- a tuple $<D, I_C, I_V, I_F, I_R>$
- I_{slot}: from D to truth-valued functions of the form $D \times D \rightarrow TV$
  - Truth valuation: $I_{Truth}(T[p\text{->}V]) = I_{slot}(I(p))(I(T), I(V))$
- I_{SF}: interprets terms with named arguments
  - $I(f(p_1\text{->}t_1 \ldots p_n\text{->}t_n)) = I_{SF}(f)(\{\langle I(p_1), I(t_1)\rangle, \ldots\ldots, \langle I(p_n), I(t_n)\rangle\})$
  - Here, each pair $\langle s, v\rangle \in D \times D$ represents a slot name-value pair
- I_{SR}: interprets predicates with slotted arguments
  - $I_{Truth}(p(p_1\text{->}val_1 \ldots p_k\text{->}t_k)) = I_{SR}(p)(\{\langle I(p_1)I(t_1)\rangle, \ldots, \langle I(p_k)I(t_k)\rangle\})$
- I_{sub}: gives meaning to the subclass relationship
  - $I_{Truth}(sc\#\#cl) = I_{sub}(I(sc), I(cl))$
  - Additionally, in all allowed interpretations this is an axiom:
    ```
    ?C1##?C3 :- Exists ?C2 (And ( ?C1##?C2 ?C2##?C3 ) )
    ```
- I_{isa}: gives meaning to class membership
  - $I_{Truth}(o\#cl) = I_{isa}(I(o), I(cl))$
  - Additionally, in all allowed interpretations this is an axiom:
    ```
    ?X#?C2 :- Exists ?C1 (And ( ?X#?C1 ?C1##?C2 ) )
    ```

Slots + Frames – Semantics

A **semantic structure**, I, is a tuple of the form

- a tuple $<D, I_C, I_V, I_F, I_R, I_{slot}, I_{SF}, I_{SR}, I_{sub}, I_{isa}>$
- I_{slot}: from D to truth-valued functions of the form $D \times D \rightarrow TV$
  - Truth valuation: $I_{Truth}(T[p{-}{>}V]) = I_{slot}(I(p))(I(T), I(V))$
- I_{SF}: interprets terms with named arguments
  - $I(f(p_1{-}{>}t_1 \ldots p_n{-}{>}t_n)) = I_{SF}(f)(\{\langle I(p_1), I(t_1)\rangle, \ldots\ldots, \langle I(p_n), I(t_n)\rangle\})$
  - Here, each pair $\langle s, v\rangle \in D \times D$ represents a slot name-value pair
- I_{SR}: interprets predicates with slotted arguments
  - $I_{Truth}(p(p_1{-}{>}val_1 \ldots p_k{-}{>}t_k)) = I_{SR}(p)(\{\langle I(p_1) I(t_1)\rangle, \ldots, \langle I(p_k) I(t_k)\rangle\})$
- I_{sub}: gives meaning to the subclass relationship
  - $I_{Truth}(sc\#\#cl) = I_{sub}(I(sc), I(cl))$
  - Additionally, in all allowed interpretations this is an axiom:
    ```
    ?C1##?C3 :- Exists ?C2 (And ( ?C1##?C2 ?C2##?C3 ) )
    ```
- I_{isa}: gives meaning to class membership
  - $I_{Truth}(o\#cl) = I_{isa}(I(o), I(cl))$
  - Additionally, in all allowed interpretations this is an axiom:
    ```
    ?X#?C2 :- Exists ?C1 (And ( ?X#?C1 ?C1##?C2 ) )
    ```

Slots + Frames – Semantics

A **semantic structure**, I, is a tuple of the form

- a tuple $<D, I_C, I_V, I_F, I_R, I_{slot}, I_{SF}, I_{SR}, I_{sub}, I_{isa}>$
- I_{slot}: from D to truth-valued functions of the form $D \times D \rightarrow TV$
  - Truth valuation: $I_{Truth}(T[p\text{->}V]) = I_{slot}(I(p))(I(T), I(V))$
- I_{SF}: interprets terms with named arguments
  - $I(f(p_1\text{->}t_1 ... p_n\text{->}t_n)) = I_{SF}(f)(\{\langle I(p_1), I(t_1)\rangle, \ldots\ldots, \langle I(p_n), I(t_n)\rangle\})$
  - Here, each pair $\langle s, v\rangle \in D \times D$ represents a slot name-value pair
- I_{SR}: interprets predicates with slotted arguments
  - $I_{Truth}(p(p_1\text{->}val_1 ... p_k\text{->}t_k)) = I_{SR}(p)(\{\langle I(p_1) I(t_1)\rangle, \ldots, \langle I(p_k) I(t_k)\rangle\})$
- I_{sub}: gives meaning to the subclass relationship
  - $I_{Truth}(sc\#\#cl) = I_{sub}(I(sc), I(cl))$
  - Additionally, in all allowed interpretations this is an axiom:
    ```
    ?C1##?C3 :- Exists ?C2 (And ( ?C1##?C2 ?C2##?C3 ) )
    ```
- I_{isa}: gives meaning to class membership
  - $I_{Truth}(o\#cl) = I_{isa}(I(o), I(cl))$
  - Additionally, in all allowed interpretations this is an axiom:
    ```
    ?X#?C2 :- Exists ?C1 (And ( ?X#?C1 ?C1##?C2 ) )
    ```

Slots + Frames – Semantics

A **semantic structure**, I, is a tuple of the form

- a tuple $<D, I_C, I_V, I_F, I_R, I_{slot}, I_{SF}, I_{SR}, I_{sub}, I_{isa}>$
- I_{slot}: from D to truth-valued functions of the form $D \times D \rightarrow TV$
  - Truth valuation: $I_{Truth}(T[p{-}{>}V]) = I_{slot}(I(p))(I(T), I(V))$
- I_{SF}: interprets terms with named arguments
  - $I(f(p_1{-}{>}t_1 ... p_n{-}{>}t_n)) = I_{SF}(f)(\{\langle I(p_1), I(t_1)\rangle, \ldots\ldots, \langle I(p_n), I(t_n)\rangle\})$
  - Here, each pair $\langle s, v\rangle \in D \times D$ represents a slot name-value pair
- I_{SR}: interprets predicates with slotted arguments
  - $I_{Truth}(p(p_1{-}{>}val_1 ... p_k{-}{>}t_k)) = I_{SR}(p)(\{\langle I(p_1) I(t_1)\rangle, \ldots, \langle I(p_k) I(t_k)\rangle\})$
- I_{sub}: gives meaning to the subclass relationship
  - $I_{Truth}(sc\#\#cl) = I_{sub}(I(sc), I(cl))$
  - Additionally, in all allowed interpretations this is an axiom:
    ```
    ?C1##?C3 :- Exists ?C2 (And ( ?C1##?C2 ?C2##?C3 ) )
    ```
- I_{isa}: gives meaning to class membership
  - $I_{Truth}(o\#cl) = I_{isa}(I(o), I(cl))$
  - Additionally, in all allowed interpretations this is an axiom:
    ```
    ?X#?C2 :- Exists ?C1 (And ( ?X#?C1 ?C1##?C2 ) )
    ```

Slots + Frames – Semantics

A **semantic structure**, I, is a tuple of the form

- a tuple $<D, I_C, I_V, I_F, I_R, I_{slot}, I_{SF}, I_{SR}, I_{sub}, I_{isa}>$
- I_{slot}: from D to truth-valued functions of the form $D \times D \rightarrow TV$
  - Truth valuation: $I_{Truth}(T[p\text{-}{>}V]) = I_{slot}(I(p))(I(T), I(V))$
- I_{SF}: interprets terms with named arguments
  - $I(f(p_1\text{-}{>}t_1 ... p_n\text{-}{>}t_n)) = I_{SF}(f)(\{\langle I(p_1), I(t_1)\rangle, \ldots ..., \langle I(p_n), I(t_n)\rangle\})$
  - Here, each pair $\langle s, v\rangle \in D \times D$ represents a slot name-value pair
- I_{SR}: interprets predicates with slotted arguments
  - $I_{Truth}(p(p_1\text{-}{>}val_1 ... p_k\text{-}{>}t_k)) = I_{SR}(p)(\{\langle I(p_1) I(t_1)\rangle, ..., \langle I(p_k) I(t_k)\rangle\})$
- I_{sub}: gives meaning to the subclass relationship
  - $I_{Truth}(sc\#\#cl) = I_{sub}(I(sc), I(cl))$
  - Additionally, in all allowed interpretations this is an axiom:
    ```
    ?C1##?C3 :- Exists ?C2 (And ( ?C1##?C2 ?C2##?C3 ) )
    ```
- I_{isa}: gives meaning to class membership
  - $I_{Truth}(o\#cl) = I_{isa}(I(o), I(cl))$
  - Additionally, in all allowed interpretations this is an axiom:
    ```
    ?X#?C2 :- Exists ?C1 (And ( ?X#?C1 ?C1##?C2 ) )
    ```

Slots + Frames – Semantics

A **semantic structure**, I, is a tuple of the form

- a tuple $<D, I_C, I_V, I_F, I_R, I_{slot}, I_{SF}, I_{SR}, I_{sub}, I_{isa}>$
- I_{slot}: from D to truth-valued functions of the form $D \times D \to TV$
  - Truth valuation: $I_{Truth}(T[p\text{->}V]) = I_{slot}(I(p))(I(T), I(V))$
- I_{SF}: interprets terms with named arguments
  - $I(f(p_1\text{->}t_1 ... p_n\text{->}t_n)) = I_{SF}(f)(\{\langle I(p_1), I(t_1)\rangle, ..., \langle I(p_n), I(t_n)\rangle\})$
  - Here, each pair $\langle s, v\rangle \in D \times D$ represents a slot name-value pair
- I_{SR}: interprets predicates with slotted arguments
  - $I_{Truth}(p(p_1\text{->}val_1 ... p_k\text{->}t_k)) = I_{SR}(p)(\{\langle I(p_1) I(t_1)\rangle, ..., \langle I(p_k) I(t_k)\rangle\})$
- I_{sub}: gives meaning to the subclass relationship
  - $I_{Truth}(sc\#\#cl) = I_{sub}(I(sc), I(cl))$
  - Additionally, in all allowed interpretations this is an axiom:

```
?C1##?C3 :- Exists ?C2 (And ( ?C1##?C2 ?C2##?C3 ) )
```

- I_{isa}: gives meaning to class membership
  - $I_{Truth}(o\#cl) = I_{isa}(I(o), I(cl))$
  - Additionally, in all allowed interpretations this is an axiom:

```
?X#?C2 :- Exists ?C1 (And ( ?X#?C1 ?C1##?C2 ) )
```

Slots + Frames – Semantics

A **semantic structure**, I, is a tuple of the form

- a tuple $<D, I_C, I_V, I_F, I_R, I_{slot}, I_{SF}, I_{SR}, I_{sub}, I_{isa}>$
- I_{slot}: from D to truth-valued functions of the form $D \times D \rightarrow TV$
  - Truth valuation: $I_{Truth}(T[p{-}{>}V]) = I_{slot}(I(p))(I(T), I(V))$
- I_{SF}: interprets terms with named arguments
  - $I(f(p_1{-}{>}t_1 ... p_n{-}{>}t_n)) = I_{SF}(f)(\{\langle I(p_1), I(t_1)\rangle, \ldots\ldots, \langle I(p_n), I(t_n)\rangle\})$
  - Here, each pair $\langle s, v\rangle \in D \times D$ represents a slot name-value pair
- I_{SR}: interprets predicates with slotted arguments
  - $I_{Truth}(p(p_1{-}{>}val_1 ... p_k{-}{>}t_k)) = I_{SR}(p)(\{\langle I(p_1) I(t_1)\rangle, \ldots, \langle I(p_k) I(t_k)\rangle\})$
- I_{sub}: gives meaning to the subclass relationship
  - $I_{Truth}(sc\#\#cl) = I_{sub}(I(sc), I(cl))$
  - Additionally, in all allowed interpretations this is an axiom:
    ```
    ?C1##?C3 :- Exists ?C2 (And ( ?C1##?C2 ?C2##?C3 ) )
    ```
- I_{isa}: gives meaning to class membership
  - $I_{Truth}(o\#cl) = I_{isa}(I(o), I(cl))$
  - Additionally, in all allowed interpretations this is an axiom:
    ```
    ?X#?C2 :- Exists ?C1 (And ( ?X#?C1 ?C1##?C2 ) )
    ```

Slots + Frames – Semantics

A **semantic structure**, I, is a tuple of the form

- a tuple $<D, I_C, I_V, I_F, I_R, I_{slot}, I_{SF}, I_{SR}, I_{sub}, I_{isa}>$
- I_{slot}: from D to truth-valued functions of the form $D \times D \rightarrow TV$
  - Truth valuation: $I_{Truth}(T[p\text{->}V]) = I_{slot}(I(p))(I(T), I(V))$
- I_{SF}: interprets terms with named arguments
  - $I(f(p_1\text{->}t_1 \ldots p_n\text{->}t_n)) = I_{SF}(f)(\{\langle I(p_1), I(t_1)\rangle, \ldots\ldots, \langle I(p_n), I(t_n)\rangle\})$
  - Here, each pair $\langle s, v\rangle \in D \times D$ represents a slot name-value pair
- I_{SR}: interprets predicates with slotted arguments
  - $I_{Truth}(p(p_1\text{->}val_1 \ldots p_k\text{->}t_k)) = I_{SR}(p)(\{\langle I(p_1)I(t_1)\rangle, \ldots, \langle I(p_k)I(t_k)\rangle\})$
- I_{sub}: gives meaning to the subclass relationship
  - $I_{Truth}(sc\#\#cl) = I_{sub}(I(sc), I(cl))$
  - Additionally, in all allowed interpretations this is an axiom:
    ```
    ?C1##?C3 :- Exists ?C2 (And ( ?C1##?C2 ?C2##?C3 ) )
    ```
- I_{isa}: gives meaning to class membership
  - $I_{Truth}(o\#cl) = I_{isa}(I(o), I(cl))$
  - Additionally, in all allowed interpretations this is an axiom:
    ```
    ?X#?C2 :- Exists ?C1 (And ( ?X#?C1 ?C1##?C2 ) )
    ```

Signatures

- Under discussion how to model signatures
- Aim: Generalization of first-order signatures
- how to define/restrict what may appear in function/predicate/constant positions
- whether or not same symbol is allowed with different arities
- whether or not complex terms are allowed as term constructors
- etc.

Signatures

- Under discussion how to model signatures
- Aim: Generalization of first-order signatures
- how to define/restrict what may appear in function/predicate/constant positions
- whether or not same symbol is allowed with different arities
- whether or not complex terms are allowed as term constructors
- etc.

Signatures

- Under discussion how to model signatures
- Aim: Generalization of first-order signatures
- how to define/restrict what may appear in function/predicate/constant positions
- whether or not same symbol is allowed with different arities
- whether or not complex terms are allowed as term constructors
- etc.

Signatures

- Under discussion how to model signatures
- Aim: Generalization of first-order signatures
- how to define/restrict what may appear in function/predicate/constant positions
- whether or not same symbol is allowed with different arities
- whether or not complex terms are allowed as term constructors
- etc.

Signatures

- Under discussion how to model signatures
- Aim: Generalization of first-order signatures
- how to define/restrict what may appear in function/predicate/constant positions
- whether or not same symbol is allowed with different arities
- whether or not complex terms are allowed as term constructors
- etc.

RDF Compatibility

As mentioned in examples before, several options to embed RDF into RIF.[2]

Overall idea

```
embed(rdfset1) ---> rif-entailed ---> embed(rdfset2)
   ^                                      ^
   |                                      |
   |                                      |
   |                                      |
   |                                      |
rdfset1 ---------- rdf-entailed ----> rdfset2
```

[2] Some ASCII art from a WG mail from Michael Kifer from yesterday ;-)

RDF Compatibility – example RDFS embedding

RDF semantics defines three semantic “flavors”

- simple RDF (define only equivalence between two RDF graphs modulo blank node renaming)
- RDF (takes RDF vocabulary into account)
- RDFS (takes RDFS vocabulary into account)

Idea: All embeddable in RIF by kind of “axiomatic” rulesets

RDF Compatibility – example RDFS embedding

RDF semantics defines three semantic “flavors”

- simple RDF (define only equivalence between two RDF graphs modulo blank node renaming)
- RDF (takes RDF vocabulary into account)
- RDFS (takes RDFS vocabulary into account)

Idea: All embeddable in RIF by kind of “axiomatic” rulesets

RDF Compatibility – example RDFS embedding

RDF entailment embedding (recent proposal by WG member Jos de Bruijn[3]):

R^{RDF}	=	(`Forall` tr(`s p o .`)) for every RDF axiomatic triple `s p o .`) union
		(`Forall ?x ?x[rdf:type -> rdf:Property] :- Exists ?y,?z (?y[?x -> ?z]),`
		`Forall ?x ?x[rdf:type -> rdf:XMLLiteral] :- wellxml(?x)`)

C^{RDF}	=	(`Exists ?x (And(?x[rdf:type -> rdf:XMLLiteral] illxml(?x))`)

- R^{RDF} . . . a set of RIF axiomatic deductive rules
- C^{RDF} . . . normative rule which *must not* be entailed (constraint on the data)
- Here, fixed interpretation (often called "built-in") predicates `wellxml` and `illxml` are assumed.
  - BTW: How to define, in general, built-in predicates is another issue, many rule languages and systems provide these.

[3] `http://www.w3.org/2005/rules/wg/wiki/Core/RIF-RDF_Compatibility`

RDF Compatibility – example RDFS embedding

RDF entailment embedding (recent proposal by WG member Jos de Bruijn[3]):

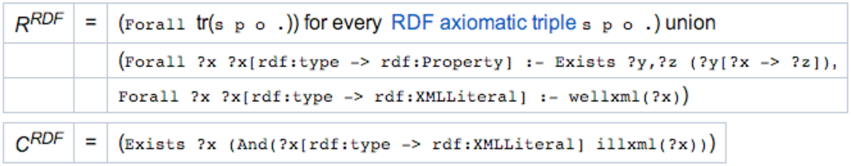
R^{RDF}	=	(`Forall` tr(`s p o .`)) for every RDF axiomatic triple `s p o .`) union
		(`Forall ?x ?x[rdf:type -> rdf:Property] :- Exists ?y,?z (?y[?x -> ?z]),`
		`Forall ?x ?x[rdf:type -> rdf:XMLLiteral] :- wellxml(?x)`)
C^{RDF}	=	(`Exists ?x (And(?x[rdf:type -> rdf:XMLLiteral] illxml(?x))`)

- R^{RDF} . . . a set of RIF axiomatic deductive rules
- C^{RDF} . . . normative rule which *must not* be entailed (constraint on the data)
- Here, fixed interpretation (often called “built-in”) predicates `wellxml` and `illxml` are assumed.
  - BTW: How to define, in general, built-in predicates is another issue, many rule languages and systems provide these.

[3] `http://www.w3.org/2005/rules/wg/wiki/Core/RIF-RDF_Compatibility`

RDF Compatibility – example RDFS embedding

RDF entailment embedding (recent proposal by WG member Jos de Bruijn[3]):



R^{RDF}	=	(`Forall` tr(`s p o .`)) for every RDF axiomatic triple `s p o .`) union
		(`Forall ?x ?x[rdf:type -> rdf:Property] :- Exists ?y,?z (?y[?x -> ?z]),`
		`Forall ?x ?x[rdf:type -> rdf:XMLLiteral] :- wellxml(?x)`)
C^{RDF}	=	(`Exists ?x (And(?x[rdf:type -> rdf:XMLLiteral] illxml(?x))`)

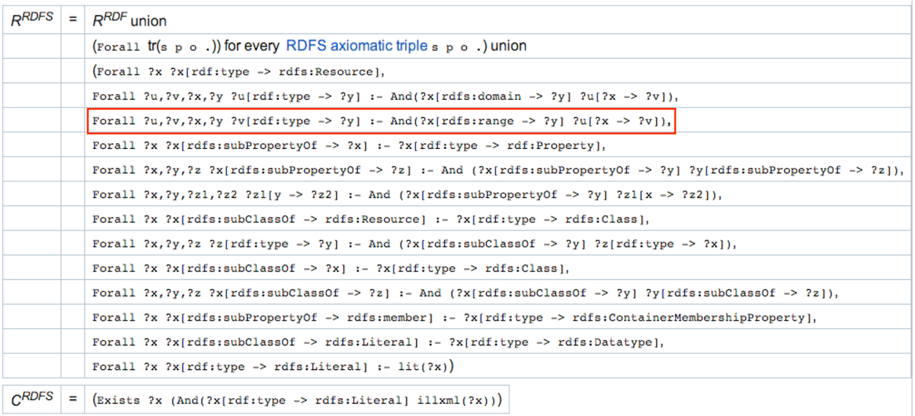
- R^{RDF} . . . a set of RIF axiomatic deductive rules
- C^{RDF} . . . normative rule which *must not* be entailed (constraint on the data)

- Here, fixed interpretation (often called "built-in") predicates `wellxml` and `illxml` are assumed.
  - BTW: How to define, in general, built-in predicates is another issue, many rule languages and systems provide these.

[3]`http://www.w3.org/2005/rules/wg/wiki/Core/RIF-RDF_Compatibility`

RDF Compatibility – example RDFS embedding

RDFS entailment embedding:



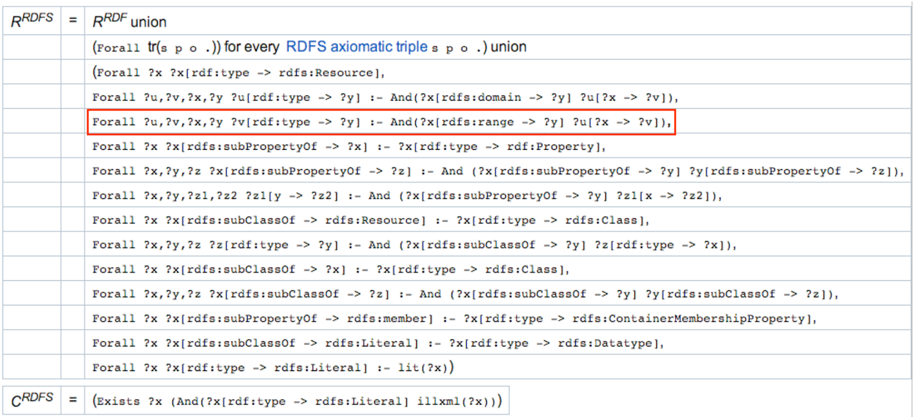
R^{RDFS}	=	R^{RDF} union
		(`Forall` tr(`s p o .`)) for every RDFS axiomatic triple `s p o .`) union
		(`Forall ?x ?x[rdf:type -> rdfs:Resource],`
		`Forall ?u,?v,?x,?y ?u[rdf:type -> ?y] :- And(?x[rdfs:domain -> ?y] ?u[?x -> ?v]),`
		`Forall ?u,?v,?x,?y ?v[rdf:type -> ?y] :- And(?x[rdfs:range -> ?y] ?u[?x -> ?v]),`
		`Forall ?x ?x[rdfs:subPropertyOf -> ?x] :- ?x[rdf:type -> rdf:Property],`
		`Forall ?x,?y,?z ?x[rdfs:subPropertyOf -> ?z] :- And (?x[rdfs:subPropertyOf -> ?y] ?y[rdfs:subPropertyOf -> ?z]),`
		`Forall ?x,?y,?z1,?z2 ?z1[y -> ?z2] :- And (?x[rdfs:subPropertyOf -> ?y] ?z1[x -> ?z2]),`
		`Forall ?x ?x[rdfs:subClassOf -> rdfs:Resource] :- ?x[rdf:type -> rdfs:Class],`
		`Forall ?x,?y,?z ?z[rdf:type -> ?y] :- And (?x[rdfs:subClassOf -> ?y] ?z[rdf:type -> ?x]),`
		`Forall ?x ?x[rdfs:subClassOf -> ?x] :- ?x[rdf:type -> rdfs:Class],`
		`Forall ?x,?y,?z ?x[rdfs:subClassOf -> ?z] :- And (?x[rdfs:subClassOf -> ?y] ?y[rdfs:subClassOf -> ?z]),`
		`Forall ?x ?x[rdfs:subPropertyOf -> rdfs:member] :- ?x[rdf:type -> rdfs:ContainerMembershipProperty],`
		`Forall ?x ?x[rdfs:subClassOf -> rdfs:Literal] :- ?x[rdf:type -> rdfs:Datatype],`
		`Forall ?x ?x[rdf:type -> rdfs:Literal] :- lit(?x)`)
C^{RDFS}	=	(`Exists ?x (And(?x[rdf:type -> rdfs:Literal] illxml(?x))`)

rule rdfs3 from our previous examples marked here.

Such a simple embedding is not possible for OWL of course!

RDF Compatibility – example RDFS embedding

RDFS entailment embedding:



R^{RDFS}	=	R^{RDF} union
		(Forall tr(s p o .)) for every RDFS axiomatic triple s p o .) union
		(Forall ?x ?x[rdf:type -> rdfs:Resource],
		Forall ?u,?v,?x,?y ?u[rdf:type -> ?y] :- And(?x[rdfs:domain -> ?y] ?u[?x -> ?v]),
		Forall ?u,?v,?x,?y ?v[rdf:type -> ?y] :- And(?x[rdfs:range -> ?y] ?u[?x -> ?v]),
		Forall ?x ?x[rdfs:subPropertyOf -> ?x] :- ?x[rdf:type -> rdf:Property],
		Forall ?x,?y,?z ?x[rdfs:subPropertyOf -> ?z] :- And (?x[rdfs:subPropertyOf -> ?y] ?y[rdfs:subPropertyOf -> ?z]),
		Forall ?x,?y,?z1,?z2 ?z1[y -> ?z2] :- And (?x[rdfs:subPropertyOf -> ?y] ?z1[x -> ?z2]),
		Forall ?x ?x[rdfs:subClassOf -> rdfs:Resource] :- ?x[rdf:type -> rdfs:Class],
		Forall ?x,?y,?z ?z[rdf:type -> ?y] :- And (?x[rdfs:subClassOf -> ?y] ?z[rdf:type -> ?x]),
		Forall ?x ?x[rdfs:subClassOf -> ?x] :- ?x[rdf:type -> rdfs:Class],
		Forall ?x,?y,?z ?x[rdfs:subClassOf -> ?z] :- And (?x[rdfs:subClassOf -> ?y] ?y[rdfs:subClassOf -> ?z]),
		Forall ?x ?x[rdfs:subPropertyOf -> rdfs:member] :- ?x[rdf:type -> rdfs:ContainerMembershipProperty],
		Forall ?x ?x[rdfs:subClassOf -> rdfs:Literal] :- ?x[rdf:type -> rdfs:Datatype],
		Forall ?x ?x[rdf:type -> rdfs:Literal] :- lit(?x))
C^{RDFS}	=	(Exists ?x (And(?x[rdf:type -> rdfs:Literal] illxml(?x)))

rule rdfs3 from our previous examples marked here.

Such a simple embedding is not possible for OWL of course!

RDF Compatibility – example RDFS embedding

RDFS entailment embedding:

R^{RDFS}	=	R^{RDF} union
		(Forall tr(s p o .)) for every RDFS axiomatic triple s p o .) union
		(Forall ?x ?x[rdf:type -> rdfs:Resource],
		Forall ?u,?v,?x,?y ?u[rdf:type -> ?y] :- And(?x[rdfs:domain -> ?y] ?u[?x -> ?v]),
		Forall ?u,?v,?x,?y ?v[rdf:type -> ?y] :- And(?x[rdfs:range -> ?y] ?u[?x -> ?v]),
		Forall ?x ?x[rdfs:subPropertyOf -> ?x] :- ?x[rdf:type -> rdf:Property],
		Forall ?x,?y,?z ?x[rdfs:subPropertyOf -> ?z] :- And (?x[rdfs:subPropertyOf -> ?y] ?y[rdfs:subPropertyOf -> ?z]),
		Forall ?x,?y,?z1,?z2 ?z1[y -> ?z2] :- And (?x[rdfs:subPropertyOf -> ?y] ?z1[x -> ?z2]),
		Forall ?x ?x[rdfs:subClassOf -> rdfs:Resource] :- ?x[rdf:type -> rdfs:Class],
		Forall ?x,?y,?z ?z[rdf:type -> ?y] :- And (?x[rdfs:subClassOf -> ?y] ?z[rdf:type -> ?x]),
		Forall ?x ?x[rdfs:subClassOf -> ?x] :- ?x[rdf:type -> rdfs:Class],
		Forall ?x,?y,?z ?x[rdfs:subClassOf -> ?z] :- And (?x[rdfs:subClassOf -> ?y] ?y[rdfs:subClassOf -> ?z]),
		Forall ?x ?x[rdfs:subPropertyOf -> rdfs:member] :- ?x[rdf:type -> rdfs:ContainerMembershipProperty],
		Forall ?x ?x[rdfs:subClassOf -> rdfs:Literal] :- ?x[rdf:type -> rdfs:Datatype],
		Forall ?x ?x[rdf:type -> rdfs:Literal] :- lit(?x))
C^{RDFS}	=	(Exists ?x (And(?x[rdf:type -> rdfs:Literal] illxml(?x)))

rule rdfs3 from our previous examples marked here.

Such a simple embedding is not possible for OWL of course!

Towards a PR Dialect for RIF

We have now

- a (still ongoing work on a) core interchange format and
- a strong interest in extending it with PRs

... and (thus) a first proposal for a **PR dialect** for RIF

- first steps towards a RIF dialect for Production Rules
- extends the existing RIF Core
- possibility to *retract* facts

Towards a PR Dialect for RIF

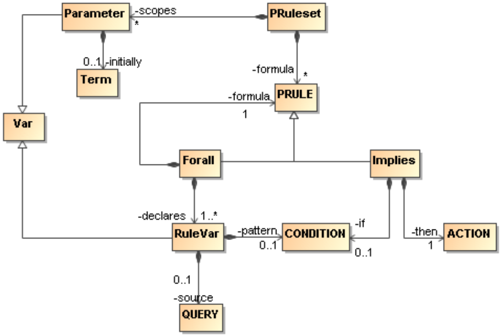
We have now

- a (still ongoing work on a) core interchange format and
- a strong interest in extending it with PRs

... and (thus) a first proposal for a **PR dialect** for RIF

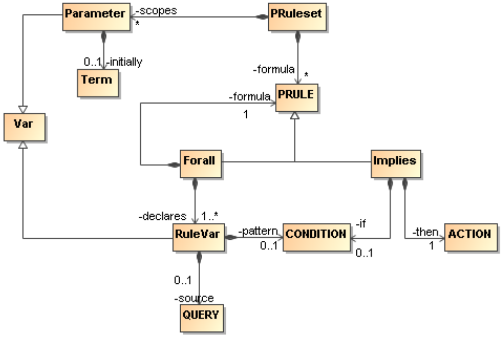
- first steps towards a RIF dialect for Production Rules
- extends the existing RIF Core
- possibility to *retract* facts

Towards a PR Dialect for RIF



- *then* part (head) of rules specifies an action
- *Parameter* subclass of *Var*, *Term* gives its initial value
- *RuleVar* extends *Var* by *source* and *pattern*
    - for its valuation domain

Towards a PR Dialect for RIF



- *then* part (head) of rules specifies an action
- *Parameter* subclass of *Var*, *Term* gives its initial value
- *RuleVar* extends *Var* by *source* and *pattern*
  - for its valuation domain

Towards a PR Dialects for RIF

Current status

- very simple actions (assert and retract facts)
- no proposal for concrete syntax
- semantics not yet specified
- as RIF Core, the proposal for the PR dialect is ongoing work
- (probably) focus of Phase II work

No other RIF dialect under development at moment within the W3C RIF WG!

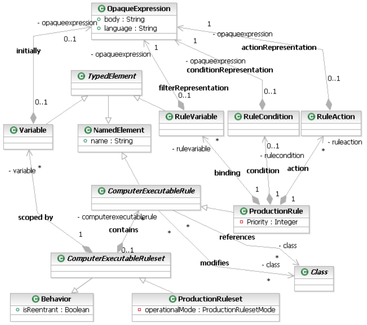
Towards a PR Dialects for RIF

Current status

- very simple actions (assert and retract facts)
- no proposal for concrete syntax
- semantics not yet specified
- as RIF Core, the proposal for the PR dialect is ongoing work
- (probably) focus of Phase II work

No other RIF dialect under development at moment within the W3C RIF WG!

Related Efforts: W3C RIF and OMG PRR



- Alignment definitely desirable
- Alignment with related efforts in W3C (and not only) via so-called "*Liaisons*"

Concluding Remarks

- Developing a useful format for rules on the Web
  - is a challenging and time-consuming task
  - different communities (e.g. PR vendors, Semantic Web researchers) are interested in it
- First steps towards a simple and extensible core format
  - for interchanging derivation rules
  - published as RIF Core in a 1st Working Draft of W3C
- More interesting and useful extensions to RIF Core in the near future
- ... follow the work at `http://www.w3.org/2005/rules/`

This presentation: Axel Polleres (DERI Galway), Paula Pătrânjan (REWERSE)

(both W3C WORLD WIDE WEB consortium member organisations)