RDF and RDFS Semantics

Gilles Falquet
Semantic Web Technologies
2014

RDF semantics

Objectives

- Define a notion of interpretation
  - to evaluate the truth of a triple/graph
- Define entailment
  - what can be deduced from the triples of an RDF graph

Official Reference
http://www.w3.org/TR/2014/REC-rdf11-mt-20140225//

RDF and other languages

- RDF can be used as a base notation for other languages
  - a kind of abstract syntax
- In these languages some IRI may be given particular meanings
  - => more extensive entailment

OWL Axioms:
A ⊑ (B or C)
C ⊑ (q some D)

in RDF:

```
:A rdfs:subClassOf
  [ rdf:type owl:Class ;
    owl:unionOf ( :B :C  )
  ]

:C rdfs:subClassOf
  [ rdf:type owl:Restriction ;
    owl:onProperty :q ;
    owl:someValuesFrom :D
  ]
```


Example: OWL in RDF

OWL Expressions:

A ⊑ (B or C)
C ⊑ (q some D)

in RDF:

```
:A rdfs:subClassOf
  [ rdf:type owl:Class ;
    owl:unionOf ( :B :C  )
  ]

:C rdfs:subClassOf
  [ rdf:type owl:Restriction ;
    owl:onProperty :q ;
    owl:someValuesFrom :D
  ]
```

Example: SPARQL in RDF

SPARQL Expression:

```
SELECT ?y
WHERE
{ ?class a rdfs:Class .
  { SELECT ?y WHERE {
     ?class rdfs:label ?y . }
  } .
}
```

in RDF:

```
[ a sp:Select ;
  sp:resultVariables (_:b2) ;
  sp:where ([
    sp:subject _:b1 ;
    sp:predicate rdf:type ;
    sp:object rdfs:Class ;
    ] [
    a sp:SubQuery ;
    sp:query [
        a sp:Select ;
        sp:resultVariables (_:b2) ;
        sp:where ([
           sp:subject _:b1 ;
           sp:predicate rdfs:label ;
           sp:object _:b2 ;
           ])
     ]
  ]) ]
```


Other languages extend the semantics of RDF

a:Bob a:friend a:Alice
a:friend rdfs:domain foaf:Person

RDF entailment

⇒ a:friend rdf:type rdf:Property

RDFS entailment

⇒ a:Alice rdf:type foaf:Person

- cause: for RDFS rdfs:domain has a special meaning

Defining the interpretation of a vocabulary

Simple interpretation

- Map each entity *x* of the vocabulary (URI reference, literal) to an element I(*x*) of the interpretation domain (IR)
- If an entity *p* is a property then its interpretation is is also mapped to a binary relation between real-world objects (IEXT(I(*p*)) ⊆ IR x IR)

More formally

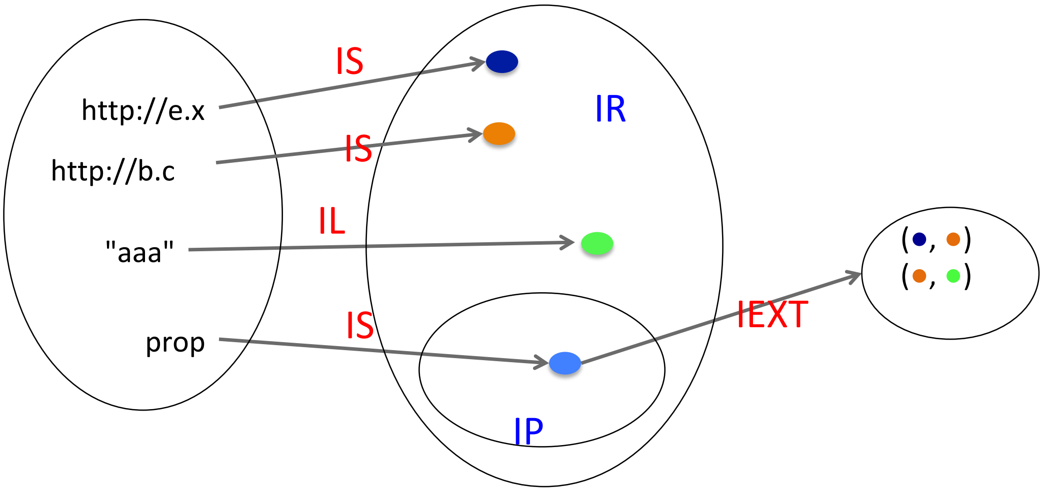
An interpretation I is a structure made of

IR : interpretation domain = (set of all resources)

IP ⊆ IR, interpretation domain for properties

Interpretation functions

- IS maps each URI reference to a resource of IR ∪ IP
- IL maps (partially) literal values to resources of IR
- IEXT maps each element of IP (property interpretation) to a relation in IR × IR (a set of resource pairs)



Interpretation of a triple

I([*s*, *p*, *o* .]) = true

if and only if

IEXT(*IS*(*p*)) contains (*IS*(*s*), *IS*(*o*))

notation

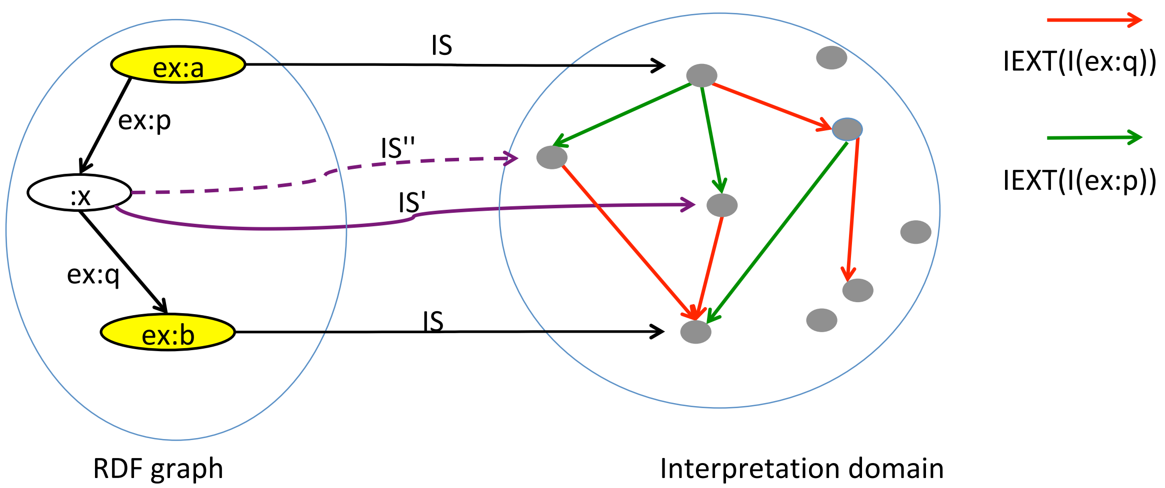
I ⊨ [*s*, *p*, *o* .]

A graph is true (for a given interpretation) iff each triple is true

The meaning of blank nodes

- A blank node represents the existence of some resource
- A graph with blank nodes is true if
  - there is an extension *IS'* of *IS* to blank nodes
  - the graph is true for *IS'*

Two possible extensions



Blank nodes and instances

- Let M be a function: B → C
  - B a set of blank nodes
  - C a set of literals, blank nodes and IRIs.

Definition. An **instance** of G is any graph obtained from G by replacing some or all of the blank nodes N in G by M(N).

Properties
- any graph is an instance of itself,
- an instance of an instance of G is an instance of G,
- if H is an instance of G then every triple in H is an instance of at least one triple in G.

Simple entailment

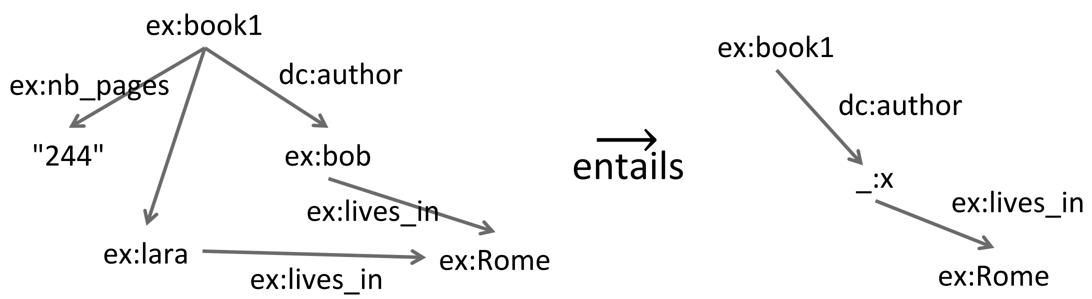
- I satisfies E when I(E) = true
- G entails E when every interpretation I which satisfies G also satisfies E

Property
- every graph is simply satisfiable

Interpolation lemma
- G simply entails E if and only if a subgraph of G is an instance of E

⇒ entailment can be checked purely syntactically

Graph entailment



Datatypes and literals

Datatypes are identified by IRIs

Each datatype *d* has a lexical-to-value (partial) mapping *L2V*(*d*) from character strings to values.

L2V : *Datatype* → (*String* → *Values*)

e.g. L2V(integer)("901") = 901; L2V(integer)("9k0s1") is undefined;

The value space of *d* is the range of *L2V*(*d*)

A literal "str" with type *d* denotes the value *L2V*(*d*)(str) (if defined) otherwise it is ill-typed

D-interpretation

If D is a set of IRIs identifying recognized datatypes

A D-interpretation is an interpretation that satisfies

(if rdf:langString ∈ D) *special treatment for language-tagged strings*

- For every language-tagged string "str"@lang,
  - IL("str"@lang) = (str, lowercase(lang))
- For every IRI *t* in D – {rdf:langString} that identifies type *d*
  - I(*t*) = *d*
  - IL("str"^^*t*) = L2V(I(*t*)(str) = L2V(*d*)(str)

- If a literal is ill-typed (value undefined), every triple containing it is *false*
- => the graph is D-unsatisfiable

D-entailment

G D-entails H if
 every D-interpretation that D-satisfies G also D-satisfies H

D-entailment cannot be lexically checked because the values of the strings must be computed.

:a :p "25.0"^^xsd:decimal

D-entails

:a :p "25"^^xsd:decimal, :a :p "25.0000"^^xsd:decimal, ...
and
:a :p "25"^^xsd:integer

RDF interpretation

An RDF interpretation recognizing D is a D–interpretation I where D includes rdf:langString and xs:string and which satisfies

the conditions

- *x* is in IP if and only if (*x*, I(rdf:Property)) ∈ IEXT(I(rdf:type))
- for every IRI *t* in D, (*x*, I(*t*)) ∈ IEXT(I(rdf:type)) if and only if *x* is in the value space of I(t)

and the triples

rdf:type rdf:type rdf:Property	rdf:subject rdf:type rdf:Property .
rdf:predicate rdf:type rdf:Property	rdf:object rdf:type rdf:Property .
rdf:first rdf:type rdf:Property	rdf:rest rdf:type rdf:Property .
rdf:value rdf:type rdf:Property .	rdf:_1 rdf:type rdf:Property .
rdf:nil rdf:type rdf:List	rdf:_2 rdf:type rdf:Property .
	...

RDFS-interpretations

Must take into account the intended semantics of the RDFS vocabulary

rdfs:domain	rdfs:range	rdfs:Resource
rdfs:Literal	rdfs:Datatype	rdfs:Class
rdfs:subClassOf		rdfs:subPropertyOf
rdfs:member	rdfs:Container	rdfs:ContainerMembershipProperty
rdfs:comment	rdfs:seeAlso	rdfs:isDefinedBy
rdfs:label		

Interpretation of a class

In model theoretic approach, a class is usually interpreted as a set (the set of its instances)

Impossible with RDF since

```
x:A rdf:type rdfs:Class. x:B rdf:type rdfs:Class.
x:A rdf:type x:B.
x:B rdf:type x:A .
```

is a legal RDF graph

With a set-based interpretation we would have

$$I(A) \in I(B) \in I(A) \in I(B) \in ....$$

But X∈X is forbidden in set theory by the regularity axiom

Interpretation of a class

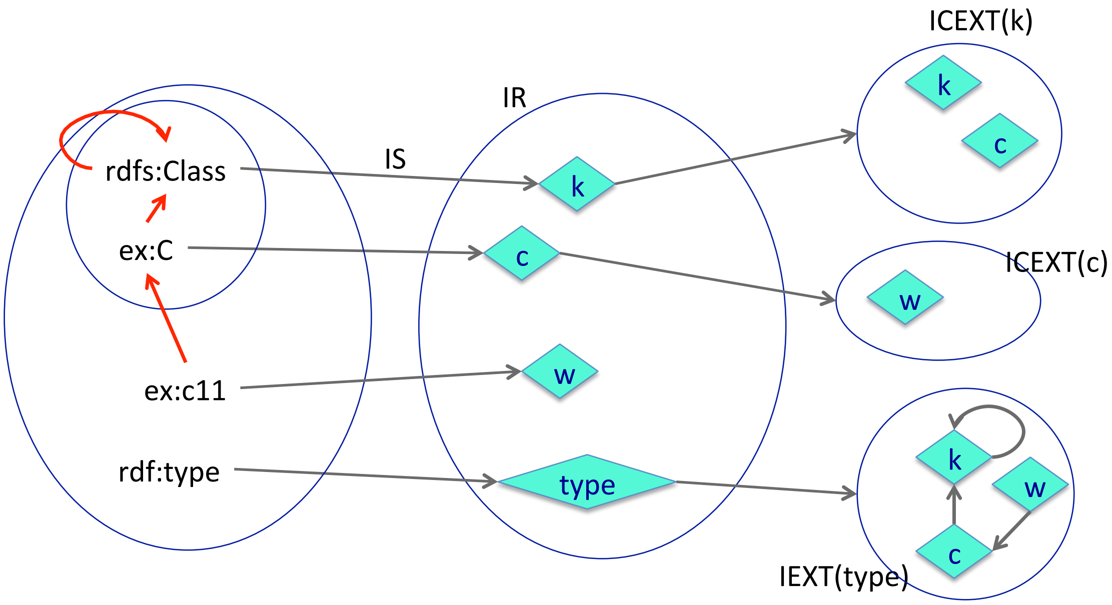
Introduce a class extension function : *ICEXT*(*class*)

ICEXT(*c*) is defined as {*x* : (*x*, *c*) is in *IEXT*(*I*(rdf:type))}

No problem with set theory, the interpretation of a class is a set of resources that are not sets.

`rdfs:Class rdf:type rdfs:Class .`

- entails *IS*(`rdfs:Class`) ∈ *ICEXT*(`rdfs:Class`)
- does not entail *IS*(`rdfs:Class`) ∈ *IS*(`rdfs:Class`)



Other semantic conditions

- (p, c) ∈ IEXT(I(`rdfs:domain`)) and (u,v) ∈ IEXT(p) ⇒ u ∈ IEXT(c)
- (p, c) ∈ IEXT(I(`rdfs:range`)) and (u,v) ∈ IEXT(p) ⇒ v ∈ IEXT(c)
- (p, q) ∈ IEXT(I(rdfs:subPropertyOf)) ⇒
  - p and q ∈ IP and IEXT(p) ⊆ IEXT(q)
- `subPropertyOf` is transitive and reflexive for properties
- (c, d) ∈ IEXT(I(rdfs:subClassOf)) ⇒
  - c and d ∈ ICEXT(I(rdfs:Class))
  - ICEXT(c) ⊆ ICEXT(d)
- `subClassOf` is transitive and reflexive for classes
- etc.

Some RDFS axiomatic triples

Must be satisfied by every rdfs-interpretation

```
rdf:type rdfs:range rdfs:Class .

rdfs:domain rdfs:range rdfs:Class .
rdfs:range rdfs:range rdfs:Class .

rdfs:subPropertyOf rdfs:range rdf:Property .
rdfs:subClassOf rdfs:range rdfs:Class .

rdf:subject rdfs:range rdfs:Resource .
rdf:predicate rdfs:range rdfs:Resource .
rdf:object rdfs:range rdfs:Resource .

rdfs:member rdfs:range rdfs:Resource .
rdf:first rdfs:range rdfs:Resource .
rdf:rest rdfs:range rdf:List .
```


Inference rules

- Using rules to generate all the entailed facts

Works for

- RDF
- RDFS

Conclusion

A family of interpretations

- basic interpretations
- D-interpretations (literals)
- RDF interpretation (semantic conditions and triples)

For represented languages

- RDFS
  - RDFS interpretation (class interpretation, semantic conditions and triples)
- more languages (OWL, etc.)
  - additional structures in the interpretation
  - additional semantic conditions and triples