Using RDF Datatypes
(Deduction beyond syllogism)

Graham Klyne

http://www.ninebynine.org/
RDF Datatypes (recap)

```
"1"
"2"
. .
"99"
.
```

```
<xsd:integer>
<x"1",1>
x<"2",2>
x<"99",99>
.
```

```
1
2
. 
99
.
```

- Lexical space
- Lexical-to-value Mapping
- Value space

Datatype URI

Denotes

ICEXT (class extension)

xsd:integer

(Datatype)
Sketch of Datatype Extension

- A number of named \( n \)-ary relations
  - e.g. xsd_integer:add:
    \[
    \{<a,b,c> | a=b+c}\]
  - e.g. xsd_integer:divrem:
    \[
    \{<a,b,c,d> |
    a=c \text{ div } d, \\
    b=c \text{ rem } d\}
    \]
- Functions to compute relations
  - e.g. for xsd_integer:add:
    \[<4,?,2> \rightarrow <4,2,2>\]
Using RDF Datatypes

• Two approaches, implemented in Swish
  – Swish is a framework, implemented in Haskell, for experimenting with RDF inference, and other stuff

• Others are possible
  – e.g. CWM / Euler
  – Survey:
Motivating Example

• Class of passenger vehicles
• Properties for vehicle passenger capacity:
  – Seated
  – Standing
  – Total
  – where: Total = Seated + Standing
Some Criteria to Consider

• Ease of definition?
• Use RDF syntax?
• Direction of inference
  – “Forward” vs “backward” deduction
  – Variation of available information
    • \(?=2+3 \rightarrow <5,2,3>\)
    • \(5=?+3 \rightarrow <5,2,3>\)
    • \(5=2+? \rightarrow <5,2,3>\)

• Separate domain knowledge from generic
• Formal model for RDF datatypes?
1st Approach:
Rule with Variable Binding Modifier

```
ex:Rule01Ant :-
  { ?pv a :PassengerVehicle ;
    :seatedCapacity ?c1 ;
    :standingCapacity ?c2 . }

ex:Rule01Con :-
  { ?pv :totalCapacity ?ct . }

@rule ex:Rule1 :-
  ( ex:Rule01Ant ) => ex:Rule01Con
  | ( xsd_integer:sum ?ct ?c1 ?c2 )
```
2nd Approach: Generalized Class Restriction

ex:VehicleRule2 :-
   { :PassengerVehicle
     a rdfs:Class ;
     rdfs:subClassOf rdfs:NamedIndividual ;
     rdfs:isDefinedBy rdfs:URIRef "ex:VehicleRule2" .
   }

   a rdfs:Class ;
   rdfs:subClassOf rdfs:NamedIndividual ;
   rdfs:isDefinedBy rdfs:URIRef "ex:VehicleRule2" .

   rdfs:comment "Generalized class restriction for vehicles." .

   rdfs:comment "The rule states that the sum of total capacity, seated capacity,
   and standing capacity of a vehicle must be less than or equal to 1."

   rdfs:comment "This is used to validate the total capacity of a vehicle.
   The total capacity is defined as the sum of seated capacity and standing capacity.
   The seated capacity and standing capacity must be non-negative integers."
Observations

• Both approaches:
  – support forward and some backward chaining
  – based on similar extension of RDF datatype

• Variable binding modifier:
  – is easier for writing more complex rules
  – can combine query and calculation in a single rule
  – requires fewer intermediate steps (transient subgraphs)

• General restriction:
  – can be implemented with standard RDF parser
  – handles backward chaining more flexibly (but slowly)
  – does not handle query/selection of RDF data
Datatype Extension Comments

- Formalism of adding named relations seems to have some practical value as a way to capture datatype idiosyncrasies.
- Completeness of datatype inference cannot, in general, be guaranteed.
- No approach for multiple-datatype inferences is worked out yet.
Demonstration...

• Preview script
  – file: VehicleCapacity.ss

• Run Swish
  – file: SwishMain.hs