

# Relationships Between Logic Programming and RDF\*

Harold Boley

DFKI GmbH  
boley@dfki.de

**Abstract.** Mutual relationships between logic programming and RDF are examined. Basic RDF is formalized with ground binary Datalog Horn facts. Containers are modeled using ('active') polyadic constructors. For meta-statements a modal-logic treatment is suggested. To reduce large fact sets, an "inferential RDF", with special Horn rules, is introduced. A direct representation of non-binary relations is proposed. Reification is thus abandoned and RDF diagrams are generalized using hypergraphs. RDF types are considered as sort predicates. RDF Schema's class/property hierarchies are regarded as a second-order subsumes/subsumes2 syntax or as simple Horn rules. Its domain/range constraints are extended to (polymorphic) signatures. All concepts are explained via knowledge-representation examples as usable by information agents.

## 1 Introduction

The Resource Description Framework (RDF) [LS99,BG00] has been standardized by the W3C mainly to capture metadata about arbitrary resources addressed by URIs/URLs. It is thus intended for encoding a **resource index**, as e.g. needed for information localization by semantic search engines and for information interoperation by broker agents. However, since the boundary between metadata and data is not an absolute one, RDF can also be used for encoding a **knowledge base**, as e.g. needed for question answering. Moreover, one can use an RDF-like encoding as a combination of a knowledge base, e.g. answering some questions directly, and a resource index, e.g. finding further helpful documents [Bol99]. RDF may be regarded as an initial language for the *Semantic Web* [BLF99] [<http://www.w3.org/2001/sw/>].

RDF in the narrow sense [LS99] [<http://www.w3.org/TR/REC-rdf-syntax/>] consists of a directed-labeled-graph or triple-set model with full and abridged XML serialization syntaxes. It is broadened by RDF Schema

---

\* I would like to thank Leon Sterling, Seng Wai Loke, and the program committee of the 1st Pacific Rim International Workshop on Intelligent Information Agents for inviting me to this article. It is based on preparatory research done for the EU project CLOCKWORK. Special thanks for valuable RDF discussions and proof-reading go to Michael Sintek.

[BG00] [<http://www.w3.org/TR/rdf-schema/>], itself specified in RDF syntax, for (pre)categorizing RDF nodes and arc labels. RDF was thus influenced by work in knowledge representation (KR) such as semantic nets, frame systems, and logic languages.

It may therefore be worthwhile to again look at RDF from a KR perspective. Here, we focus on relationships between RDF/XML and logic programming (LP); the companion papers [<http://www.dfki.uni-kl.de/~boley/xmlp-engl.ps>] and [Bol01] deal with XML-LP comparisons. To cover the diagrammatic aspects of (extended) RDF, we will employ (hyper)graphs as discussed for KR in [Bol92]: While the current RDF standard – in the tradition of simple semantic nets – focusses on binary relations visualized as arcs, this paper moves on towards a more direct treatment of non-binary relations, demonstrating their direct visualization as hyperarcs (cf. appendix A).

More generally, we will show trade-offs between the simplicity of the representation language – e.g. RDF’s binary Horn facts – and the complexity of representation constructions in such a language – e.g. RDF’s need for reification to represent containers, meta-statements, Horn rules, and N-ary relations. For RDF Schema, we will discuss first-order reductions of the (useful) second-order syntactic sugar of class/property hierarchies and extend domain/range constraints to polymorphic signatures.

## 2 Basic RDF Becomes Simple Horn Facts

Let us start with a version of the **informal** XML sample sentence from section 2 in [Bol01]:

```
<sentence> Onoffbook sold 12417 copies of XML4You online </sentence>
```

As in that paper we can proceed to its **semiformal** XML triple element representation:

```
<triple>
  <subject> Onoffbook </subject>
  <predicate> sold online </predicate>
  <object> 12417 copies of XML4You </object>
</triple>
```

But now let us assume the bookstore Onoffbook and the book XML4You are considered as ‘resources’ represented by their URIs/URLs. Then, as a new third stage, we obtain the **formal** XML element consisting of a graph of triple subelements:<sup>1</sup>

<sup>1</sup> The third triple, briefly ("<http://www.onoffbook.com/bookId/00498>", quantity, 12417), stands for “the number of "<http://www.onoffbook.com/bookId/00498>" items sold is 12417”. This would be clearer if combined with the first triple into a single statement, foreshadowing the need for non-binary predicates as discussed in section 6.

```

<graph>
  <triple>
    <subject resource="http://www.onoffbook.com"/>
    <predicate>online-sales</predicate>
    <object resource="http://www.onoffbook.com/bookId/00498"/>
  </triple>
  <triple>
    <subject resource="http://www.onoffbook.com/bookId/00498"/>
    <predicate>name</predicate>
    <object>XML4You</object>
  </triple>
  <triple>
    <subject resource="http://www.onoffbook.com/bookId/00498"/>
    <predicate>quantity</predicate>
    <object>12417</object>
  </triple>
</graph>

```

Resources (in RDF always in the subject position and sometimes in the object position) are represented by a URL attribute of empty elements. Literals (in RDF sometimes in the object position) become ordinary (PCDATA-)content elements.

In general, a `graph` element is an XML representation of a directed labeled graph (DLG) via a set of triples,  $\{ \dots, (\text{source-node}, \text{arc-label}, \text{target-node}), \dots \}$ , marked up as

```

<graph>
  ...
  <triple> source-node' arc-label' target-node' </triple>
  ...
</graph>

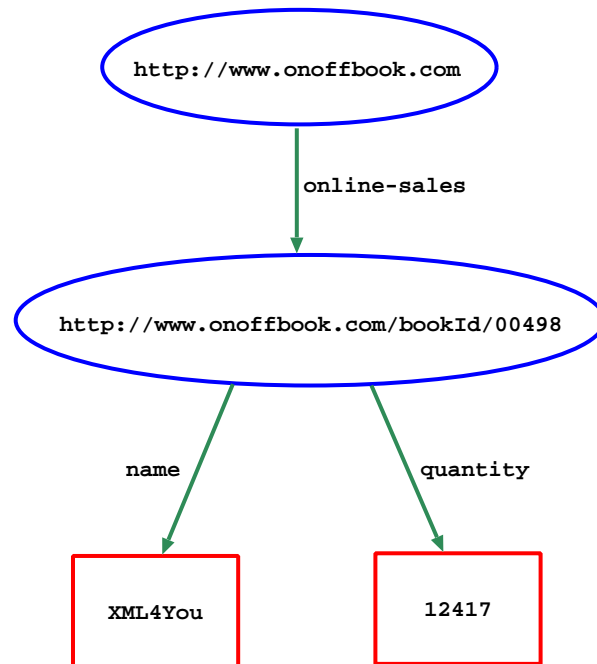
```

where the primes indicate further source (subject) and target (object) node as well as label (predicate) markup.

Our example corresponds to the following DLG diagram, which already happens to be an RDF diagram:<sup>2</sup>

---

<sup>2</sup> Because of other diagram features such as the different shapes for resources and literals, the colors of the online version will not be essential here and later on.



Its above XML triple-graph serialization becomes full (unabridged) RDF/XML via a join of triples having the same subject and an attribute renaming in the subject position (*resource* into *about*) together with some further element restructuring/renaming (using XML's *namespace:localname* tags):

```
<rdf:RDF>
  <rdf:Description about="http://www.onoffbook.com">
    <s:online-sales
      rdf:resource="http://www.onoffbook.com/bookId/00498"/>
  </rdf:Description>
  <rdf:Description about="http://www.onoffbook.com/bookId/00498">
    <v:name>XML4You</v:name>
    <v:quantity>12417</v:quantity>
  </rdf:Description>
</rdf:RDF>
```

In LP, the earlier triple form can be compactly rewritten as binary relations over URI/URL and ordinary constants, which (when enriched by XML namespaces) also capture the RDF/XML version:

```
online-sales("http://www.onoffbook.com",
            "http://www.onoffbook.com/bookId/00498").
name("http://www.onoffbook.com/bookId/00498",xml4you).
quantity("http://www.onoffbook.com/bookId/00498",12417).
```

Since each such RDF re-representation leads to a finite set of ground Datalog Horn facts, this also illustrates how the RDF model, which is a *data model*, can be semantically construed as simple *Herbrand models*, in the sense of model theory [Llo87] (giving up the *unique name assumption* for multiple-URL individuals): A set of ground facts like the above actually is its own Herbrand model.

Of course, these three RDF/XML-derived Prolog facts could again be represented as XML elements, e.g. in the manner of section 4 in [Bol01] or as in RuleML [<http://www.dfki.de/ruleml>]. Since the subject of an RDF statement must be a resource, the first argument of each such fact must be a URI/URL; in RuleML we thus specified a “UR-centered” DTD for RDF-like languages [<http://www.dfki.de/ruleml/inrdf.html>].

### 3 Containers Lead to Generalized Structures

In RDF containers are treated by a kind of reification, with a new anonymous resource node standing for a container as a whole, and special arcs leading to its elements. In LP, we can avoid these ‘new nodes’ by using generalized Prolog structures applying a polyadic (N-ary,  $N \geq 0$ ) and possibly ‘active’ constructor to the N container elements. The corresponding diagram versions will similarly avoid reification by using *hyperarcs*, connecting N nodes [Bol92] (cf. appendix A).

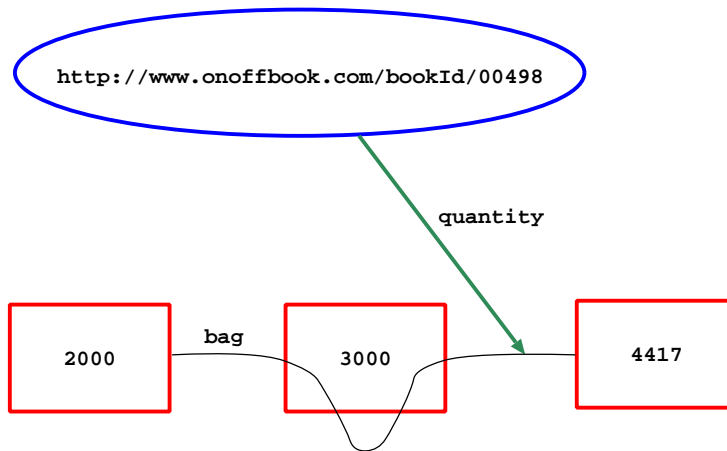
*Sequence* just becomes the list (or tuple) datatype of N ordered elements with repetitions, i.e. a **tup** constructor applied to N arguments. This may be diagrammed as a directed **tup**-labeled hyperarc similar to unasserted arcs in section 4.

*Bag* becomes a multiset datatype of N unordered elements with repetitions. It employs an N-ary constructor, **bag**, that disregards the argument order, e.g. by sorting ground arguments lexicographically (and using AC-unification for non-ground arguments). This may be diagrammed as an undirected **bag**-labeled hyperarc or as a complex **bag**-labeled node containing labeled nodes. While both diagram forms can cope with the duplicates permitted within bags we here prefer undirected hyperarcs in order to avoid labeled nodes.

In our example suppose the **quantity** is split into a sum of four unspecified (temporal or regional) subquantities, where two or more subquantities may be identical. This leads to a **bag** structure like **bag**[2000,3000,3000,4417] usable in facts such as

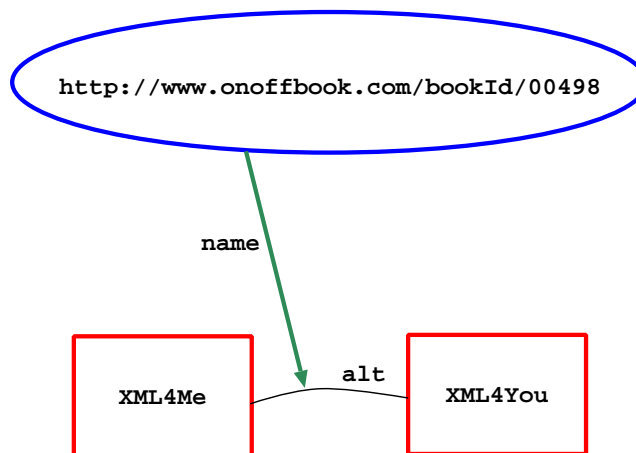
```
quantity("http://www.onoffbook.com/bookId/00498",
        bag[2000,3000,3000,4417]).
```

It is diagrammed as the following undirected (no arrow head) **bag**-labeled hyperarc with a duplicate 3000 node (cut twice by the hyperarc):



*Alternative* may be similarly formalized as a kind of disjunctive datatype and diagrammed as an undirected `alt`-labeled hyperarc or as a complex `alt`-labeled node containing unlabeled nodes. For uniformity we again prefer undirected hyperarcs.

In our example suppose the name has two alternatives, `XML4You` and `XML4Me`. This leads to the `alt` structure `alt[xml4me,xml4you]` usable in facts such as `name("http://www.onoffbook.com/bookId/00498",alt[xml4me,xml4you])`. This is diagrammed as the following `alt`-labeled undirected hyperarc (which happens to be an arc):



However, if we want `alt` to ‘distribute out of’ certain contexts, ‘absorb’ failures in its arguments, or exhibit similar behavior, it begins to look more like a control structure than like a data structure. In this view `alt` may exhibit Prolog’s

notion of “*don’t know*” *non-determinism* or committed-choice languages’ notion of “*don’t care*” *non-determinism*, where the latter, search-free non-determinism may be preferable in the open Web environment of RDF.

RDF’s *distributive referents* use an `aboutEach` attribute for property ‘distribution into’ containers. They could be represented in LP by reinterpreting `aboutEach` as a universal quantifier over structures.

## 4 Meta-Statements via Modal Operators

RDF meta-statements (statements about statements) are based on *reified statements* similar to our original `triple` formalization in section 2, with an additional property specifying that the described new resource has an `rdf:type` of `Statement`. Such a reified statement can then become the target of another statement using a further property, `a:attributedTo`. For example, we could specify that Onoffbook themselves make the `online-sales` claim as follows:

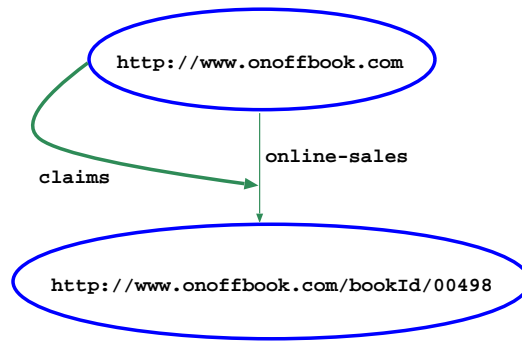
```
<rdf:RDF>
  <rdf:Description>
    <rdf:subject resource="http://www.onoffbook.com"/>
    <rdf:predicate resource=
      "http://description.org/schema/online-sales"/>
    <rdf:object resource="http://www.onoffbook.com/bookId/00498"/>
    <rdf:type resource=
      "http://www.w3.org/1999/02/22-rdf-syntax-ns#Statement"/>
    <a:attributedTo resource="http://www.onoffbook.com"/>
  </rdf:Description>
</rdf:RDF>
```

However, a modal-logic treatment as indicated in section 5 of [Bol01] with a belief operator would more directly capture the intended semantics. Here we similarly employ a `claims` statement – corresponding to the inverse of `a:attributedTo` – about an `online-sales` statement:

```
claims("http://www.onoffbook.com",
       online-sales("http://www.onoffbook.com",
                    "http://www.onoffbook.com/bookId/00498")).
```

The DLG representation can be accommodated to such modal statements as follows: I. Allow nodes and node-connecting (level 1) arcs as the source and/or target of (level 2) arcs, allow all of these in (level 3) arcs, etc., to any level of modal nesting. II. Introduce two weights for arcs, where heavy arrows denote asserted arcs (expressing true statements) while light arrows denote unasserted arcs (needed to express other statements).

In the example, the heavy `claims` (level 2) arc has the `"http://www.onoffbook.com"` node as the source and the light `online-sales` (level 1) arc from there to the `"http://www.onoffbook.com/bookId/00498"` node as the target:



In this DLG extension it is easy to switch a given arc between light and heavy. Thus, after verifying Onoffbook’s sales claim, we need just toggle the light arc to heavy for asserting both Onoffbook’s claim and its truth (lightness and heaviness of arcs do not affect the accessing higher-level arcs). In LP, this requires an additional copy of the embedded `online-sales` as the fact from section 2. In RDF it even requires a completely differently represented non-reified statement. Reified and non-reified statement versions may be confusing since the epistemological status of ‘unused’ reified statements is unclear. Here we regard a top-level light arc as a spurious construction, which should lead to user notification calling for either top-level deletion or heavy-making, where the graph that pops up after top-level deletion will be re-checked recursively.

## 5 Inferential RDF Through Horn Rules

Suppose we want to use the RDF/XML example from section 2 – enriched by price information (cf. section 6) etc. – as metadata to describe some other data such as bookshop portals for use by, say, comparison-shopping agents. Since metadata should change more slowly than data, the exact quantity (= 12417) of books (here: XML4You) sold is clearly inappropriate. Some magnitude interval (say  $> 10000, \leq 20000$ ) would be preferable. To achieve this, in the LP formulation it would be easy to generalize the above `quantity` fact to a `magnitude` rule (the builtin “<”, called here in prefix notation, in full Prolog could be replaced by a user-defined `less-than` relation, e.g. via successor structures):

```

magnitude("http://www.onoffbook.com/bookId/00498",Int) :-
    <(10000,Int), <(Int,20001).
  
```

As one example out of 10000, this rule can infer the result for the goal `magnitude(Url,12417)` via `<(10000,12417), <(12417,20001)`, by binding its `Url` variable to `"http://www.onoffbook.com/bookId/00498"`. More completely, the Herbrand model would now include this set of 10000 ground facts:

```

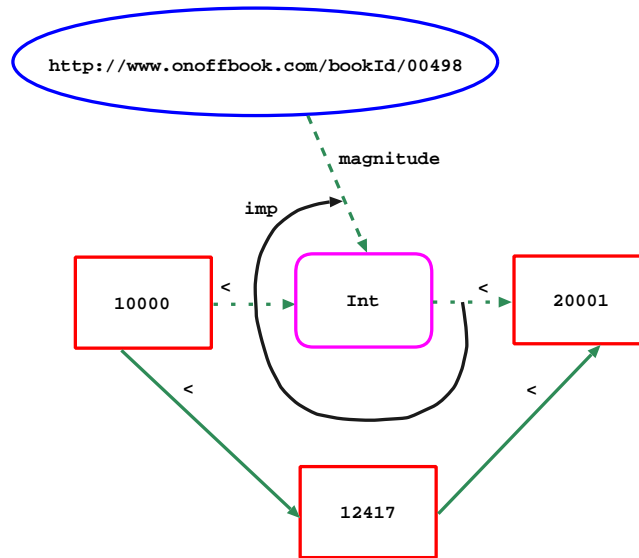
magnitude("http://www.onoffbook.com/bookId/00498",10001).
magnitude("http://www.onoffbook.com/bookId/00498",10002).
. . .
magnitude("http://www.onoffbook.com/bookId/00498",20000).
  
```

To obtain an unbounded interval one could omit the rule's second premise, leading to a Herbrand model with an infinite set of ground Horn facts.

Rules could again be marked up in XML, e.g. as shown in section 4 of [Bol01], for textually ordered Prolog rules, or as in RuleML (cf. end of section 2).

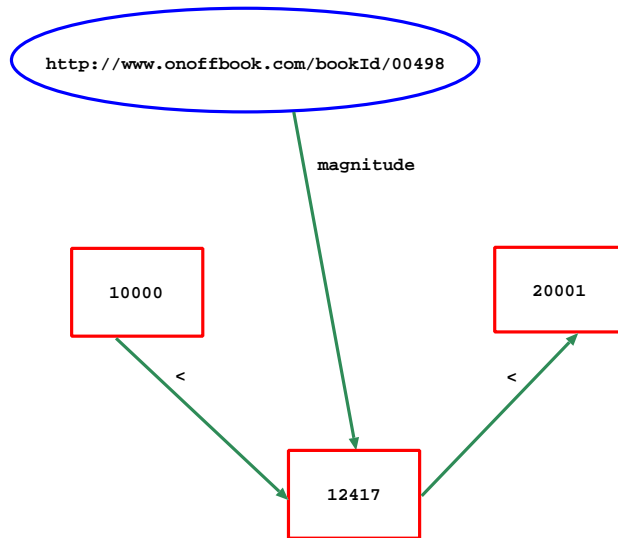
As an extension of (RDF) DLG diagrams, a rule can be depicted with dotted and dashed arcs for, respectively, premises and the conclusion, and with a generalized directed hyperarc labeled `implies` connecting a (Prolog-like) ordered sequence of premise arcs<sup>3</sup> with the conclusion arc. Logic variables like `Int` are depicted by rounded rectangles, whose names should be unique in a given namespace: since we do not allow node labels, the logical uniqueness of variables coincides with the graph-theoretical uniqueness of nodes.

Our example rule, augmented by the two premise facts `<(10000,12417)` and `<(12417,20001)`, thus corresponds to the following extended diagram:



For applying this rule to the premise facts the variable `Int` is unified with the constant `12417`. With the binding `Int = 12417` both “<” premises can then be verified. Therefore, the conclusion `magnitude("http://www.onoffbook.com/bookId/00498",12417)` is inferred. As a fact, it can again be depicted as a solid arc in a basic (RDF) diagram:

<sup>3</sup> The premises could thus also be represented by the RDF container `Sequence`; alternatively, in a more declarative representation, the premises would constitute an unordered multiset, represented by the RDF container `Bag`; cf. section 3. However, the properties of logical conjunction would require a container `Set` (N unordered elements without repetitions), still absent from RDF [LS99].



Such recursionless Datalog Horn rules – corresponding to relational views – would thus naturally enhance the expressive power of basic RDF, arriving at an inferential RDF. It is less obvious, however, where to stop in the expressiveness hierarchy towards recursive pure Prolog Horn rules or beyond (incl. versions of negation) for the purposes of RDF. Here, in section 3 we dealt with generalized Prolog constructors (for containers), in section 4 considered modal-logic operators (for meta-statements), and in the following section 6 will treat Prolog predicates (for N-ary relations).

## 6 Non-Binary Relations, Logically

The RDF data model intrinsically only supports binary relations, and the recommended technique to deal with higher-arity relations is using “an intermediate resource with additional properties of this resource giving the remaining relations” [LS99]. This resource can be regarded as the reification of a higher-arity relationship. The two examples used in [http://www.w3.org/TR/REC-rdf-syntax/] both have a ternary flavor. Continuing our example in a similar manner, we could specify the book price in AUstralian Dollars as follows:

```
<rdf:RDF>
  <Description about="http://www.onoffbook.com/bookId/00498">
    <n:price>
      <rdf:value>39.5</rdf:value>
      <n:units rdf:resource=
        "http://www.rba.gov.au/about/ab_monpol.html"/>
    </n:price>
  </Description>
</rdf:RDF>
```

In LP, we could directly reflect such a formalization, but it seems to be preferable to apply the unit as a constructor, `aud`, to its value, `39.5`, obtaining a Prolog-like structure. This, then, becomes the complex second argument of a binary `price` relation:

```
price("http://www.onoffbook.com/bookId/00498",aud[39.5]).
```

Actually, it would be clearer to distinguish a binary **relation** such as `online-sales` from a unary **attribute** such as `price`, which in functional-logic programming could be defined as a unary function returning a `price` value (as detailed in ONTOFILE [Bol99]):

```
price("http://www.onoffbook.com/bookId/00498") = aud[39.5]
```

Thus, these examples are not typical for N-ary relations. In a general non-binary relation the arguments cannot be grouped into a top-level pair in a natural way. Nor can they generally be split in another natural way, as e.g. – assuming a (Lisp-like) polyadic “<” – the ternary `<(10000,12417,20001)` into `<(10000,12417)`, `<(12417,20001)` from section 5. For example, consider a ternary `ships` (sender-freight-receiver) relation with relationships such as the following (`john` could be a third URL):

```
ships("http://www.onoffbook.com",
      "http://www.onoffbook.com/bookId/00498",
      john).
```

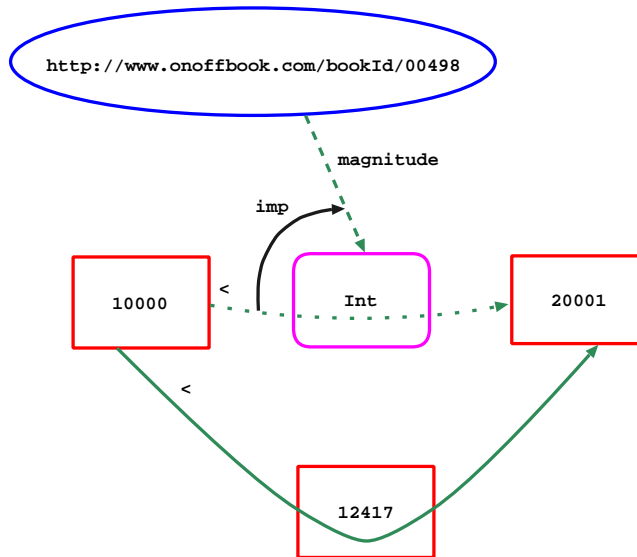
The reduction of such relationships to binary relationships would require unnatural reifications of the kind criticized in [Bol92] [<http://www.dfki.uni-kl.de/~boley/drlhops.abs.html>].

To avoid this, an N-ary extension of basic RDF can be used, where an R-labeled directed hyperarc connects the ordered sequence of N arguments of a relation R. The diagram in section 2 could, e.g., be extended by a `ships`-labeled hyperarc from the Onoffbook URL to John (cf. appendix A).

This can be combined with the rule extension of section 5. For example, a ternary “<” use permits joining the two premises of our previous rule into one, obtaining the rule

```
magnitude("http://www.onoffbook.com/bookId/00498",Int) :-
    <(10000,Int,20001).
```

Its diagram, augmented by the premise fact `<(10000,12417,20001)`., can be depicted thus:



## 7 RDF Types as Sort Predicates

In section 4 we already made use of `rdf:type` for the peculiar case of reified `Statements`. In general, using `rdf:type` properties, a resource can be specified to be an instance of one or more classes. In our RDF/XML example from section 2 we can specify the resource `"http://www.onoffbook.com"` to be an instance of class `Bookstore` and the resource `"http://www.onoffbook.com/bookId/00498"` to be an instance of class `Book`:

```
<rdf:RDF>
  <rdf:Description about="http://www.onoffbook.com">
    <rdf:type resource="http://description.org/schema/Bookstore"/>
    <s:online-sales rdf:resource="http://www.onoffbook.com/bookId/00498"/>
  </rdf:Description>
  <rdf:Description about="http://www.onoffbook.com/bookId/00498">
    <rdf:type resource="http://description.org/schema/Book"/>
    <v:name>XML4You</v:name>
    <v:quantity>12417</v:quantity>
  </rdf:Description>
</rdf:RDF>
```

Let us assume here that the classes `Bookstore` and `Book` are linked to their schema definition from a central place (rather than from each occurrence). Then, in LP, they could be used as special unary relations applied to resources to represent the above type declarations:

```
bookstore("http://www.onoffbook.com").
book("http://www.onoffbook.com/bookId/00498").
```

Such special unary predicates in sorted logics are called *sorts*, and are used for variable typing.<sup>4</sup> The above unary ground facts extensionally characterize (‘leaf’) sorts, like the A-box of description logics. The next section will intensionally characterize (‘inner’ and ‘leaf’) sorts, like their T-box.

## 8 RDF Schema Core Properties in Second-Order Syntax

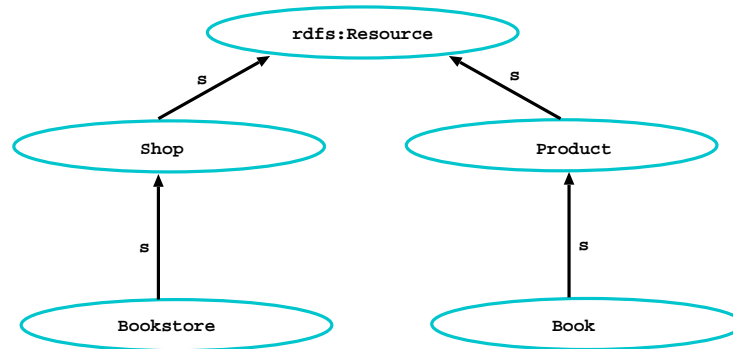
Let us now proceed to RDF Schema. It can be used to organize resources of type `Class` in a hierarchical fashion via a `subClassOf` element. For example, below a `Resource` root, we can specify that a `Bookstore` is a `Shop` and a `Book` is a `Product` as follows:

```
<rdf:RDF>
  <rdf:Description ID="Shop">
    <rdf:type resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
    <rdfs:subClassOf
      rdf:resource="http://www.w3.org/2000/01/rdf-schema#Resource"/>
  </rdf:Description>
  <rdf:Description ID="Product">
    <rdf:type resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
    <rdfs:subClassOf
      rdf:resource="http://www.w3.org/2000/01/rdf-schema#Resource"/>
  </rdf:Description>
  <rdf:Description ID="Bookstore">
    <rdf:type resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
    <rdfs:subClassOf rdf:resource="#Shop"/>
  </rdf:Description>
  <rdf:Description ID="Book">
    <rdf:type resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
    <rdfs:subClassOf rdf:resource="#Product"/>
  </rdf:Description>
</rdf:RDF>
```

We use a simplified version of the corresponding RDF Schema diagram, assuming all classes (all nodes) point to "`http://www.w3.org/2000/01/rdf-schema#Class`" instead of repeating these `t(type)` links, and only show the `s(subClassOf)` links:

---

<sup>4</sup> RDF's permission of an instance  $I$  having multiple types  $T_1, T_2, \dots, T_k$  poses problems to object-oriented/frame systems (if  $I$  is a new instance of, say, class/frame  $T_1$ , classes/frames  $T_2, \dots, T_k$  must also be instantiatable to it) and to sort systems (an  $I$ -constrained variable would require a ‘dynamic’ type intersection  $T_1 \sqcap T_2 \sqcap \dots \sqcap T_k$ ). Since it crosses the class-instance boundary, such *multiclass membership* is harder than the *multiple subclassing/inheritance* considered in section 8. Multiclass membership can be simulated via multiple subclassing by ‘statically’ creating a new named subclass like  $T = T_1 \sqcap T_2 \sqcap \dots \sqcap T_k$ , as in description logics [<http://dl.kr.org/>], of which  $I$  can then be a member.



In general, RDF Schema hierarchies need not form a tree but may constitute a directed acyclic graph (DAG). This partial order is usable for inheritance (DAGs: multiple inheritance) over classes in RDF applications.

In LP, it can be captured via a sorted logic and some second-order syntax, in the example essentially (omitting the namespaces `rdf/rdfs` and the repeated types) using the `subsumes` relationships between super- and subsorts on the left:

<code>subsumes(resource,shop).</code>	<code>resource(X) :- shop(X).</code>
<code>subsumes(resource,product).</code>	<code>resource(X) :- product(X).</code>
<code>subsumes(shop,bookstore).</code>	<code>shop(X) :- bookstore(X).</code>
<code>subsumes(product,book).</code>	<code>product(X) :- book(X).</code>

These “second-order facts” abbreviate the first-order (Horn) rules on the right. However, the normal use of `subsumes` relationships is for sort-subsumption checking, not for backward deduction.

RDF Schema also uses a `subPropertyOf` element, for hierarchically organizing properties. For example, we can specify that `online-sales` and `offline-sales` are sales, thus:

```

<rdf:RDF>
  <rdf:Description ID="online-sales">
    <rdf:type resource=
      "http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
    <rdfs:subPropertyOf rdf:resource="#sales"/>
  </rdf:Description>
  <rdf:Description ID="offline-sales">
    <rdf:type resource=
      "http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
    <rdfs:subPropertyOf rdf:resource="#sales"/>
  </rdf:Description>
</rdf:RDF>
  
```

We omit here a `subPropertyOf` diagram, which would be quite similar to the above `subClassOf` diagram. This information can be used by RDF applications for inheritance over properties.

In LP, it may again be captured via second-order syntax, in the example essentially (without namespaces etc.) using the `subsumes2` relationships between binary super- and subrelations on the left (using a degenerate `subsumes0`, and with the previous `subsumes` standing for `subsumes1`, the subsumption between N-ary relations could be generally expressed via a `subsumesN`):

```
subsumes2(sales,online-sales).    sales(S,P) :- online-sales(S,P).
subsumes2(sales,offline-sales).  sales(S,P) :- offline-sales(S,P).
```

These “second-order facts” abbreviate the first-order (Horn) rules on the right. The `subsumes2` relationships, however, can be used in several ways, including constraint checking (cf. section 9) and backward deduction.

## 9 RDF Schema Core Constraints as Signatures

RDF Schema employs `domain` and `range` constraints to specify the classes on whose instances a property can be used. For example, we can specify that `sales` has a `Shop` domain and a `Product` range as follows:

```
<rdf:RDF>
  <rdf:Description ID="sales">
    <rdf:type resource=
      "http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
    <rdfs:domain rdf:resource="#Shop"/>
    <rdfs:range rdf:resource="#Product"/>
  </rdf:Description>
</rdf:RDF>
```

In LP, this can be expressed (ignoring namespaces etc.) by signatures, which are written here as fact-like “`^`”-declarations applying, e.g., binary relations to their domain and range “`$`”-sorts [Bol99]:

```
sales($shop,$product)^
```

While RDF Schema does not explicitly demand it, RDF applications should certainly have `domain` and `range` constraints inherited from super- to subproperties (cf. section 8). For example, in our LP notation, the above signature declaration, together with the previous `subsumes2` declaration, should imply two further signature declarations:

```
online-sales($shop,$product)^
offline-sales($shop,$product)^
```

Unlike for RDF Schema properties, in LP one could also use *polymorphic* relations, further constraining certain domain-range pairs such as `bookstore` with `book` or `drugstore` with `drug`:

```
sales($bookstore,$book)^
sales($drugstore,$drug)^
```

For a shop like "<http://www.onoffbook.com>", which is a bookstore, this would mean that its product sales must actually be book sales.

Sort polymorphism can be combined with property inheritance, in our example finally implying these signature declarations:

```
online-sales($bookstore,$book)^
offline-sales($bookstore,$book)^
online-sales($drugstore,$drug)^
offline-sales($drugstore,$drug)^
```

## 10 Conclusions

This paper showed how RDF's syntaxes and diagrams can be reconsidered as a special case of knowledge representation with logic programs and hypergraph diagrams. For several representation problems we suggested RDF extensions, e.g. to permit a more direct treatment of non-binary relations and Horn rules. We discussed Herbrand semantics for facts and rules but did not go into semantics for modal logics and other LP extensions. For another formalized treatment of RDF in a KR context see [Cha00] [<http://www710.univ-lyon1.fr/~champin/rdf-tutorial/>], giving a logical account of RDF inheritance and constraints, and discussing RDF implementation issues of containers, self-reference, etc.

RDF queries and inferences have been implemented with techniques from LP, e.g. building on F-logic in SiLRI [DBSA97] and TRIPLE [ABvE<sup>+</sup>01]. Efficient subsumption checking for ontologies in RDF syntax is provided by the FaCT implementation of description logics [Hea00]. RDF inferencing based on tractable graph algorithms is implemented in Euler [<ftp://windsor.agfa.be/outgoing/RCEI/NET/euler/index.html>]. Several further approaches are indexed at "Mozilla RDF / Enabling Inference" [<http://www.mozilla.org/rdf/doc/inference.html>] and "RDF query and inference" [<http://rdfinference.org/>]; many related topics are discussed in 'RDF-Logic' [<http://lists.w3.org/Archives/Public/www-rdf-logic/>].

XML Schema with its differentiated type system in Part 2 will complement RDF typing [<http://www.w3.org/TR/xmlschema-2/>]. For this and other reasons XML Schema should be analyzed from a logic perspective as well. Future work should also explore which version(s) of negation would make most sense in the open world of the Web, where (negative) conclusions should be drawn skeptically: one candidate is van Gelder's well-founded semantics (Przymusiński: 3-valued stable semantics) [<http://lists.w3.org/Archives/Public/www-rdf-interest/1999Dec/0134.html>].

The current RDF/XML serialization syntaxes would benefit from the following revision of the XML 1.0 standard: Instead of writing each closing XML bracket redundantly as a full end-tag that even repeats the namespace prefix, one could permit – approaching Prolog structures – the use of a 'neutral' closing bracket `</>`, as in XML-QL [<http://www.w3.org/TR/NOTE-xml-ql/>]. Recent proposals by Berners-Lee and Melnik attempt to unify the RDF and XML syn-

taxes [<http://www-db.stanford.edu/~melnik/rdf/fusion.html>]; meanwhile Berners-Lee is developing Notation 3 (N3) [<http://www.w3.org/2000/10/swap/Primer.html>]. For a kind of ‘syntax-independent’ RDF editing a variant of Protégé-2000 has been developed [NSD<sup>+</sup>01] [<http://smi-web.stanford.edu/projects/protege/protege-rdf/protege-rdf.html>].

A wide use of information agents will call for a standardized, XML-based markup language for semantic resource description on the Web. RDF already provides a simple kernel of such a representation language, but will need extensions to permit less complex representation constructions for descriptions of the kinds discussed in this paper. The question is whether these extensions can be incorporated into the current RDF standard or will need some principal revision. The issues parallel the old debate of whether and how to extend simple semantic nets towards predicate logic [Woo75], except that now the nodes (and arc labels) can be URLs and the serialization syntax is based on XML. Finally, the relationships between such an extended RDF and emerging description-logic (DLML [<http://co4.inrialpes.fr/xml/dlml/>]), ontology (XOL [KCT99], OIL [Hea00]), rule [<http://www.dfki.de/ruleml>], and agent (DAML [Hen00] [<http://www.daml.org/>]) languages need further study.

## A Diagram Form of Directed Hyperarcs Exemplified

Directed-hyperarc arrows for N-ary relationships cut N-2 intermediate node occurrences. Thus, the arrow for the `ships` relationship from section 6 (N=3) cuts the node for XML4You’s book URL, "<http://www.onoffbook.com/bookId/00498>":



## References

- [ABvE<sup>+</sup>01] A. Abecker, A. Bernardi, L. van Elst, A. Lauer, H. Maus, S. Schwarz, and M. Sintek. FRODO: A Framework for Distributed Organizational Memories. Milestone M1: Requirements Analysis and System Architecture. Technical Report D-01-01, DFKI, March 2001.
- [BG00] Dan Brickley and R.V. Guha. Resource Description Framework (RDF) Schema Specification 1.0. Candidate Recommendation CR-rdf-schema-20000327, W3C, March 2000.
- [BLF99] Tim Berners-Lee and Mark Fischetti. *Weaving the Web: The Original Design and Ultimate Destiny of the World Wide Web by its Inventor*. Harper, San Francisco, 1999.

- [Bol92] Harold Boley. Declarative Operations on Nets. In Fritz Lehmann, editor, *Semantic Networks in Artificial Intelligence*, volume 23, pages 601–637. Special Issue of Computers & Mathematics with Applications, Pergamon Press, 1992.
- [Bol99] Harold Boley. ONTOFILE: Exterior and Interior Ontologies of File/HTTP URLs. In Hannu Jaakkola, Hannu Kangassalo, and Eiji Kawaguchi, editors, *Information Modelling and Knowledge Bases X*. IOS Press, Amsterdam, “Frontiers in Artificial Intelligence and Applications”, Spring 1999.
- [Bol01] Harold Boley. Cross-Fertilizing Logic Programming and XML for Knowledge Representation. In Rolf Grütter, editor, *Knowledge Media in Healthcare: Opportunities and Challenges*. Hersey - London - Melbourne - Singapore: Idea Group Publishing, 2001.
- [Cha00] Pierre-Antoine Champin. RDF Tutorial. Technical Report, LISI (Laboratory of Information Systems Engineering), Université Claude Bernard, Lyon 1, March 2000.
- [DBSA97] Stefan Decker, Dan Brickley, Janne Saarela, and Jürgen Angele. A Query and Inference Service for RDF. In *QL'98 - The Query Languages Workshop*, <http://www.w3.org/TandS/QL/QL98/>. World Wide Web Consortium, 1997.
- [Hea00] Ian Horrocks and Dieter Fensel et al. The Ontology Inference Layer OIL. Technical Report, <http://www.ontoknowledge.org/oil/TR/oil.long.html>, June 2000.
- [Hen00] James A. Hendler. Activities at DARPA. <http://www.cs.umd.edu/~hendler/darpa.html>, 2000.
- [KCT99] Peter D. Karp, Vinay K. Chaudhri, and Jerome Thomere. XOL: An XML-Based Ontology Exchange Language. Technical Report, Artificial Intelligence Center, SRI International, August 1999.
- [Llo87] John W. Lloyd. *Foundations of Logic Programming*. Springer-Verlag, Berlin, Heidelberg, New York, 1987.
- [LS99] Ora Lassila and Ralph R. Swick. Resource Description Framework (RDF) Model and Syntax Specification. Recommendation REC-rdf-syntax-19990222, W3C, February 1999.
- [NSD<sup>+</sup>01] N.F. Noy, M. Sintek, S. Decker, M. Crubézy, R. W. Fergerson, and M. A. Musen. Creating Semantic Web Contents with Protégé-2000. *IEEE Intelligent Systems*, 16(2):60–71, 2001.
- [Woo75] William A. Woods. What's in a Link: Foundations for Semantic Networks. In Daniel C. Bobrow and A. M. Collins, editors, *Representation and Understanding: Studies in Cognitive Science*, pages 35–82. Academic Press, New York, 1975.