RDF COLLECTIONS

Saket Kaushik and Duminda Wijesekera
Department of Information and Software Engineering, George Mason University, Fairfax, Virginia 22030, U.S.A.

Keywords: RDF, Ontology, Higher Order Types, Entailment, abstract data types.

Abstract: We add collection types such as sets, bags and lists to the resource description framework (RDF).

1 INTRODUCTION

The Semantic Web consists of a layered architecture (Burners-Lee et al., 2001) of web languages, and is hinged on the Resource Description Framework (RDF) (Brickley and Guha, 2003) for expressing internet ontologies. Ontology languages above the layer of RDF, such as OWL (McGuinness and van Harmelen, 2004), DAML *etc.*, add custom syntax to RDF for expressing ontologies. A different semantics for each layer/language places severe constraints on interoperability of Semantic Web agents and applications. Consequently, a need exists for specifying semantics of all Semantic Web languages in terms of a single language and its semantics – this language is called L_{base} (Guha and Hayes, 2003). It is expressive enough to encompass all current languages and ensures interoperability between them.

Though highly expressive, L_{base} lacks certain desirable features (Guha and Hayes, 2003), like temporal ontologies, version control for ontologies, adequate support for higher-order constructs, *etc.* Same is true for its successor specification – RDF Semantics (Hayes and McBride, 2004) and predecessors((Fikes and McGuinness, 2001), *etc.*). Although the cited specifications provide a syntax for higher types like bags, lists, alternatives, *etc.*, they don't interpret these types. As a result, Semantic Web agents are unable to deduce useful properties relating to collections. For example, consider the case of a shopping bag (a multiset that may include multiple copies of members). Suppose each shopping bag item has an associated price. A useful inference could be the total cost price for the bag. However, this calculation/deduction is not currently supported in RDF – a deficiency addressed in this work.

1.1 Our Contribution

In this paper we model a fragment of RDF that does not include self references. In addition to the *simple graphs* (Hayes and McBride, 2004), we provide *reified graphs* syntax with a denotational semantics to the graph. Additionally, our graphs support three common higher-order types, *viz.*, sets, sequences and multisets (bags), that are not currently interpreted in RDF. We give a system of rewrite rules and include them in the semantics of reified graphs, closely strengthening the graph entailments of (Hayes and McBride, 2004). We show the usefulness of our addition by using an example.

The paper is organized as follows. We first introduce RDF and the resources in section 2. This is followed by formalization of collections in Section 3. Section 4 introduces reified graph definitions that include higher order types. Semantics of these graphs are presented in section 5 and graph entailments in section 6. Section 7 concludes the paper.

2 RESOURCE DESCRIPTION FRAMEWORK

RDF is used to specify meta-information about *resources*, *i.e.*, entities that can be uniquely identified, and *binary relations* (binary properties between resources) between them so that they can be "machine processed". RDF does so by using the syntax of *triples* where the *subject* (the first component of the triple) is related by the *property* (the second component of the triple) to the *object* (the third component). Commonly, subjects and objects are

Kaushik S. and Wijesekera D. (2007).
RDF COLLECTIONS.
In *Proceedings of the Third International Conference on Web Information Systems and Technologies - Web Interfaces and Applications*, pages 91-97
DOI: 10.5220/0001286500910097

atomic, *i.e.*, they have no further set-theoretic structure. However, properties can be extended beyond connecting atomic objects, *i.e.*, to expressing relationships between properties and atomic objects. This process, carried out recursively, is called *reification*. RDF(S) or RDF Schema is RDFs vocabulary description language. It has syntax to describe concepts and resources using the meta-syntax of `rdfs:Class`, `rdf:type`, *etc.*, and relationships between resources through `rdf:property`. These meta classes are used to specify properties of user defined schema. Details of RDF/RDF(S) syntax and vocabulary descriptions can be found in (Brickley and Guha, 2003).

A collection of RDF triples form a graph, *i.e.*, if the object of a triple is the subject of another triple, then the two triples are merged together retaining the common object only once (with one incoming edge and one outgoing edge). Hayes and McBride (Hayes and McBride, 2004) give semantics of RDF/RDF(S) as RDF graphs (informally) using model theory. They informally state an entailment relationship between graphs. However, this semantics is lacking in several respects. Firstly, they described containers such as *multisets* and *lists* without semantics. As a remedy, we provide rewrite rules for abstract data type (Harrison, 1993) for three collections, *viz.*, sets, bags and sequences and, correspondingly, extend RDF entailments to include them.

3 CONTAINERS FORMALIZED

Sets, bags and sequences are abstract data types that support a set of algebraic operations, described next. In our formulation, a set is a collection of homogeneous base types (atomic or otherwise) that does not allow duplicates or location ordering of its elements. A bag, or a multi-set, on the other hand is a homogeneous collection that may contain duplicates. If T is a type then we use the notation $Bag[T]$ for the bag type of objects of type T (respectively, $Set[T]$ for sets). We write $x \in B$ to denote that x is an element in bag B (the same notation is used for sets). In contrast to sets, the multiplicity of an element in a bag is the number of its repetitions in the bag (Grumbach and Milo, 1993; Albert, 1991; Dayal et al., 1982; Kent, 1989; Mumick et al., 1990). That is, modeled as a function, referred to as *profile function* in the literature, mapping each member of a bag B to its repetition count (a natural number); represented in infix notation using the function symbol $\in$. Thus, for $x \in B$, $(x \in B) \in \mathbb{N}^+$ (the set of positive natural numbers). Size of a bag is given by the number of objects in the bag, counting repetitions, and provides a basis for comparison amongst bags based on their elements.

A sequence ($Seq[T]$) is a homogeneous collection with linearly ordered members – an ordering based on their, so called, location in the sequence. That is, a function (called the *ordering function*) maps each member of a sequence to the set $\mathbb{N}$, represented by μ, such that for no two distinct objects x, y of a sequence S satisfies $\mu_S(x) = \mu_S(y)$ (Our rewrite rules don't make explicit use of this function). In addition, two functions *head* and *tail* return the first element and rest of the elements of a sequence, *i.e.*, *first*(S) = x iff $\mu_S(x) = 1$ and *last*(S) = S' iff $\mu_{S'}(y) = (\mu_S(y) - 1)$, for all $y \in S, \mu_S(y) \neq 1$. Next we provide algebraic specifications for the collection types (Harrison, 1993) using the notation $x : A$ for 'x is of type A'.

Definition 1 (Basic algebra for sets of finite type) *Given base types T_i (including boolean, $\mathbb{N}$, etc.), the syntax of operations on sets over T_i (represented as $Set[T_i]$) are:*

Table 1: Set type.

Operation	Type	Name
{ x}	$T_i \rightarrow \text{Set}[T_i]$	**Constructor**
ins	$T_i \times \text{Set}[T_i] \rightarrow \text{Set}[T_i]$	**Insert**
$\cup$	$\text{Set}[T_i] \times \text{Set}[T_i] \rightarrow \text{Set}[T_i]$	**Union**
$\cap$	$\text{Set}[T_i] \times \text{Set}[T_i] \rightarrow \text{Set}[T_i]$	**Intersection**
$\setminus$	$\text{Set}[T_i] \times \text{Set}[T_i] \rightarrow \text{Set}[T_i]$	**Minus**
$\in$	$T_i \times \text{Set}[T_i] \rightarrow \text{boolean}$	**Membership**
$\subseteq$	$\text{Set}[T_i] \times \text{Set}[T_i] \rightarrow \text{boolean}$	**Subset**
==	$\text{Set}[T_i] \times \text{Set}[T_i] \rightarrow \text{boolean}$	**Equality**
rem	$T_i \times \text{Set}[T_i] \rightarrow \text{Set}[T_i]$	**Remove**
cnt	$\text{Set}[T_i] \rightarrow \mathbb{N}$	**Cardinality**

Properties of set (a) members, given a is of type $Set[T_i]$:

$$\begin{aligned} x \notin \{\}_{T_i} & \quad & false \\ x \in a : Set[T_i] & \Rightarrow & x : T_i \\ b : Set[T_i] \subset a, x \in b & \Rightarrow & x : T_i \\ ins(x,a) : Set[T_i] & \Rightarrow & x : T_i \\ rem(x,a) : Set[T_i] & \Rightarrow & x : T_i \end{aligned}$$

Axiomatic Semantics for operations on sets are as follows:

$$\begin{aligned} x \in ins(y,z) &= true \text{ if } x=y,\ x \in z \text{ otherwise} \\ cnt(\{\}_{T_i}) &= 0 \\ cnt(ins(x,z)) &= 1 + cnt(z) \text{ if } x \notin z, cnt(z) \text{ otherwise} \\ a \subset b &= true \text{ if } (x \in a \Rightarrow x \in b) \\ & \quad\ false \text{ otherwise} \\ a == b &= a \subset b \wedge b \subset a \\ rem(x,z) &= y \text{ such that } x \in z \wedge z = ins(x,y), \\ & \quad\ z \text{ otherwise} \\ a \setminus b &= y \text{ such that } x \in y \Rightarrow x \in a \wedge x \notin b \\ a \cap b &= y \text{ such that } x \in y \Rightarrow x \in a \wedge x \in b \\ a \cup b &= y \text{ such that } \ x \in y \Rightarrow x \in a \vee x \in b \end{aligned}$$

Basic bag algebra is presented next. Note that we specify 'additive union' operation for bags instead of

the usual set union. This operation adds the total number of occurrences in the resulting bag.

Definition 2 (Basic algebra for bags of finite type) *Given base types T_i (including $\{true, false\}$: boolean, natural numbers: $\mathbb{N}$); syntax of operations on bags over types T_i are:*

Table 2: Bag type.

Operation	Type	Name
$\wr x,x,\ldots,x\int$	$T_i \rightarrow \text{Bag}[T_i]$	**Constructor**
ins	$T_i \times \mathbb{N} \times \text{Bag}[T_i] \rightarrow \text{Bag}[T_i]$	**Insert**
rem	$T_i \times \mathbb{N} \times \text{Bag}[T_i] \rightarrow \text{Bag}[T_i]$	**Remove**
$\cap$	$\text{Bag}[T_i] \times \text{Bag}[T_i] \rightarrow \text{Bag}[T_i]$	**Intersection**
$\setminus$	$\text{Bag}[T_i] \times \text{Bag}[T_i] \rightarrow \text{Bag}[T_i]$	**Minus**
$\sqcup_+$	$\text{Bag}[T_i] \times \text{Bag}[T_i] \rightarrow \text{Bag}[T_i]$	**Additive Union**
$\in$	$T_i \times \text{Bag}[T_i] \rightarrow \text{boolean}$	**Membership**
$\Subset$	$T_i \times \text{Bag}[T_i] \rightarrow \mathbb{N}$	**Profile**
$\Subset$	$\text{Bag}[T_i] \times \text{Bag}[T_i] \rightarrow \text{boolean}$	**Subbag**
$\Subset$	$\text{Bag}[T_i] \times \text{Bag}[T_i] \rightarrow \text{boolean}$	**Proper Subbag**
cnt	$\text{Bag}[T_i] \rightarrow \mathbb{N}$	**Cardinality**

Given a bag a of type $Bag[T_i]$, properties of members are:

$$\begin{aligned} x \Subset \wr \int_{T_i} &= 0 \\ x \in \wr \int_{T_i} &= false \\ x \in a : Bag[T_i] &\Rightarrow x : T_i \\ (x \Subset a : Bag[T_i]) > 0 &\Rightarrow x : T_i \\ b \Subset a : Bag[T_i], x \in b &\Rightarrow x : T_i \\ ins(x,a) : Bag[T_i] &\Rightarrow x : T_i \\ rem(x,a) : Bag[T_i] &\Rightarrow x : T_i \end{aligned}$$

Given $x,y \in T; n \in \mathbb{N}; b,b' \in Bag[T]$, the semantics of operations is as follows:

$$\begin{aligned} x \in ins(y,b) &= true\ if\ x{=}y,\ x \in b\ otherwise \\ x \Subset b &= 0\ if\ x \notin b \\ x \Subset ins(x,b) &= 1 + x \Subset b \\ cnt(\wr \int_{T_i}) &= 0 \\ cnt(ins(x,b)) &= 1 + cnt(b) \\ a \Subset b &= true\ if\ (x \in a \Rightarrow x \in b) \wedge \\ &\quad (x \Subset a < x \Subset b), false\ otherwise \\ a \Subseteq b &= true\ if\ (x \in a \Rightarrow x \in b) \wedge \\ &\quad (x \Subset a \leq x \Subset b), false\ otherwise \\ rem(x,b) &= b'\ such\ that\ x \in b \wedge z = ins(x,b'), \\ &\quad b\ otherwise \\ a \setminus b &= b'\ such\ that\ x \Subset b' = \\ &\quad max(0,(x \Subset a - x \Subset b)) \\ a \cap b &= b'\ such\ that\ x \Subset b' = min(x \Subset a, x \Subset b) \\ a \sqcup_+ b &= b'\ such\ that\ x \Subset' = (x \Subset a + x \Subset b) \end{aligned}$$

Next we define the algebra of sequences as recursive types that are built over the terminal element – *nil*.

Definition 3 (Algebra of sequences) *Given base types T_i and a constant symbol nil, syntax of operations on sequence type (written $Seq[T_i]$) are as follows: Given a sequence a of type $Seq[T_i]$, properties of members are:*

Table 3: Sequence type.

Operation	Type	Name
$\langle x \rangle$	$T_i \rightarrow Seq[T_i]$	**Constructor**
hd	$Seq[T_i] \rightarrow T_i$	**Head**
tl	$Seq[T_i] \rightarrow T_i$	**Tail**
$\in$	$T_i \times Seq[T_i] \rightarrow boolean$	**Membership**
$==$	$Seq[T_i] \times Seq[T_i] \rightarrow boolean$	**Equality**

$$\begin{aligned} x \in a : Seq[T_i] &\Rightarrow x : T_i \\ x = hd(a : Seq[T_i]) &\Rightarrow x : T_i \\ x = tl(a : Seq[T_i]) &\Rightarrow x : Seq[T_i] \end{aligned}$$

Semantics of operations are provided next.

$$\begin{aligned} hd(a) &= h : T_i\ such\ that\ \exists t : Seq[T_i], a = \langle h.t \rangle \\ &\quad nil\ otherwise \\ tl(a) &= t : Seq[T_i]\ such\ that\ \exists h : T_i, a = \langle h.t \rangle \\ &\quad \langle nil \rangle\ otherwise \\ a == b &= \exists h_1,h_2 : T_i, t_1,t_2 : Sequence, a = \langle h_1.t_1 \rangle \wedge \\ &\quad b = \langle h_2.t_2 \rangle \wedge h_1 = h_2 \wedge t_1 = t_2 \\ x \in \langle nil \rangle &= false\ if\ x \neq nil,\ true\ otherwise \\ x \in a &= true\ if\ \exists h : T_i, t : Sequence, x = h \vee x \in t \\ &\quad false\ otherwise \end{aligned}$$

Definition 4 (Algebra of Triples) *Given types S, O and P (type T denotes type S, type O or type P), syntax of operations on triples is given in table 4. Given a triple a of type $S \times P \times O$, properties of members are:*

Table 4: Triple type.

Operation	Type	Name
(S,P,O)	$S \times P \times O \rightarrow (S,P,O)$	**Constructor**
sub	$(S,P,O) \rightarrow S$	**Subject**
prp	$(S,P,O) \rightarrow P$	**Property**
obj	$(S,P,O) \rightarrow O$	**Object**
$\in$	$T \times (S,P,O) \rightarrow boolean$	**Membership**
$==$	$(S',P',O') \times (S,P,O) \rightarrow boolean$	**Equality**

$$\begin{aligned} x = sub(a : S \times P \times O) &\Rightarrow x : S \\ x = prp(a : S \times P \times O) &\Rightarrow x : P \\ x = obj(a : S \times P \times O) &\Rightarrow x : O \end{aligned}$$

The semantics of operations are given as follows:

$$\begin{aligned} sub(x,y,z) &= x \\ prp(x,y,z) &= y \\ obj(x,y,z) &= z \\ u \in (x,y,z) &= true\ if\ u = x \vee u = y \vee u = z, false\ otherwise \\ a == b &= true\ if\ sub(a) = sub(b) \wedge prp(a) = prp(b) \\ &\quad \wedge obj(a) = obj(b), false\ otherwise \end{aligned}$$

4 FORMALIZING RDF

We formalize RDF as a collection of higher typed objects and binary relations among them, constructed from a set of atomic types over a universe of objects, U, using a set of (polymorphic) type construction operators as suggested in RDF (Hayes and McBride, 2004). Three different types of collections can be formed from atomic elements, *viz.*, *sets*, *bags* and *sequences*, defined earlier.

RDF consists of schema and instances in a layered manner. Elements of each layer include objects constructed from the set of atomic objects U, using the set constructor, $\{\ \}$, the bag constructor, $\wr\ \int$ and the sequence constructor, $\langle\ \rangle$. These functions form the set of constructors, called Σ. Depending upon the requirements, ontology modelers can use any subset of the set of type constructors. This set of atomic objects is called the *set of primary nodes*. Primary edges of the graph are formed from the pairs of primary nodes. Because of the possibility of reified graphs, the *set of triples* in a graph may consist of pairs of nodes and triples as well, defined next.

Definition 5 (Algebra for Nodes and Triples) *Given a set of atomic types U, a set of boolean values $\mathbb{B}$, integers $\mathbb{N}$ and a set of collection constructors – $\{\{\ \}, \wr\ \int, \langle\ \rangle\}$ over U, collectively called collections and represented by the type C, we inductively define nodes (N_i) and triples (Tr_i) in table 5. Properties of nodes and edges*

Table 5: Nodes and Triples.

Type	Name
$x : U \cup x : C[U] \rightarrow \mathrm{N}_0$	**Node 0 Constructor**
$y : (N_0 \times U \times N_0) \rightarrow \mathrm{Tr}_0$	**Triple 0 Constructor**
$x : (C[N_{i-1}] \sqcup_{-} C[Tr_{i-1})] \rightarrow \mathrm{N}_i$	**Node i Constructor**
$y : (N_i \times U \times N_i) \rightarrow \mathrm{Tr}_i$	**Triple i Constructor**

are presented next.

$$x \in Tr_i \quad \Rightarrow \quad sub(x) \in N_i \wedge obj(x) \in N_i$$
$$x \in Tr_i \quad \Rightarrow \quad prp(x) \neq sub(x) \wedge prp(x) \neq obj(x)$$

{**Question:** *Self references are allowed. Should this be changed?*}

Definition 6 (Algebra for RDF) *Given a universe U,and nodes and triples defined over this universe, we define a graph as a set over U with following operations: Properties of graphs are presented next.*

$$x : \mathcal{GRAPH} \quad \Rightarrow \quad y \in sub_0(x) \rightarrow y \in N_0$$
$$x : \mathcal{GRAPH} \quad \Rightarrow \quad y \in sub_i(x) \rightarrow y \in sub_{i-1}(x) \vee prp_{i-1}(x)$$
$$x : \mathcal{GRAPH} \quad \Rightarrow \quad y \in obj_0(x) \rightarrow y \in N_0$$
$$x : \mathcal{GRAPH} \quad \Rightarrow \quad y \in obj_i(x) \rightarrow y \in obj_{i-1}(x) \vee prp_{i-1}(x)$$
$$x : \mathcal{GRAPH} \quad \Rightarrow \quad y \in trp_i(x) \rightarrow y \in Tr_i$$
$$x : \mathcal{GRAPH} \quad \Rightarrow \quad \exists k : \mathbb{N} | Tr_k(x) = \{\}$$
$$z = x \cup y \quad \Rightarrow \quad trp_i(z) : Tr_i, trp_i(z) \subseteq trp_i(x) \cup trp_i(y)$$

Table 6: Graph type.

Opn	Type	Name
$\{x\}$	$Tr \rightarrow \mathcal{GRAPH}$	**Constructor**
sub_i	$\mathcal{GRAPH} \rightarrow Set$	**N_i nodes**
obj_i	$\mathcal{GRAPH} \rightarrow Set$	**N_i nodes**
trp_i	$\mathcal{GRAPH} \rightarrow Set$	**Tr_i triples**
$\cup$	$\mathcal{GRAPH} \times \mathcal{GRAPH} \rightarrow \mathcal{GRAPH}$	**Merge**
$\setminus$	$\mathcal{GRAPH} \times \mathcal{GRAPH} \rightarrow \mathcal{GRAPH}$	**Minus**
$\rhd$	$\mathcal{GRAPH} \times \mathcal{GRAPH} \rightarrow boolean$	**Subgraph**
$=_s$	$\mathcal{GRAPH} \times \mathcal{GRAPH} \rightarrow boolean$	**Equality**

The semantics of operations are as follows:

$$\begin{aligned}
sub_i(a \cup b) &= sub_0(a) \cup sub_0(b) \\
obj_i(a \cup b) &= obj_0(a) \cup obj_0(b) \\
trp_i(a \cup b) &= prp_0(a) \cup trp_0(b) \\
sub_i(a \setminus b) &= sub_i(a) \setminus sub_i(b) \\
obj_i(a \setminus b) &= obj_i(a) \setminus obj_i(b) \\
trp_i(a \setminus b) &= trp_i(a) \setminus trp_i(b) \\
a \rhd b &= true\ if\ trp_i(a) \subseteq trp_i(b) \\
a =_s b &= a \rhd b \wedge b \rhd a
\end{aligned}$$

Equality relation '$=_s$', defined above, equates 'structurally equivalent' graphs. That is, two graphs are structurally equal if they have exactly the same structure (based on the equality relation of constituent types). RDF also allows name-based equality, which we cover in the next section.

4.1 Named RDF Graphs

To enable name-based equality of RDF graphs, we introduce names for graph elements.

Definition 7 (Names) *Let U be a universe of objects.*

Named Graph *We use a set of names, called $\mathcal{NAMES}$ and a naming function $\mathcal{N}$ from Graph to $\mathcal{NAMES} \cup (\perp \times \mathbb{N})$ (where $(\perp \times \mathbb{N})$ represents a blank node and its 'node Id' (Hayes and McBride, 2004)) that maps each node and property in an RDF graph to its name. We abuse the notation slightly to say $\mathcal{N}(x) = \perp$ if a node is a blank node.*

Ground Graph *A graph that has no nodes mapped to $\perp$.*

Next, we define name-based equality of graphs. With structural equality as a prerequisite, blank nodes at similar structures are assumed to be equal). This is followed by the definition of graph instances.

Definition 8 (Name-based Equality '$=_n$') *Two graphs f and g, with name maps $\mathcal{N}$ and $\mathcal{N}'$, are said to be name-wise equal, i.e., $f =_n g$, if $f =_s g$ and $\forall i (t \in trp_i(f) \Rightarrow \exists t' \in trp_i(g) | \mathcal{N}(prp(t)) = \mathcal{N}'(prp(t')) \wedge \mathcal{N}(sub(t)) = \mathcal{N}'(sub(t')) \wedge \mathcal{N}(obj(t)) = \mathcal{N}'(obj(t')))$.*

Definition 9 (Instance) *A graph a with naming function $\mathcal{N}$ is said to be an instance of graph b (naming function $\mathcal{N}'$) if*

- $\forall i(x \in (sub_i(b) \cup obj_i(b) \cup prp(trp_i(b))) \wedge \mathcal{N}'(x) \neq \perp \Rightarrow \exists y \in (sub_i(a) \cup obj_i(a) \cup prp(trp_i(a)))|\mathcal{N}'(x) = \mathcal{N}(y))$.
- $\forall i(x \in (sub_i(b) \cup obj_i(b) \cup prp(trp_i(b))) \wedge \mathcal{N}'(x) = \perp \Rightarrow \exists y \in (sub_i(a) \cup obj_i(a) \cup prp(trp_i(a)))|\mathcal{N}(y)) \in \mathcal{NAMES} \cup \perp$.

We are now in a position to express basic collection types in RDF. We use available syntax for expressing bag and sequence types, *i.e.*, `rdf:Bag` and `rdf:Seq`, and introduce `rdf:Set` to express Set types. Next we formalize the shopping bag example discussed earlier.

Example 1 *Consider a group of atomic, named RDF nodes (subjects). Each of the resource has a property named 'value', i.e., domain(value)= nodes and range(value)= $\mathbb{N}$. (RDF syntax for describing these triples is straightforward and is omitted here). We assume $\mathbb{N}$ admits simple arithmetic (addition function and equality predicate) (Presburger, 1929). Consider now a* `rdf:Bag` *type resource with each of above described resources as its members. We call this bag a 'shopping bag'. Since each member has some numeric value, we can describe a higher-order property, named 'total', based on the 'value' property of the members using simple addition over values. That is, we can now express a triple with property name 'total' such that domain(total) = Bag[ShoppingItems] and range(total)= $\mathbb{N}$.*

5 SEMANTICS

In this section we provide limited semantics for finite types introduced earlier. To do so we start with a universe, *i.e.*, a set of URI's (denoted *URI*) and other constants, say *B* as *urelements* (*i.e.*, those atomic elements that do not have any further structure to them (Aczel, 1988; Kunen, 1983)). There is no URI corresponding to blank nodes. In the following $\mathcal{P}$ represents powerset operator. Hierarchical universe is built as follows:

N_0 and P_0:

$$\begin{aligned} N_0 &= \{(u,b) : u \in URI \text{ and } b \in B\} \\ P_0 &= \{(u,(b,b')) : u \in URI \text{ and } b,b' \subseteq B\} \end{aligned}$$

N_{n+1} and P_{n+1}: Suppose $\mathbf{N_n}$ and $\mathbf{P_n}$ have been defined. Let $U_n = \{x : \exists u \in URI(u,x) \in N_n\}$. Then define

$$\begin{aligned} N_{n+1} &= \{(u,B) : u \in URI \text{ and} \\ &\quad B \subseteq \mathcal{P}(U_n) \cup \mathcal{B}(U_n) \cup \mathcal{S}(U_n)\} \\ P_{n+1} &= \{(u,(B,B')) : u \in URI \text{ and} \\ &\quad B,B' \subseteq \mathcal{P}(U_n)\} \end{aligned}$$

We use this stratified named universe to interpret RDF graphs and related syntax as follows:

Definition 10 (Semantic mapping $[\![\,]\!]$)

Universe:

1. *Suppose U is the universe as defined in definition 7.*
2. *Let $[\![\,]\!]_u : U \mapsto B$ be the surjective mapping that maps the constants in the syntax to a set B of urelements over which the syntax is interpreted.*
3. *Let $[\![\,]\!]_{name} : \mathcal{NAMES} \mapsto URI$ be a mapping of RDF names to URIs.*

Mapping nodes: *Every node $n \in N_n$ named a is mapped as $[\![(a,n)]\!] = ([\![a]\!]_{name}, [\![n]\!])$ where $[\![n]\!]$ is constructed by replacing every $u \in U$ in n with $[\![u]\!]$.*

Mapping properties: *For every property $p \in P_n$ named a is mapped as $[\![(a,p)]\!] = ([\![a]\!]_{name}, [\![p]\!])$ where $[\![p]\!]$ is constructed by substituting every $u \in U$ in p with $[\![u]\!]$.*

Interpreting triples: *For every triple $(s,p,o) \in Tr_n$ is interpreted as $[\![(s,p,o)]\!] = [\![s]\!] \wedge [\![p]\!] \wedge [\![o]\!]$.*

Interpreting `rdf:type`: *We say that $[\![x$, rdf:type, $y]\!]$ iff $x = (a,b)$, $y = (p,q)$ and $[\![b]\!] \in [\![q]\!]$.*

Interpreting `rdfs:subClassOf`: *$[\![x$, rdfs:subClassOf, $y]\!]$ iff $x = (a,b)$, $y = (p,q)$ and $[\![b]\!] \subseteq [\![q]\!]$.*

Interpreting names: *The name of an object or a property x is defined as the first coordinate of $[\![x]\!]$.*

Interpreting intensional equality: *x is said to be intensionally equal to y (written $x =_{int} y$) iff $[\![x]\!] = [\![y]\!]$*

Interpreting structural equality of nodes and properties: *x is said to be structurally equal to y, written $x =_{str} y$ iff $(a,b) = [\![x]\!]$, $(p,q) = [\![y]\!]$ and $b = q$.*

Interpreting structural equality of triples: *(s,p,o) is said to be structurally equal to (s',p',o'), written $(s,p,o) =_{str} (s',p',o')$ iff $[\![(s,p,o)]\!] \wedge [\![(s',p',o')]\!] \wedge s =_{str} s' \wedge p =_{str} p' \wedge o =_{str} o'$*

Interpreting intensional equality of triples: *(s,p,o) is said to be intensionally equal to (s',p',o'), written $(s,p,o) =_{int} (s',p',o')$ iff $[\![(s,p,o)]\!] = [\![(s',p',o')]\!]$*

Interpreting ground graph: *Ground graph $[\![\{(s,p,o)\}]\!]$ interpretation is constructed by interpreting each triple in the graph as $\{[\![(s,p,o)]\!]\}$*

Interpreting structural equality of graphs: *A graph g is structurally equal to a graph g' iff for each triple $t \in g$ there exists exactly one triple $t' \in g'$ such that $t =_{str} t'$.*

Interpreting graph: *Graph $[\![g]\!]$ is interpreted if*

- *$\exists g' : \mathcal{GRAPH}$ such that g' is ground and $[\![g']\!]$ is interpreted and*
- *$g =_{str} g'$ and*
- *For all ground triples $t \in g$ there exists a triple $t' \in g'$ such that $t =_{int} t'$ and for all triples $t \in g$ if $sub(t)$ or $obj(t)$ is ground, then there exists a triple $t' \in g'$ such that $t =_{str} t'$ and $sub(t) =_{int} sub(t')$ or $obj(t) =_{int} obj(t')$.*

6 GRAPH ENTAILMENTS

In this section we make precise Hayes and McBride's (Hayes and McBride, 2004) informal treatment of graph entailments that include higher order constructs.

Definition 11 (Entailment) *We say that a graph g entails another graph g' if $[\![g']\!] \subseteq [\![g]\!]$. We write this as $g \vdash g'$. Otherwise we say g does not entail g', i.e., $g \nvdash g'$*

Graph entailments simply state the *satisfiability* of a graph given a related graph is *satisfied*. Since we consider finite graphs, compactness is easily shown.

Lemma 6.1 (Graph Entailments) *Hayes and McBride's (Hayes and McBride, 2004) results hold in our framework:*

Empty graph entailment *Given g:$\mathcal{GRAPH}$, g entails empty graph, i.e., $g \vdash \{\}$*

Empty graph *Given g:$\mathcal{GRAPH}$ and g is not an empty graph, g is not entailed by empty graph, i.e., $\{\} \nvdash g$*

Subgraph entailment *Given g,f:$\mathcal{GRAPH}$ and $g \rhd f$, then $g \vdash f$.*

Instance entailment *Given g,f:$\mathcal{GRAPH}$ and g is an instance of f, then $g \vdash f$.*

Merging (L) *Given g,f:$\mathcal{GRAPH}$ then $g \wedge f \vdash g \cup f$.*

Merging (R) *Given g,f:$\mathcal{GRAPH}$ then $g \cup f \vdash g \wedge f$.*

Interpolation entailment *Given f,h:$\mathcal{GRAPH}$, $g_i : \mathcal{GRAPH}, (i \leq n)$ $(\bigcup_i g_i) \rhd f$ and f is an instance of h then $(\bigcup_i g_i) \vdash h$.*

Monotonicity *Given g,f,h:$\mathcal{GRAPH}$, $g \rhd f$ and $f \vdash h$ then $g \vdash h$*

Proof Sketch:

Empty graph entailment Let g:$\mathcal{GRAPH}$ and g = $\{\}$. Now, $\emptyset \in [\![g]\!]$.

Empty graph Let g:$\mathcal{GRAPH}$, g $\neq \{\}$. By definition, $[\![g]\!] \neq \emptyset$.

Subgraph entailment Let g,f:$\mathcal{GRAPH}$ and $g \rhd f \Rightarrow Tr_i(g) \subseteq Tr_i(f)$. Let $\mathcal{N}$ be the naming function for g. Given $[\![g]\!]$ is interpreted, $[\![f]\!]$ can be interpreted with $\mathcal{N}$ as the naming function.

Instance entailment Let g,f:$\mathcal{GRAPH}$, g instance of f and $\mathcal{N}, \mathcal{N}'$ the respective naming functions. $[\![g]\!]$ is interpreted; by definition $\forall x \in sub(trp_i(f))$ such that $\mathcal{N}'(x) \neq \perp (\exists y \in sub(trp_i(g))$ such that $y =_{str} x \wedge \mathcal{N}'(x) = \mathcal{N}(y))$. Now, $\forall x \in sub(trp_i(f))$ such that $\mathcal{N}'(x) = \perp$, define $[\![x]\!]$ = $[\![y$ such that $y \in sub(trp_i(g)), y =_{str} x]\!]$. Similar construction is possible for objects and properties.

Merging L By definition, if $g_1, g_2 : \mathcal{GRAPH}$, then $\forall i (trp_i(g_1 \cup g_2) = trp_i(g_1) \cup trp_i(g_2))$. Clearly, with $N_\cup = N_{g_1} \cup N_{g_2}$ it is easy to show $g_1 \wedge g_2 \vdash g_1 \cup g_2$. This result is easily extended to a set of graphs.

Merging R Shown in the similar way as in previous case.

Interpolation entailment Shown similarly using subgraph, instance and merging lemmas.

Monotonicity Shown similarly using subgraph lemma.

Next we state an entailment relation for collections. That is, given a property for a collection of objects, and allowing a simple arithmetic (Presburger arithmetic (Presburger, 1929)) we can infer a higher order property for the collection.

Lemma 6.2 (Collection Entailment) *g:$\mathcal{GRAPH}$, x:C[T] for some T, for some i $\forall x : C[T] \in sub_i(g)(\exists t \in Tr_i | sub(t) = x \wedge prp(t) = p)$, where p is some mathematical function*

$$g, \forall x(x \in sub_i(g) : C[T](\exists t \in Tr_i(g), prp(t) = p, sub(t) = x))$$
$$\forall x(x : C[T], p(x) = n \rightarrow p'(C[T]) = N)$$
$$\vdash$$
$$g \cup \{(C[T], p', N)\}, \text{ where } p' \text{ is a function and}$$
$$(C[T], p', N) \in Tr_{i+i}(g \cup \{(C[T], p', N)\})$$

Proof Sketch: Follows from subgraph lemma and basic arithmetic.

Collection entailment states that if there is a graph, say *g*, containing a node, say n, that is a collection (set, bag or a sequence) and each of its members have an arithmetic property, then *g* entails a graph that expresses a collective property for *n* in addition to all other properties expressed by *g*. This property is derived by basic arithmetic over the lower order arithmetic properties. We revisit our running example to show collection entailment in action. We omit RDF syntax in our example, however, it is easily checked that a simple translation to RDF syntax exists.

Example 2 *Consider an RDF graph g with some k level nodes, $x_i, i \leq n : N$ such that for each x_i there is a node y_i that is a number (literal), related to x_i through a property ψ_k. Also, there is a $k+1$ level node X of type* `rdf:Bag`*, with each x_i as its member. This graph models the shopping bag in example 1, where each member of a bag has some number value, expressed as a 'value' triple. Since this property is functional, we can use a function to represent such edges. That is, we express ψ_k as a function $h(x) := x \in X \rightarrow \mathbb{N}$ for all members x_i of the bag X. Clearly, it's easy to compute the total value of all the items stored in X by adding them all together:*

$$h'(X) = \forall x \in X(\sum_x h(x))$$

Thus, h' is a function whose domain is a shopping bag with number valued items as members and range is $\mathbb{N}$. That is, we can now express a $k+1$ level property for a collection type, expressible in an RDF graph using some edge, say ψ_{k+1}. More concretely, suppose X is $\wr apple, orange, orange\int$ and graph g has edges value(apple, 5) *and* value(orange, 10). *Then, using basic arithmetic, we can deduce an edge* total(X, 5+10+10) *or* total(X, 25).

Using the collection entailment lemma, we are able to deduce a new graph from g, with an additional edge of name 'total' connecting X with a node named 25.

7 CONCLUSION

RDF syntax allows for ontologists to express higher-order types like sequences, bags and alternatives. These collections are essential for representing many real world facts like work flows, shopping bags, payment receipts, *etc*. Although collections can be expressed, but they are not interpreted in RDF. Consequently, Semantic Web agents cannot deduce many useful inferences based on these collections.

Authoritative specifications of RDF syntax and semantics (Guha and Hayes, 2003; Hayes and McBride, 2004) acknowledge the need for building these inferences into RDF, however, to the best of our knowledge, this deficiency has not yet been addressed in the literature. In this paper, we provide an approach to construct three higher-order types, *viz.*, sets, bags and sequences. We integrate these types into a reified RDF framework with a hierarchical semantics. To incorporate collections into graph entailments, we formalize graph entailments (Hayes and McBride, 2004) with collections in reified graphs and prove collection entailments for incorporating deductions based higher-order types.

REFERENCES

Aczel, P. (1988). *Non-well-founded sets*, volume 14 of *Lecture Notes*. CSLI.

Albert, J. (1991). Algebraic properties of bag data types. In *17th International conference on Very Large Databases*, pages 211–219.

Brickley, D. and Guha, R. (2003). Resource Description Framework (RDF) Schema Specification 1.0: RDF schema. W3C workding Draft.

Burners-Lee, T., Hendler, J., and Lassila, O. (2001). The semantic web. *Scientific American*.

Dayal, U., Goodman, N., and Katz, R. H. (1982). An extended relational algebra with control over duplicate elimination. In *1st ACM SIGACT-SIGMOD symposium on Principles of database systems*, pages 117–123.

Fikes, R. and McGuinness, D. L. (2001). An axiomatic semantics for rdf, rdf-s, and daml+oil. Technical Report Note 18, World Wide Web Committee (W3C).

Grumbach, S. and Milo, T. (1993). Towards tractable algebras for bags. In *Twelfth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 49–58.

Guha, R. and Hayes, P. (2003). LBase: semantics for languages of the semantic web. W3C workding Draft.

Harrison, R. (1993). *Abstract Data Types in Standard ML*. John Wiley and Sons.

Hayes, P. and McBride, B. (2004). RDF semantics. W3C Recommendation.

Kent, W. (1989). Profile functions and bag theory. Technical Report HPL-SAL-89-19, Hewlett-Packard Laboratories, Palo Alto, CA.

Kunen, K. J. (1983). *Set Theory*. North Holland, Reprint edition.

McGuinness, D. L. and van Harmelen, F. (2004). OWL web ontology language. http://www.w3.org/TR/owl-features/.

Mumick, I., FinkelStein, S., Pirahesh, H., and Ramakrishnan, R. (1990). Magic is relevant. In *ACM SIGMOD international conference on Management of data*, pages 247–258.

Presburger, M. (1929). Ueber die vollstaendigkeit eines gewissen systems der arithmetik ganzer zahlen, in welchem die addition als einzige operation hervortritt. In *Comptes Rendus du I congrs de Mathmaticiens des Pays Slaves*, pages 92–101, Warsaw, Poland.