

INFO-H-509 XML Technologies

RDF Schema and OWL

Stijn Vansummeren

May 10, 2012

Objectives

1. Ontologies: What and Why
2. RDF Schema syntax and semantics
3. A brief introduction to OWL

Our story so far . . .

Genome

From Wikipedia, the free encyclopedia

For a non-technical introduction to the topic, see [Introduction to genetics](#).

For other uses, see [Genome \(disambiguation\)](#).

In modern [molecular biology](#), the **genome** is the entirety of an organism's [hereditary](#) information. It is encoded either in [DNA](#) or, for [many types of virus](#), in [RNA](#).

The genome includes both the [genes](#) and the [non-coding sequences](#) of the DNA.^[1] The term was adapted in 1920 by [Hans Winkler](#), Professor of [Botany](#) at the [University of Hamburg, Germany](#). The Oxford English Dictionary suggests the name to be a [portmanteau](#) of the words **gene** and *chromosome*. A few related *-ome* words already existed, such as *biome* and *rhizome*, forming a vocabulary into which *genome* fits systematically.^[2]

- Natural language
- No structure
- **Difficult to process automatically**

Our story far ...

Current – no structure

In modern [molecular biology](#), the **genome** is the entirety of an organism's [hereditary](#) information. It is encoded either in [DNA](#) or, for [many types of virus](#), in [RNA](#).

The genome includes both the [genes](#) and the [non-coding sequences](#) of the DNA.^[1] The term was adapted in 1920 by [Hans Winkler](#), Professor of [Botany](#) at the [University of Hamburg, Germany](#). The Oxford English Dictionary suggests the name to be a [portmanteau](#) of the words **gene** and **chromosome**. A few related *-ome* words already existed, such as [biome](#) and [rhizome](#), forming a vocabulary into which *genome* fits systematically.^[2]

Future – structured by RDF ([subject](#), [predicate](#), [object](#))

b:genome	b:field	b:molecular-bio
b:DNA	b:encode	b:genes
b:DNA	b:encode	b:non-coding-seq
b:genome	b:include	b:non-coding-seq
b:genome	b:include	b:gene
b:genome	b:related-to	b:rhizome

- RDF is meant to assert knowledge ([statements](#)) about [entities](#) ([resources](#))
- By convention is clear what the [subject](#), [predicate](#), and [object](#) are
- Easier to process automatically, but a computer still does not know their meaning ...
- Can we add some [semantics](#) to the statements?

What do you mean: Semantics?

Input

```
@prefix prod: <http://www.example.org/products/> .
@prefix terms: <http://www.example.org/terms/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

prod:cam1    rdf:type      terms:digital-camera    .
prod:cam1    terms:price   150                          .
prod:nb1     rdf:type      terms:netbook                .
prod:nb1     terms:price   300                          .
prod:book1   rdf:type      terms:book                   .
prod:book1   terms:price   2.50                         .
```

How do we find all products that are digital devices?

What do you mean: Semantics?

Input

```
@prefix prod: <http://www.example.org/products/> .
@prefix terms: <http://www.example.org/terms/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
prod:cam1    rdf:type      terms:digital-camera .
prod:cam1    terms:price   150 .
prod:nb1     rdf:type      terms:netbook .
prod:nb1     terms:price   300 .
prod:book1   rdf:type      terms:book .
prod:book1   terms:price   2.50 .
```

How do we find all products that are digital devices?

```
PREFIX prod: <http://www.example.org/products/>
PREFIX terms: <http://www.example.org/terms/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
```

```
SELECT ?prod
WHERE {
  { ?prod rdf:type terms:digital-camera .}
  UNION
  { ?prod rdf:type terms:netbook .}
}
```

What do you mean: Semantics?

Input

```
@prefix prod: <http://www.example.org/products/> .
@prefix terms: <http://www.example.org/terms/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
prod:cam1    rdf:type        terms:digital-camera    .
prod:cam1    terms:price     150                      .
prod:nb1     rdf:type        terms:netbook            .
prod:nb1     terms:price     300                      .
prod:book1   rdf:type        terms:book              .
prod:book1   terms:price     2.50                    .
```

- The computer has no “knowledge of the world” stating that cameras and netbooks are digital devices
- So we have to manually encode this “knowledge of the world” in the query
- This solution is inadequate: error-prone and difficult to maintain
- **It would be better if we could tell the computer our “knowledge of the world” and let him do the reasoning!**

Reasoning by means of Inference: the General Idea

Explicitly Asserted Knowledge

I am a man
John is a man
Jane is a woman

Reasoning by means of Inference: the General Idea

Explicitly Asserted Knowledge

I am a man
John is a man
Jane is a woman

Knowledge About the World (**Ontology**)

Every man is human
Every woman is human

Reasoning by means of Inference: the General Idea

Explicitly Asserted Knowledge

I am a man
John is a man
Jane is a woman

Knowledge About the World (**Ontology**)

Every man is human
Every woman is human

Inference

I am a man
John is a man
Jane is a woman
I am human
John is human
Jane is human

Enriched Knowledge

Towards a Smarter Web

“Knowledge about the world” for our example:

- Every camera is a digital device
- Every netbook is a digital device
- Every computer is a digital device
- Every book is human-readable

Such knowledge is **set-based** (also called **class-based**)

- The set of cameras is a subset of the set of digital devices
- The set of netbooks is a subset of the set of digital devices
- The set of books is a subset of the set of human-readable objects

Definition

Ontologies (sometimes also called Schemas or Vocabularies) provide formal specifications of the **classes of objects** that inhabit “the world”, the relationships between classes, and their properties.

Reasoning by means of Inference: the General Idea (2)

Explicitly Asserted Knowledge

prod:cam1	rdf:type	terms:digital-camera
prod:cam1	terms:price	150
prod:nb1	rdf:type	terms:netbook
prod:nb1	terms:price	300
prod:book1	rdf:type	terms:book
prod:book1	terms:price	2.50

Reasoning by means of Inference: the General Idea (2)

Explicitly Asserted Knowledge

prod:cam1	rdf:type	terms:digital-camera
prod:cam1	terms:price	150
prod:nb1	rdf:type	terms:netbook
prod:nb1	terms:price	300
prod:book1	rdf:type	terms:book
prod:book1	terms:price	2.50

Knowledge About the World (**Ontology**)

Every digital camera is a digital device
Every netbook is a digital device

Reasoning by means of Inference: the General Idea (2)

Explicitly Asserted Knowledge

prod:cam1	rdf:type	terms:digital-camera
prod:cam1	terms:price	150
prod:nb1	rdf:type	terms:netbook
prod:nb1	terms:price	300
prod:book1	rdf:type	terms:book
prod:book1	terms:price	2.50

Knowledge About the World (Ontology)

Every digital camera is a digital device
Every netbook is a digital device

Inference

prod:cam1	rdf:type	terms:digital-camera
prod:cam1	terms:price	150
prod:nb1	rdf:type	terms:netbook
prod:nb1	terms:price	300
prod:book1	rdf:type	terms:book
prod:book1	terms:price	2.50
prod:cam1	rdf:type	terms:digital-device
prod:nb1	rdf:type	terms:digital-device

Enriched Knowledge

Reasoning by means of Inference: the General Idea (2)

Explicitly Asserted Knowledge

prod:cam1	rdf:type	terms:digital-camera
prod:cam1	terms:price	150
prod:nb1	rdf:type	terms:netbook
prod:nb1	terms:price	300
prod:book1	rdf:type	terms:book
prod:book1	terms:price	2.50

Knowledge About the World (Ontology)

Every digital camera is a digital device
Every netbook is a digital device

Inference

prod:cam1	rdf:type	terms:digital-camera
prod:cam1	terms:price	150
prod:nb1	rdf:type	terms:netbook
prod:nb1	terms:price	300
prod:book1	rdf:type	terms:book
prod:book1	terms:price	2.50
prod:cam1	rdf:type	terms:digital-device
prod:nb1	rdf:type	terms:digital-device

SPARQL

Enriched Knowledge

RDF Schema and OWL

How do we specify ontologies?

- RDF Schema and the Ontology Web Language (OWL) allow ontologies to be specified **in RDF itself**: they provide resources and properties in a **standard namespace** to talk about class relationships, properties of classes, ...
- They also specify how these resources and properties should be interpreted (in terms of inference)
- OWL allows more fine-grained knowledge of the world to be specified, at the cost of less efficient inference and reasoning

Example:

```
@prefix terms: <http://www.example.org/terms/> .
```

```
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
```

```
terms:digital-camera    rdfs:subClassOf    terms:digital-device .  
terms:netbook           rdfs:subClassOf    terms:digital-device .  
terms:book              rdfs:subClassOf    terms:media .  
terms:digital-device    rdfs:subClassOf    terms:electronic-device .  
terms:electronic-device rdfs:subClassOf    terms:device .
```


RDF Schema and OWL

Example Ontology in RDF Schema:

```
@prefix terms: <http://www.example.org/terms/> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .

terms:digital-camera    rdfs:subClassOf    terms:digital-device .
terms:netbook           rdfs:subClassOf    terms:digital-device .
terms:book              rdfs:subClassOf    terms:media .
```

Starting from the following asserted triples ...

```
@prefix prod: <http://www.example.org/products/> .
@prefix terms: <http://www.example.org/terms/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

prod:cam1    rdf:type    terms:digital-camera .
prod:cam1    terms:price 150 .
prod:nb1     rdf:type    terms:netbook .
prod:nb1     terms:price 300 .
prod:book1   rdf:type    terms:book .
prod:book1   terms:price 2.50 .
```

RDF Schema and OWL

Example Ontology in RDF Schema:

```
@prefix terms: <http://www.example.org/terms/> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .

terms:digital-camera    rdfs:subClassOf    terms:digital-device .
terms:netbook           rdfs:subClassOf    terms:digital-device .
terms:book              rdfs:subClassOf    terms:media .
```

... an RDF Schema-aware processor will infer the following triples (in purple)

```
@prefix prod: <http://www.example.org/products/> .
@prefix terms: <http://www.example.org/terms/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

prod:cam1    rdf:type    terms:digital-camera    .
prod:cam1    terms:price 150                      .
prod:nb1     rdf:type    terms:netbook           .
prod:nb1     terms:price 300                      .
prod:book1   rdf:type    terms:book              .
prod:book1   terms:price 2.50                    .
prod:cam1    rdf:type    terms:digital-device    .
prod:nb1     rdf:type    terms:digital-device    .
prod:book1   rdf:type    terms:media             .
```

RDF Schema and OWL

Example Ontology in RDF Schema:

```
@prefix terms: <http://www.example.org/terms/> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .

terms:digital-camera    rdfs:subClassOf    terms:digital-device .
terms:netbook           rdfs:subClassOf    terms:digital-device .
terms:book              rdfs:subClassOf    terms:media .
```

So finding all products that are digital devices becomes easy (assuming our SPARQL engine is RDFS-aware):

```
PREFIX prod: <http://www.example.org/products/>
PREFIX terms: <http://www.example.org/terms/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

SELECT ?prod
WHERE {
    ?prod rdf:type terms:digital-device .
}
```

A chalkboard with a gold frame. The chalkboard is dark blue/black and has some faint, light blue chalk marks. The text "Part I: RDF Schema" is written in white on the chalkboard.

Part I: RDF Schema

Diving right in: `rdfs:subClassOf`

- Like all RDF Schema resources and properties, `subClassOf` lives in the namespace `http://www.w3.org/1999/02/22-rdf-syntax-ns#`
- `X rdfs:subClassOf Y` is used to specify that all members of a class `X` are also members of a class `Y`

Inference Rule [I-SUBCLASS]

If `?A rdfs:subClassOf ?B .`
And `?x rdf:type ?A .`
Then add `?x rdf:type ?B .`

Example

```
:C1    rdfs:subClassOf    :C2 .  
:C2    rdfs:subClassOf    :C3 .  
:x      rdf:type           :C1 .
```

Diving right in: `rdfs:subClassOf`

- Like all RDF Schema resources and properties, `subClassOf` lives in the namespace `http://www.w3.org/1999/02/22-rdf-syntax-ns#`
- `X rdfs:subClassOf Y` is used to specify that all members of a class `X` are also members of a class `Y`

Inference Rule [I-SUBCLASS]

If `?A rdfs:subClassOf ?B .`
And `?x rdf:type ?A .`
Then add `?x rdf:type ?B .`

Example

```
:C1    rdfs:subClassOf    :C2 .  
:C2    rdfs:subClassOf    :C3 .  
:x      rdf:type           :C1 .  
:x      rdf:type           :C2 .
```

Diving right in: `rdfs:subClassOf`

- Like all RDF Schema resources and properties, `subClassOf` lives in the namespace `http://www.w3.org/1999/02/22-rdf-syntax-ns#`
- `X rdfs:subClassOf Y` is used to specify that all members of a class `X` are also members of a class `Y`

Inference Rule [I-SUBCLASS]

If `?A rdfs:subClassOf ?B .`
And `?x rdf:type ?A .`
Then add `?x rdf:type ?B .`

Example

<code>:C1</code>	<code>rdfs:subClassOf</code>	<code>:C2 .</code>
<code>:C2</code>	<code>rdfs:subClassOf</code>	<code>:C3 .</code>
<code>:x</code>	<code>rdf:type</code>	<code>:C1 .</code>
<code>:x</code>	<code>rdf:type</code>	<code>:C2 .</code>
<code>:x</code>	<code>rdf:type</code>	<code>:C3 .</code>

Diving right in: `rdfs:subPropertyOf`

- `X rdfs:subPropertyOf Y` is used to specify that all resources with property `X` also have property `Y`

Inference Rule [I-SUBPROP]

If `?P rdfs:subPropertyOf ?R .`
And `?x ?P ?y .`
Then add `?x ?R ?y .`

Example

`:P rdfs:subPropertyOf :R .`
`:John :P :o1 .`

Diving right in: `rdfs:subPropertyOf`

- `X rdfs:subPropertyOf Y` is used to specify that all resources with property `X` also have property `Y`

Inference Rule [I-SUBPROP]

If `?P rdfs:subPropertyOf ?R .`
And `?x ?P ?y .`
Then add `?x ?R ?y .`

Example

<code>:P</code>	<code>rdfs:subPropertyOf</code>	<code>:R .</code>
<code>:John</code>	<code>:P</code>	<code>:o1 .</code>
<code>:John</code>	<code>:R</code>	<code>:o1 .</code>

Diving right in: `rdfs:domain`

- `P rdfs:domain C` is used to specify that all resources with outgoing property *P* belong to class *C*

Inference Rule [I-DOMAIN]

If `?P rdfs:domain ?C .`
And `?x ?P ?y .`
Then add `?x rdf:type ?C .`

Example

```
:P            rdfs:domain    :C1 .  
:John        :P                :o1 .
```

Diving right in: `rdfs:domain`

- `P rdfs:domain C` is used to specify that all resources with outgoing property *P* belong to class *C*

Inference Rule [I-DOMAIN]

If `?P rdfs:domain ?C .`
And `?x ?P ?y .`
Then add `?x rdf:type ?C .`

Example

<code>:P</code>	<code>rdfs:domain</code>	<code>:C1 .</code>
<code>:John</code>	<code>:P</code>	<code>:o1 .</code>
<code>:John</code>	<code>rdf:type</code>	<code>:C1 .</code>

Interaction of Rules

Consider the following example. What new triples are inferred?

```
:MarriedWoman    rdfs:subClassOf    :Woman .  
:maidenName      rdfs:domain        :MarriedWoman .  
  
:Karen           :maidenName        "Stephens" .
```

Interaction of Rules

Consider the following example. What new triples are inferred?

:MarriedWoman	rdfs:subClassOf	:Woman	.
:maidenName	rdfs:domain	:MarriedWoman	.
:Karen	:maidenName	"Stephens"	.
:Karen	rdf:type	:MarriedWoman	.

Interaction of Rules

Consider the following example. What new triples are inferred?

:MarriedWoman	rdfs:subClassOf	:Woman	.
:maidenName	rdfs:domain	:MarriedWoman	.
:Karen	:maidenName	"Stephens"	.
:Karen	rdf:type	:MarriedWoman	.
:Karen	rdf:type	:Woman	.

Interaction of Rules

Consider the following example. What new triples are inferred?

```
:MarriedWoman    rdfs:subClassOf    :Woman .
:maidenName      rdfs:domain        :MarriedWoman .

:Karen           :maidenName        "Stephens" .
:Karen           rdf:type            :MarriedWoman .
:Karen           rdf:type            :Woman .
```

This illustrates the following effect

Inference Rule [Interaction-1]

```
If           ?P rdfs:domain ?D .
And          ?D rdfs:subClassOf ?C .
Then add     ?P rdfs:domain ?C .
```

Diving right in: `rdfs:range`

- `P rdfs:range C` is used to specify that all resources with incoming property *P* belong to class *C*

Inference Rule [I-RANGE]

If `?P rdfs:range ?C .`
And `?x ?P ?y .`
Then add `?y rdf:type ?C .`

Example

<code>:P</code>	<code>rdfs:domain</code>	<code>:C1 .</code>
<code>:P</code>	<code>rdfs:range</code>	<code>:C2 .</code>
<code>:John</code>	<code>:P</code>	<code>:o1 .</code>

Diving right in: `rdfs:range`

- `P rdfs:range C` is used to specify that all resources with incoming property *P* belong to class *C*

Inference Rule [I-RANGE]

If `?P rdfs:range ?C .`
And `?x ?P ?y .`
Then add `?y rdf:type ?C .`

Example

<code>:P</code>	<code>rdfs:domain</code>	<code>:C1 .</code>
<code>:P</code>	<code>rdfs:range</code>	<code>:C2 .</code>
<code>:John</code>	<code>:P</code>	<code>:o1 .</code>
<code>:John</code>	<code>rdf:type</code>	<code>:C1 .</code>
<code>:o1</code>	<code>rdf:type</code>	<code>:C2 .</code>


Meta-Concepts

RDF Schema also provides the following resources:

- `rdfs:Resource` — The class of all resources. Every resource is an instance of this class
- `rdfs:Class` — The class of all classes.
- `rdfs:Literal` — The class of all literals.
- `rdfs:Property` — The class of all properties.
- `rdfs:Datatype` — The class of all datatypes.

RDF Schema is equipped with the following built-in triples (among others)

<code>rdfs:subClassOf</code>	<code>rdfs:domain</code>	<code>rdfs:Class</code>	<code>.</code>
<code>rdfs:subClassOf</code>	<code>rdfs:range</code>	<code>rdfs:Class</code>	<code>.</code>
<code>rdfs:subClassOf</code>	<code>rdf:type</code>	<code>rdfs:Property</code>	<code>.</code>
<code>rdfs:subPropertyOf</code>	<code>rdfs:domain</code>	<code>rdfs:Property</code>	<code>.</code>
<code>rdfs:subPropertyOf</code>	<code>rdfs:range</code>	<code>rdfs:Property</code>	<code>.</code>
<code>rdfs:subPropertyOf</code>	<code>rdf:type</code>	<code>rdfs:Property</code>	<code>.</code>
<code>rdfs:domain</code>	<code>rdf:type</code>	<code>rdfs:Property</code>	<code>.</code>
<code>rdfs:range</code>	<code>rdf:type</code>	<code>rdfs:Property</code>	<code>.</code>
<code>rdfs:Resource</code>	<code>rdf:type</code>	<code>rdfs:Class</code>	<code>.</code>
<code>rdfs:Class</code>	<code>rdfs:subClassOf</code>	<code>rdfs:Resource</code>	<code>.</code>

A rectangular chalkboard with a thick, gold-colored frame. The chalkboard surface is dark blue or black and shows faint, light-colored chalk marks and scratches. Centered on the board is the text "Part II: OWL" in a white, sans-serif font.

Part II: OWL

Limitations of RDF Schema

RDF Schema allows us to represent **some** ontological knowledge:

- Typed hierarchies using classes and subclasses, properties and subproperties
- Domain and range restrictions
- Describing instances of classes (through subclasses and `rdf:type`)

Sometimes we want more:

- **Local scope of properties** Using `rdfs:range` and `rdfs:domain` we can't state that cows only eat plants while other animals may eat meat too.
- **Disjointness of classes**. We can't state, for example, that `terms:male` and `terms:female` do not have any members in common.
- **Special characteristics of properties**. Sometimes it is convenient to be able to say that a property is **transitive** (like “greater than”), **unique** (like “father of”), or the **inverse** of another property (like “father of” and “child of”).
- **Cardinality restrictions** like “a person has exactly 2 parents”

OWL: Ontology Web Language

The **Ontology Web Language (OWL)** allows us to talk about such things (among others)

OWL: Ontology Web Language

The **Ontology Web Language (OWL)** allows us to talk about such things (among others)



There is always a trade-off between expressiveness and efficient reasoning support:

- The more expressive a language ...
- The more inefficient the inferencing becomes ...
- ... it may even become **undecidable**!

OWL: Ontology Web Language

There are actually 3 OWL sublanguages:

- **OWL Full**: all features, but undecidable
- **OWL DL**^a: expressive, but not entirely compatible with RDF Schema; efficient reasoning support
- **OWL Lite**: the least expressive, but compatible with RDF Schema; efficient reasoning support

^aDL=Description Logic

Here, we will illustrate some features of OWL Lite

Special Characteristics of Properties: Inverses

- All OWL resources and properties live in the namespace `http://www.w3.org/2002/07/owl#`
- `P owl:inverseOf R` is used to specify that property *R* is the inverse of property *P* (and vice versa)

Inference Rule [I-INVERSE]

```
If           ?P owl:inverseOf ?R .  
And          ?x ?P ?y .  
Then add    ?y ?R ?x .
```

```
If           ?P owl:inverseOf ?R .  
And          ?x ?R ?y .  
Then add    ?y ?P ?x .
```

Example

```
:fatherOf    owl:inverseOf    :childOf .  
:Jake        :fatherOf          :John    .
```


Special Characteristics of Properties: Inverses

- All OWL resources and properties live in the namespace `http://www.w3.org/2002/07/owl#`
- `P owl:inverseOf R` is used to specify that property *R* is the inverse of property *P* (and vice versa)

Inference Rule [I-INVERSE]

```
If           ?P owl:inverseOf ?R .  
And          ?x ?P ?y .  
Then add    ?y ?R ?x .
```

```
If           ?P owl:inverseOf ?R .  
And          ?x ?R ?y .  
Then add    ?y ?P ?x .
```

Example

<code>:fatherOf</code>	<code>owl:inverseOf</code>	<code>:childOf</code>	<code>.</code>
<code>:Jake</code>	<code>:fatherOf</code>	<code>:John</code>	<code>.</code>
<code>:John</code>	<code>:childOf</code>	<code>:Jake</code>	<code>.</code>

Special Characteristics of Properties: Symmetry

- `P rdf:type owl:SymmetricProperty` is used to specify that property *P* is a symmetric property

Inference Rule [I-SYMMETRIC]

If `?P rdf:type owl:SymmetricProperty .`
And `?x ?P ?y .`
Then add `?y ?P ?x .`

Example

```
:marriedTo    rdf:type    owl:SymmetricProperty .  
:Jake         :marriedTo  :Eve
```

Special Characteristics of Properties: Symmetry

- `P rdf:type owl:SymmetricProperty` is used to specify that property *P* is a symmetric property

Inference Rule [I-SYMMETRIC]

If `?P rdf:type owl:SymmetricProperty .`
And `?x ?P ?y .`
Then add `?y ?P ?x .`

Example

<code>:marriedTo</code>	<code>rdf:type</code>	<code>owl:SymmetricProperty</code>	<code>.</code>
<code>:Jake</code>	<code>:marriedTo</code>	<code>:Eve</code>	<code>.</code>
<code>:Eve</code>	<code>:marriedTo</code>	<code>:Jake</code>	<code>.</code>

Special Characteristics of Properties: Transitivity

- `P rdf:type owl:TransitiveProperty` is used to specify that property P is a Transitive property

Inference Rule [I-TRANSITIVE]

```
If      ?P rdf:type owl:TransitiveProperty .  
And     ?x ?P ?y .  
And     ?y ?P ?z .  
Then add ?x ?P ?z .
```

Example

```
:ancestor  rdf:type    owl:TransitiveProperty .  
:Jake      :ancestor  :John                      .  
:Jill      :ancestor  :Jake                      .
```

Special Characteristics of Properties: Transitivity

- `P rdf:type owl:TransitiveProperty` is used to specify that property *P* is a Transitive property

Inference Rule [I-TRANSITIVE]

```
If      ?P rdf:type owl:TransitiveProperty .  
And     ?x ?P ?y .  
And     ?y ?P ?z .  
Then add ?x ?P ?z .
```

Example

```
:ancestor  rdf:type    owl:TransitiveProperty .  
:Jake      :ancestor  :John  
:Jill      :ancestor  :Jake  
:Jill      :ancestor  :John
```

Asserting Equivalence of Classes

- It is always possible that we have used a specific URI to identify a particular concept while someone else has used a different URI for the same concept
- `C owl:equivalentClass D` is used to specify that every member of class *C* is a member of class *D*, and vice versa

Semantics is given by

```
owl:equivalentClass rdf:type owl:SymmetricProperty .  
owl:equivalentClass rdf:type owl:TransitiveProperty .  
owl:equivalentClass rdfs:subPropertyOf rdfs:subClassOf .
```

Example

```
:Man    owl:equivalentClass    :Homme .  
:Jake   rdf:type                  :Man    .
```

Asserting Equivalence of Classes

- It is always possible that we have used a specific URI to identify a particular concept while someone else has used a different URI for the same concept
- `C owl:equivalentClass D` is used to specify that every member of class *C* is a member of class *D*, and vice versa

Semantics is given by

```
owl:equivalentClass rdf:type owl:SymmetricProperty .  
owl:equivalentClass rdf:type owl:TransitiveProperty .  
owl:equivalentClass rdfs:subPropertyOf rdfs:subClassOf .
```

Example

```
:Man      owl:equivalentClass  :Homme .  
:Jake     rdf:type               :Man .  
:Homme     owl:equivalentclass  :Man .  
:Man      rdfs:subClassOf        :Homme .  
:Homme     rdfs:subClassOf        :Man .  
:Jake     rdf:type               :Homme .
```

Asserting Equivalence of Predicates

- It is always possible that we have used a specific URI to identify a particular concept while someone else has used a different URI for the same concept
- `P owl:equivalentProperty R` is used to specify that properties *P* and *R* are equivalent

Semantics is given by

```
owl:equivalentProperty rdf:type owl:SymmetricProperty .  
owl:equivalentProperty rdf:type owl:TransitiveProperty .  
owl:equivalentProperty rdfs:subPropertyOf rdfs:subPropertyOf .
```

```
:fatherOf    owl:equivalentProperty    :père .  
:Jake        :père                        :John .
```


Asserting Equivalence of Predicates

- It is always possible that we have used a specific URI to identify a particular concept while someone else has used a different URI for the same concept
- `P owl:equivalentProperty R` is used to specify that properties *P* and *R* are equivalent

Semantics is given by

```
owl:equivalentProperty rdf:type owl:SymmetricProperty .  
owl:equivalentProperty rdf:type owl:TransitiveProperty .  
owl:equivalentProperty rdfs:subPropertyOf rdfs:subPropertyOf .
```

<code>:fatherOf</code>	<code>owl:equivalentProperty</code>	<code>:père</code>	<code>.</code>
<code>:Jake</code>	<code>:père</code>	<code>:John</code>	<code>.</code>
<code>:père</code>	<code>owl:equivalentProperty</code>	<code>:fatherOf</code>	<code>.</code>
<code>:père</code>	<code>rdfs:subPropertyOf</code>	<code>:fatherOf</code>	<code>.</code>
<code>:fatherOf</code>	<code>rdfs:subPropertyOf</code>	<code>:père</code>	<code>.</code>
<code>:Jake</code>	<code>:fatherOf</code>	<code>:John</code>	<code>.</code>

Asserting Equivalence of Resources

- It is always possible that we have used a specific URI to identify a particular concept while someone else has used a different URI for the same concept
- `x owl:sameAs y` is used to specify that `x` and `y` are the same resources

Semantics is given by

```
owl:sameAs rdf:type owl:SymmetricProperty .
```

```
If          ?x owl:sameAs ?y .
```

```
And         ?x ?P ?z .
```

```
Then add    ?y ?P ?z .
```

```
If          ?x owl:sameAs ?y .
```

```
And         ?z ?P ?x .
```

```
Then add    ?z ?P ?y .
```

Functional Properties

- `P rdf:type owl:FunctionalProperty` is used to specify that P can only take one object for a particular subject

Inference Rule:

```
If      ?P rdf:type owl:FunctionalProperty .
And     ?x ?P ?y .
And     ?x ?P ?z .
Then add ?y owl:sameAs ?z .
```

Example

```
:hasFather    rdf:type      owl:FunctionalProperty .
:John         :hasFather    :Jake
:John         :hasFather    ex:Jake-J
```

Functional Properties

- `P rdf:type owl:FunctionalProperty` is used to specify that P can only take one object for a particular subject

Inference Rule:

```
If      ?P rdf:type owl:FunctionalProperty .
And     ?x ?P ?y .
And     ?x ?P ?z .
Then add ?y owl:sameAs ?z .
```

Example

<code>:hasFather</code>	<code>rdf:type</code>	<code>owl:FunctionalProperty</code>	<code>.</code>
<code>:John</code>	<code>:hasFather</code>	<code>:Jake</code>	<code>.</code>
<code>:John</code>	<code>:hasFather</code>	<code>ex:Jake-J</code>	<code>.</code>
<code>:Jake</code>	<code>owl:sameAs</code>	<code>ex:Jake-J</code>	<code>.</code>

Inverse Functional Properties

- `P rdf:type owl:InverseFunctionalProperty` is used to specify that P can only take one subject for a particular object

Inference Rule:

```
If      ?P rdf:type owl:InverseFunctionalProperty .  
And     ?y ?P ?x .  
And     ?z ?P ?x .  
Then add ?y owl:sameAs ?z .
```