1. Motivation for RDF: the Semantic Web, Linked Data

2. The RDF Data Model

3. RDF Serialization formats

4. Simple ontologies in RDFS
Part I: Motivation
THE WEB IN THE RECENT PAST
THE WEB IN THE RECENT PAST: UNSTRUCTURED DATA

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 automatically process the knowledge encoded in these pages
**WHAT DO YOU MEAN: NO STRUCTURE?**

Doesn’t HTML have structure?

In modern [molecular biology](https://en.wikipedia.org/wiki/Molecular_biology), the genome includes both the [genes](https://en.wikipedia.org/wiki/Gene) and the [Diploid](https://en.wikipedia.org/wiki/Diploid). Some organisms have multiple copies of chromosomes. Both the number of [Base pair](https://en.wikipedia.org/wiki/Base_pair) and the number of [Base pair](https://en.wikipedia.org/wiki/Base_pair) per book. An analogy to the human genome stored on DNA is that of instructions stored in a library:

- The library would contain 46 books (chromosomes).
- The books range in size from 400 to 3340 pages (genes).
- Which is 48 to 250 million letters (A,C,G,T) per book.
- Hence the library contains over six billion letters total.
- The library fits into a cell nucleus the size of a pinpoint.
- A copy of the library (all 46 books) is contained in almost every cell of our body.

- Yes, but that structure only describes presentation
- It does not structure the information (DNA, genes, ...)

WHAT DO YOU MEAN: NO STRUCTURE?

Doesn’t XML have structure?

2 Examples with same “structure”:

```xml
<lecturer name="John Doe">
  <teaches>XML Technologies</teaches>
</lecturer>

<course name="XML Technologies">
  <lecturer>John Doe</lecturer>
</course>
```

- Yes, but that structure specifies only tag nesting (i.e., hierarchy)
- The tag names themselves are meaningless to a computer
- And it is impossible to unambiguously assign meaning to them ...
- … unless you explicitly program a specific application (e.g. lecturers and courses)
WHAT DO YOU MEAN: NO STRUCTURE?

Doesn’t XML have structure?

For a computer, this is the same as:

```xml
<padoijfa dkfjao="Imla Loqa">
  <Blablahdo>LMX Idjflal dld</Blablahdo>
</padoijfa>

<Rfjaomdla dkfjao="LMX Idjflal dld">
  <padoijfa>Imla Loqa</padoijfa>
</Rfjaomdla>
```

- Yes, but that structure specifies only tag nesting (i.e., hierarchy)
- The tag names themselves are meaningless to a computer
- And it is impossible to unambiguously assign meaning to them ...
- ... unless you explicitly program a **specific application** (e.g. lecturers and courses)
"most of the Web’s content... is designed for humans to read, not for computer programs to manipulate meaningfully. Computers can adeptly parse Web pages for layout and routine processing – here a header, there a link to another page but, in general, computers have no reliable way to process the Semantics...’, or the meaning of the content of the page.”

(Berners-Lee, Hendler and Lissila, 2001)
THE SEMANTIC WEB VISION

- Move from a Web of Documents to a Web of Knowledge
- This knowledge should be represented to have unambiguous semantics so that computers can process it unambiguously.
- This should make reasoning less programming-intensive: “Agents” (computer programs) should be able to reason with this knowledge without (too much) explicit programming.
- See e.g. “The Semantic Web” by Berners-Lee et. al. in Scientific American Magazine (link available on Course Webpage).
John posts a profile on the Web (name, email address, profession) ...he also mentions that he has a tree nut allergy
John posts a profile on the Web (name, email address, profession) ...he also mentions that he has a **tree nut allergy**
The World Health Organization mentions on its site that people with tree nut allergy are usually also allergic to seeds, including sesamy seeds.
Restaurant “The good life” mentions that its *house salad contains sesame seeds.*
John’s web agent concludes that when eating at the “The good life”, John should not take the house salad.
We want to describe knowledge in such a way that (computers) can automatically reason with this.

This has been the research topic in AI for decades ... 

...and a fundamental question in science throughout history ...

...and it is really, really hard! (even impossible).
400 BC: birth of philosophical ontology

- Greek philosopher Plato asked: What is reality? Which things can be said to “exist”. What is the true nature of things?
- Philosophical ontology = the study of existence and being as such, and of the fundamental classes and relationships of existing things.
350 BC: first approach to classification

- Aristotle (384-322 BC) held the view that reality is not given in universal ideas; we should carefully observe reality, and describe our observations.

- Hereto, Aristotle developed ten (exhaustive) categories to classify all things that may exist, and subcategories to further specify them.

- Example: Aristotle’s category of animals was composed of rational ones (humans) and irrational ones (beasts).
Taxonomy: the science

- The basic idea of classification has fueled many influential scientific models in later centuries.
- Carolus Linnaeus (1707-1778) laid the basis for modern biological classification by introducing Linnaen taxonomy.
- The term taxonomy now refers to the science of classification.
- Classifications are often tree-shaped, although not necessarily so (e.g., periodic table of elements).
Automated reasoning

- Traditionally, knowledge was recorded to be accessed and processed by human beings.
- Starting with Aristotle, however, the observation grew that logical deduction can itself be formalized and cast into a set of rules in a way reminiscent of arithmetics.
- Leibniz (1646-1716) formulated the dream to resolve conflicts in scientific discourse by just calculating the correct answer.

“If controversies were to arise, there would be no more need of disputation between two philosophers than between two accountants. For it would suffice to take their pencils in hands, sit down to their slates, and to say to each other: let us calculate.”
Automated reasoning

- In summer 1956, John McCarthy and other leading researchers, inspired by the accessibility of digital computers, explored the possibility of using computers to simulate or generate intelligent behavior.

- In the course of this event, the term artificial intelligence was coined.

- Predominant in this context was the task of deducing new knowledge from known facts.
“A body of formally represented knowledge is based on a conceptualization: the objects, concepts, and other entities that are presumed to exist in some area of interest and the relationships that hold them. A conceptualization is an abstract, simplified view of the world that we wish to represent for some purpose.

Every knowledge base, knowledge-based system, or knowledge-level agent is committed to some conceptualization, explicitly or implicitly.

An ontology is an explicit specification of a conceptualization.

Thomas R. Gruber, 1993

Ontologies in computer science

- Ultimately, computers are only working with symbols (a.k.a terms) that, to them do not have meaning.
- Therefore, we need to rigorously describe, for a given set of symbols, what properties these symbols are, how they relate to each other, and how they are collected in groups (“classes”).
- Such rigorous descriptions are called ontologies in computer science/AI.
What do ontologies talk about?

- We usually have:
  - Symbols that represent **individuals** (John, sesame seed)
  - Symbols that represent **classes** (Groups of individuals like people with tree nut allergy; all people with seed allergy)
  - Relationships between instances/groups and instances/groups (people with nut allergy are also allergic to seeds).
BASIC IDEA OF SEMANTIC WEB

Copyright Pascal Hitzler; reused under the terms of the Creative Commons CC-BY license.
KNOWLEDGE REPRESENTATION ON THE SEMANTIC WEB (2005 VISION)
data exchange/simple reasoning
(this lecture)
advanced reasoning (next lectures)
data exchange/simple reasoning (this lecture)
KNOWLEDGE REPRESENTATION ON THE SEMANTIC WEB (2013)
The original vision of the Semantic Web was too futuristic and created the wrong expectations; attaining this vision would require solving AI-complete problems.

But the research and development that was spurred after the publication of the Semantic Web article has gone a significant way towards realizing the vision of machine-treatable data through the Linking Open Data initiative.

Linked Data is a more principled way of publishing data on the web, linking it to other data (without the sci-fi stuff).
THE CURRENT WEB: WITH STRUCTURED DATA

Nieuws over obama

Amerikaanse roddejaars: "Hommeles ten huize Obama"
Het Laatste Nieuws - 1 dag geleden
Het huwelijk van de Amerikaanse president Barack Obama (52) en first lady Michelle (50) staat op de helling. Dat schrijft The National Enquirer...

Barack Obama - Wikipedia
nl.wikipedia.org/wiki/Barack_Obama
Barack Hussein Obama II (Honolulu (Hawaii), 4 augustus 1961) is de 44e en huidige president van de Verenigde Staten. Hij is de eerste Amerikaan van (deels)...
Biografie - Politiek - Presidentschap - Schrijverswerk

Barack Obama - Wikipedia, the free encyclopedia
en.wikipedia.org/wiki/Barack_Obama
Vertaal deze pagina
Barack Hussein Obama II is de 44e en current President of the United States, and the first African American to hold the office. Born in Honolulu, Hawaii, ...

Barack Obama (BarackObama) on Twitter
https://twitter.com/BarackObama
Vertaal deze pagina
The latest from Barack Obama (@BarackObama). This account is run by Organizing for Action staff. Tweets from the President are signed -bo. Washington, DC.

Barack Obama
www.barackobama.com/
Vertaal deze pagina
Official re-election campaign website of President Barack Obama provides the latest updates, election news, videos, local events and ways to volunteer and ...

Barack Obama - Washington, DC - Politician | Facebook
https://www.facebook.com/barackobama
Vertaal deze pagina
Barack Obama, Washington, DC. 38779320 likes - 634613 talking about this. This page is run by Organizing for Action. To visit the White House Facebook page, ...

Barack Obama
3.801.741 volgers op Google+
Barack Hussein Obama II is de 44e en huidige president van de Verenigde Staten. Hij is de eerste Amerikaan van Afrikaanse afkomst in deze functie. Wikipedia

Geboren: 4 augustus 1961 (52 jaar), Honolulu, Hawai, Verenigde Staten
Echtgenote: Michelle Obama (geb. 1964)
Presidentiële termijn: 20 januari 2009 –
Ouders: Ann Dunham, Barack Obama Sr.
Kinderen: Natasha Obama, Malia Ann Obama
Broers/zussen: Malik Abongo Obama, Maya Soetoro-Ng,

Recente Google+ berichten
THE CURRENT WEB: WITH STRUCTURED DATA

Structured information, based on Linked Data
Many datasets and ontologies are openly made available and \textit{interlinked} as RDF data.
Part II: The RDF Data Model
Before we begin.

Some of the following examples have been copied from:

- The W3C RDF Primer at [http://www.w3.org/TR/rdf-primer/](http://www.w3.org/TR/rdf-primer/)

Disclaimer
The Resource Description Framework (RDF) is a general method for describing knowledge in the form of a directed, labeled graph.

- Nodes represent the “entities” or “resources” we are describing; or literal values;
- Edges relate resources to other resources, or to literal values.

Example (not valid RDF):

![Graph Diagram]

- DNA
  \[\text{encoded_by}\] genome
  \[\text{occurs_in}\] molecular_biology
The Resource Description Framework (RDF) is a general method for describing knowledge in the form of a directed, labeled graph.

- Nodes represent the “entities” or “resources” we are describing; or literal values;
- Edges relate resources to other resources, or to literal values.

Example (not valid RDF):

```
staff_id:85740

creator

http://www.example.org/index.html

creation-date: August 16, 1999

language: en
```
The Resource Description Framework (RDF) is a general method for describing knowledge in the form of a directed, labeled graph.

- Nodes represent the “entities” or “resources” we are describing; or literal values;
- Edges relate resources to other resources, or to literal values.

Example (not valid RDF):
The Resource Description Framework (RDF) is a general method for describing knowledge in the form of a directed, labeled graph.

- Nodes represent the “entities” or “resources” we are describing; or literal values;
- Edges relate resources to other resources, or to literal values.

Example (not valid RDF):

```
vincent_donofrio

starred_in

..._floor

same_plot_as

"1999"

..._matrix

released_in

"1999"

...movie

...is_a

...tv_show

law_&_order

...is_a
```
There are two kinds of nodes:

- Nodes that represent the ‘things’ we are describing: these are called resources or entities.
- Nodes that represent particular values of a property (like the string John Doe, the integer 1999, the data August 16, 1999 and so on): these are called literals or literal nodes.
- Literals can only have incoming edges, whereas resources can have both incoming and outgoing edges.

Example (not valid RDF):

```
Lecturer is_a John_Doe
"XML Technologies" title INFO_H_509
"John Doe" has_name

John_Doe teaches "XML Technologies"
INFO_H_509 is_a course

legend: resource literal
```
The edge labels are called **predicates** or **properties**

Example (not valid RDF):

```
Lecturer is_a John_Doe has_name "John Doe"

"XML Technologies" title INFO_H_509 is_a course
```

**Legend:**
- resource
- literal
- property
Consider again our genome information

The term “genome” can mean different things:

- Genome — the totality of genetic material carried by an organism
- Genome (novel) — science fiction novel by Sergey Lukyanenko
- Genome — a superior humanoid race in Square’s console role-playing game Final Fantasy IX
- ...

In order to unambiguously refer to an entity, it needs a a unique name
Question:
How do we unambiguously name things in a way that allows everyone to invent their own names without centralized agreement?
Question:
How do we unambiguously name things in a way that allows everyone to invent their own names without centralized agreement?

Answer:
- Use Uniform Resource Identifiers!
- For instance, if I own the domain http://www.example.org I am free to create URLs starting with that address.

- **URN example:** urn:isbn:0-486-27557-4
- **URL example:** http://www.example.org/staffid/85740
For this reason, RDF requires:
- every resource to be a URI
- every property to be a URI

Example (this time a valid RDF graph):

Note that these URIs are used to identify entities and properties. Even if they take the form of a URL, they need not be dereferencable: if you type them in the location bar of a browser you may get an error!
For this reason, RDF requires:

- every resource to be a URI
- every property to be a URI

Another Example (also a valid RDF graph):

http://www.example.org/staffid/85740
http://purl.org/dc/elements/1.1/creator
http://www.example.org/index.html
http://www.example.org/terms/creation-date
August 16, 1999
http://www.example.org/terms/language
en
For this reason, RDF requires:
- every resource to be a URI
- every property to be a URI

Another Example (also a valid RDF graph):

In essence: similar to namespaces in XML
In RDF, the only relationships that we can make is between two URIs (resources), or between a URI and a literal.

To express \( n \)-ary relationships, we need to create extra resources.

**Example: street address as a literal value**

```
http://www.example.org/staffid/85740
```

```
http://www.example.org/terms/address
```

```
1501 Grant Av., Bedford, Massachusetts, 0173
```

```
In RDF, the only relationships that we can make is between two URIs (resources), or between a URI and a literal.

To express \( n \)-ary relationships, we need to create extra resources.

Example: street address as its own resource (\( n \)-ary relation)
Sometimes inventing a new URI for every resource becomes tedious.

**Example: street address**

```
http://www.example.org/staffid/85740
http://www.example.org/terms/address
http://www.example.org/addressid/85740
http://www.example.org/terms/city
http://www.example.org/terms/street
http://www.example.org/terms/state
http://www.example.org/terms/postalcode
Bedford
1501 Grant Avenue
Massachusetts
0173
```
Sometimes inventing a new URI for every resource becomes tedious
RDF also allows for nodes without URIs: these are called blank nodes or anonymous resources
A blank node hence denotes an entity for which we do not have or do not want to create a globally unique name

Example: street address with blank node

```
http://www.example.org/staffid/85740

http://www.example.org/terms/address

http://www.example.org/terms/city

http://www.example.org/terms/postalcode

http://www.example.org/terms/street

http://www.example.org/terms/state

Bedford

0173

1501 Grant Avenue

Massachusetts
```
THE RDF GRAPH: TYPED LITERALS

- Without further information, it is often difficult to discern how a literal should be interpreted: is it an integer, a string, a date, ...?
- An **RDF Typed Literal** is formed by pairing a literal with a URI that identifies a particular datatype.
- RDF itself does not specify datatypes, but one can use for example the XML Schema datatypes.
- Some XML Schema datatypes, like **xsd:duration** and **xsd:QName** cannot be used in RDF.

**Example:**

```
http://www.example.org/index.html

http://www.example.org/terms/creation-date

1999-08-16^^http://www.w3.org/2001/XMLSchema#date
```
Without further information, it is often difficult to discern how a literal should be interpreted: is it an integer, a string, a date, …?

An RDF Typed Literal is formed by pairing a literal with a URI that identifies a particular datatype.

RDF itself does not specify datatypes, but one can use for example the XML Schema datatypes.

Some XML Schema datatypes, like `xsd:duration` and `xsd:QName` cannot be used in RDF.

**Example:**

```
http://www.example.org/staffid/85740
27^^http://www.w3.org/2001/XMLSchema#integer
```

```
http://www.example.org/terms/age
```
THE RDF GRAPH: THE “IS A” RELATIONSHIP

- To express statements of the form
  
  "Resource A is an instance of resource B"
  
  like “John is a lecturer” or “Law & Order is a TV show”, RDF has reserved the following URI:
  
  http://www.w3.org/1999/02/22-rdf-syntax-ns#type

- We will see later that, in combination with RDF Schema or OWL, this allows us to infer new statements not explicitly present in the RDF Graph

Example:
Example: Course 509 has the students Amy, Mohamed, and John
There is often a need to describe groups of things.

- RDF Defines three predefined types to denote containers:
  - Unordered groups with duplicates (a Bag)
    http://www.w3.org/1999/02/22-rdf-syntax-ns#Bag
  - Ordered groups with duplicates (a Sequence)
    http://www.w3.org/1999/02/22-rdf-syntax-ns#Seq
  - A group of alternatives (an Alternative)
    http://www.w3.org/1999/02/22-rdf-syntax-ns#Alt

- The $n$-th item in a collection is specified by the property
  http://www.w3.org/1999/02/22-rdf-syntax-ns#_n
THE RDF GRAPH: COLLECTIONS

- Containers cannot be “closed”
- To specify “closed” groups RDF introduces **collections**, which are list structures formed by:
  - http://www.w3.org/1999/02/22-rdf-syntax-ns#first
  - http://www.w3.org/1999/02/22-rdf-syntax-ns#rest
  - http://www.w3.org/1999/02/22-rdf-syntax-ns#nil

**Example:** Course 509 has exactly the students Amy, Mohamed, and John
Example: staf 85740 created the edge “item10245 has weight 2.4”
To make statements about statements RDF provides the type

- \texttt{http://www.w3.org/1999/02/22-rdf-syntax-ns#statement}

and the properties

- \texttt{http://www.w3.org/1999/02/22-rdf-syntax-ns#subject}
- \texttt{http://www.w3.org/1999/02/22-rdf-syntax-ns#predicate}
- \texttt{http://www.w3.org/1999/02/22-rdf-syntax-ns#object}
A **triple** is also called a **statement**

To store a (piece of) an RDF graph, it suffices to store its set of triples

This can be done in multiple **serialization formats**
Part III: RDF Serialization Formats
<http://www.example.org/staffid/85740> <http://www.example.org/terms/address> _:addr .
_:addr <http://www.example.org/terms/city> "Bedford" .
_:addr <http://www.example.org/terms/street> "1501 Grant Avenue" .
_:addr <http://www.example.org/terms/state> "Massachusetts" .
_:addr <http://www.example.org/terms/postalcode> "0713" .
NTRIPLES: DEFINITION

NTriples is a simple but verbose serialization format

- Each line represents a single statement containing a subject, predicate, and an object, terminated by a dot
- URIs are enclosed in angle brackets <...>
- Anonymous nodes are represented as _:name. This name is only valid within the NTriples file.
- Literals are enclosed between quotes "...", optionally followed by ^^ and a datatype
Turtle extends the NTriples format with some convenient features:

- the `@prefix` directive allows abbreviation of URIs
- the `a` property abbreviates `http://www.w3.org/1999/02/22-rdf-syntax-ns#type`
- XML Schema datatypes may be abbreviated using the `xsd:` prefix
- The comma symbol can be used to repeat the subject and predicate of triples that only differ in the object
- The semicolon symbol can be used to repeat the subject of triples that only differ in the predicate and object

Example:

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

staff:85740 terms:address _:addr
_:addr terms:city "Bedford"^^xsd:string
_:addr terms:street "1501 Grant Avenue" .
_:addr terms:state "Massachusetts" .
_:addr terms:postalcode "0713" .
staff:85740 a terms:employee .
```
Turtle extends the NTriples format with some convenient features:

- the `@prefix` directive allows abbreviation of URIs
- the `a` property abbreviates `http://www.w3.org/1999/02/22-rdf-syntax-ns#type`
- XML Schema datatypes may be abbreviated using the `xsd:` prefix
- The comma symbol can be used to repeat the subject and predicate of triples that only differ in the object
- The semicolon symbol can be used to repeat the subject of triples that only differ in the predicate and object

Example:

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

staff:85740 terms:address _:addr
   terms:street "Bedford"^^xsd:string ;
   terms:state "Massachusetts" ;
   terms:postalcode "0713" .

_:addr terms:city "1501 Grant Avenue" ;
   terms:street "Bedford" ;
   terms:state "Massachusetts" ;
   terms:postalcode "0713" .

staff:85740 a terms:employee .
```
Turtle extends the NTriples format with some convenient features:

- the `@prefix` directive allows abbreviation of URIs
- the `a` property abbreviates `http://www.w3.org/1999/02/22-rdf-syntax-ns#type`
- XML Schema datatypes may be abbreviated using the `xsd:` prefix
- The comma symbol can be used to repeat the subject and predicate of triples that only differ in the object
- The semicolon symbol can be used to repeat the subject of triples that only differ in the predicate and object

Example:

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

staff:85740 :address _:addr
_:addr :city "Bedford"^^xsd:string ;
:street "1501 Grant Avenue" ;
:state "Massachusetts" ;
:postalcode "0713" .

staff:85740 a :employee .
```
Turtle extends the NTriples format with some convenient features:

- Blank nodes can be described by nesting Turtle statements in `[ ]`
- Collections can be described by resources between parenthesis `(...)`

**Example:**

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

staff:85740 :address _:addr
_:addr :city "Bedford"^^xsd:string
:street "1501 Grant Avenue"
:state "Massachusetts"
:postalcode "0713".

staff:85740 a :employee .
```
Turtle extends the NTriples format with some convenient features:

- Blank nodes can be described by nesting Turtle statements in `[]`
- Collections can be described by resources between parenthesis `(...)`

Example:

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

staff:85740 :address [ :city "Bedford"^^xsd:string ;
                    :street "1501 Grant Avenue" ;
                    :state "Massachusetts" ;
                    :postalcode "0713"] .

staff:85740 a :employee .
```
Turtle extends the NTriples format with some convenient features:

- Blank nodes can be described by nesting Turtle statements in [ ]
- Collections can be described by resources between parenthesis (…)

Example:

```turtle
@prefix courses: <http://ulb.be/courses/> .
@prefix terms: <http://ulb.be/terms/> .
@prefix : <http://ulb.be/students/> .

courses:509 terms:students _:a .
_:a rdf:first :amy .
_:a rdf:rest _:b .
_:b rdf:first :mohamed .
_:b rdf:rest _:c .
_:b rdf:first :john .
_:b rdf:rest rdf:nil .
```
Turtle extends the NTriples format with some convenient features:

- Blank nodes can be described by nesting Turtle statements in [ ]
- Collections can be described by resources between parenthesis (...)

Example:

```turtle
@prefix courses: <http://ulb.be/courses/> .
@prefix terms: <http://ulb.be/terms/> .
@prefix : <http://ulb.be/students/> .

courses:509 terms:students ( :amy :mohamed :john ) .
```
Notation 3 (N3) is an extension of Turtle

- Allows the specification of blank nodes without making up temporary identifiers of the form \_\_:\text{name} for them
- It allows the definition of reasoning rules ...
- ...but those rules are not part of the RDF data model!
RDF/XML is a description of RDF Triples and Graphs in XML

Example:

```xml
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:exterms="http://www.example.org/terms/">
  <rdf:Description rdf:about="http://www.example.org/index.html">
    <exterms:creation-date>August 16, 1999</exterms:creation-date>
  </rdf:Description>
</rdf:RDF>
```
Example of multiple statements:

```xml
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:dc="http://purl.org/dc/elements/1.1/"
    xmlns:exterms="http://www.example.org/terms/">

    <rdf:Description rdf:about="http://www.example.org/index.html">
        <exterms:creation-date>August 16, 1999</exterms:creation-date>
    </rdf:Description>

    <rdf:Description rdf:about="http://www.example.org/index.html">
        <dc:language>en</dc:language>
    </rdf:Description>

</rdf:RDF>
```
Example of multiple statements with abbreviation:

```xml
<?xml version="1.0"?><rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:dc="http://purl.org/dc/elements/1.1/"
    xmlns:exterms="http://www.example.org/terms/">
    <rdf:Description rdf:about="http://www.example.org/index.html">
        <exterms:creation-date>August 16, 1999</exterms:creation-date>
        <dc:language>en</dc:language>
    </rdf:Description>
</rdf:RDF>
```
Example of a statement with a resource as object:

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:dc="http://purl.org/dc/elements/1.1/"
  xmlns:exterms="http://www.example.org/terms/"/>

<rdf:Description rdf:about="http://www.example.org/index.html">
  <exterms:creation-date>August 16, 1999</exterms:creation-date>
  <dc:language>en</dc:language>
  <dc:creator rdf:resource="http://www.example.org/staffid/85740"/>
</rdf:Description>
</rdf:RDF>
```
Example of a blank node:

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:exterms="http://www.example.org/terms/">

  <rdf:Description rdf:about="http://www.example.org/staffid/85740">
    <exterms:addresss rdf:nodeID="abc"/>
  </rdf:Description>

  <rdf:Description rdf:nodeID="abc">
    <exterms:city>Bedford</exterms:city>
    <exterms:street>1501 Grant Avenue</exterms:street>
    <exterms:state>Massachusetts</exterms:state>
    <exterms:postalcode>0173</exterms:postalcode>
  </rdf:Description>

</rdf:RDF>
```
Example of a blank node (alternate syntax):

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:exterms="http://www.example.org/terms/"
>
  <rdf:Description rdf:about="http://www.example.org/staffid/85740">
    <exterms:addresss rdf:parseType="Resource">
      <exterms:city>Bedford</exterms:city>
      <exterms:street>1501 Grant Avenue</exterms:street>
      <exterms:state>Massachusetts</exterms:state>
      <exterms:postalcode>0173</exterms:postalcode>
    </exterms:addresss>
  </rdf:Description>
</rdf:RDF>
```
Example of a typed literal:

```xml
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
       xmlns:exterms="http://www.example.org/terms/">
  <rdf:Description rdf:about="http://www.example.org/index.html">
    <exterms:creation-date rdf:datatype="http://www.w3.org/2001/XMLSchema#date">
      1999-08-16
    </exterms:creation-date>
  </rdf:Description>
</rdf:RDF>
```
Example of a typed literal:

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:exterms="http://www.example.org/terms/">
    <rdf:Description rdf:about="http://www.example.org/index.html">
        <exterms:creation-date rdf:datatype="http://www.w3.org/2001/XMLSchema#date">
            1999-08-16
        </exterms:creation-date>
    </rdf:Description>
</rdf:RDF>
```

Note that URIs that occur in attributes must be written in full!
Example of using entities to abbreviate URIs:

```xml
<?xml version="1.0"?>
<!DOCTYPE rdf:RDF [<!ENTITY xsd "http://www.w3.org/2001/XMLSchema#"]>]
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:exterms="http://www.example.org/terms/"
    xmlns:xs="http://www.w3.org/2001/XMLSchema#">
    <rdf:Description rdf:about="http://www.example.org/index.html">
        <exterms:creation-date rdf:datatype="&xsd;date">
            1999-08-16
        </exterms:creation-date>
    </rdf:Description>
</rdf:RDF>
```
Example of container - first possibility:

```xml
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
       xmlns:s="http://example.org/students/vocab#">
    <rdf:Description rdf:about="http://example.org/courses/6.001">
        <s:students>
            <rdf:Bag>
                <rdf:_1 rdf:resource="http://example.org/students/Amy"/>
                <rdf:_2 rdf:resource="http://example.org/students/Mohamed"/>
                <rdf:_3 rdf:resource="http://example.org/students/Johann"/>
                <rdf:_4 rdf:resource="http://example.org/students/Maria"/>
                <rdf:_5 rdf:resource="http://example.org/students/Phuong"/>
            </rdf:Bag>
        </s:students>
    </rdf:Description>
</rdf:RDF>
```
Example of container - second possibility:

```xml
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
          xmlns:s="http://example.org/students/vocab#">
  <rdf:Description rdf:about="http://example.org/courses/6.001">
    <s:students>
      <rdf:Bag>
        <rdf:li rdf:resource="http://example.org/students/Amy"/>
        <rdf:li rdf:resource="http://example.org/students/Mohamed"/>
        <rdf:li rdf:resource="http://example.org/students/Johann"/>
        <rdf:li rdf:resource="http://example.org/students/Maria"/>
        <rdf:li rdf:resource="http://example.org/students/Phuong"/>
      </rdf:Bag>
    </s:students>
  </rdf:Description>
</rdf:RDF>
```
XML/RDF has many other useful abbreviations

- For collections
- For reification
- ...

See the handouts available on website!
RDF graphs can also be serialized as JSON objects.

- Recent standard, outside scope of this class.
- See http://www.w3.org/TR/json-ld/ if you are interested.
Part IV: RDF Schema
**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]
Recent past – 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]

Current/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 asserts 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 ...
• How do we add some semantics to the statements?
Input

```turtle
@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        terms:price     2.50 .
```

How do we find all products that are digital devices?
WHAT DO YOU MEAN: SEMANTICS?

Input

```prolog
@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 terms:price 2.50 .
```

How do we find all products that are digital devices?

Hmm, digital cameras are digital devices

```prolog
select all x such that
x rdf:type terms:digital-camera .
```
WHAT DO YOU MEAN: SEMANTICS?

Input

```ttl
@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 terms:price 2.50 .
```

How do we find all products that are digital devices?

Hmm, digital cameras are digital devices
... so are netbooks

```
select all x such that
  x rdf:type terms:digital-camera .
OR
  x 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 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!
Explicitly Asserted Knowledge

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

Ontologies

Ontologies provide formal specifications of the classes of objects that inhabit “the world”, the relationships between individual and classes, and their properties.
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
Explicitly Asserted Knowledge

<table>
<thead>
<tr>
<th>prod:cam1</th>
<th>rdf:type</th>
<th>terms:digital-camera</th>
</tr>
</thead>
<tbody>
<tr>
<td>prod:cam1</td>
<td>terms:price</td>
<td>150</td>
</tr>
<tr>
<td>prod:nb1</td>
<td>rdf:type</td>
<td>terms:netbook</td>
</tr>
<tr>
<td>prod:nb1</td>
<td>terms:price</td>
<td>300</td>
</tr>
<tr>
<td>prod:book1</td>
<td>rdf:type</td>
<td>terms:book</td>
</tr>
<tr>
<td>prod:book1</td>
<td>terms:price</td>
<td>2.50</td>
</tr>
</tbody>
</table>

Knowledge About the World (Ontology)

- Every digital camera is a digital device
- Every netbook is a digital device
Explicitly Asserted Knowledge

<table>
<thead>
<tr>
<th>prod:cam1</th>
<th>rdf:type</th>
<th>terms:digital-camera</th>
</tr>
</thead>
<tbody>
<tr>
<td>prod:cam1</td>
<td>terms:price</td>
<td>150</td>
</tr>
<tr>
<td>prod:nb1</td>
<td>rdf:type</td>
<td>terms:netbook</td>
</tr>
<tr>
<td>prod:nb1</td>
<td>terms:price</td>
<td>300</td>
</tr>
<tr>
<td>prod:book1</td>
<td>rdf:type</td>
<td>terms:book</td>
</tr>
<tr>
<td>prod:book1</td>
<td>terms:price</td>
<td>2.50</td>
</tr>
</tbody>
</table>

Knowledge About the World (Ontology)

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

Inference

Enriched Knowledge

<table>
<thead>
<tr>
<th>prod:cam1</th>
<th>rdf:type</th>
<th>terms:digital-camera</th>
</tr>
</thead>
<tbody>
<tr>
<td>prod:cam1</td>
<td>terms:price</td>
<td>150</td>
</tr>
<tr>
<td>prod:nb1</td>
<td>rdf:type</td>
<td>terms:netbook</td>
</tr>
<tr>
<td>prod:nb1</td>
<td>terms:price</td>
<td>300</td>
</tr>
<tr>
<td>prod:book1</td>
<td>rdf:type</td>
<td>terms:book</td>
</tr>
<tr>
<td>prod:book1</td>
<td>terms:price</td>
<td>2.50</td>
</tr>
<tr>
<td>prod:cam1</td>
<td>rdf:type</td>
<td>terms:digital-device</td>
</tr>
<tr>
<td>prod:nb1</td>
<td>rdf:type</td>
<td>terms:digital-device</td>
</tr>
</tbody>
</table>
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

Query

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:netbook rdfs:subClassOf terms:digital-device .
terms:book rdfs:subClassOf terms:media .
terms:electronic-device rdfs:subClassOf terms:device .
Example Ontology in RDF Schema:

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

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

Starting from the following asserted triples ...

```rdfs
@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 terms:price 2.50 .
```
Example Ontology in RDF Schema:

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

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 terms:price 2.50 .
prod:cam1 rdf:type terms:digital-device .
prod:nb1 rdf:type terms:digital-device .
prod:book1 rdf:type terms:media .
```
Example Ontology in RDF Schema:

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

- The prefix **rdf** abbreviates the RDF namespace
  \url{http://www.w3.org/1999/02/22-rdf-syntax-ns#}
- The prefix **rdfs** abbreviates the RDFS namespace
  \url{http://www.w3.org/2000/01/rdf-schema#}
- We range over arbitrary URIs by \(a\) and \(b\) (i.e., anything admissible for the predicate position of a triple)
- \(u\) and \(v\) refer to arbitrary URIs or blank node IDs by (i.e., anything admissible for the subject position of a triple)
- \(x\) and \(y\) can be used for arbitrary URIs, blank node IDs or literals
Classes and class hierarchies

- Classes stand for sets of things (in RDF: sets of URIs)
- Use `rdf:type` to indicate class membership
- A URI can belong to several classes.

Example

```
:mary  rdf:type  :Woman .
:mary  rdf:type  :Student .
```
Classes stand for sets of things (in RDF: sets of URIs)

Use `rdf:type` to indicate class membership

A URI can belong to several classes.

\( X \text{ rdfs:subClassOf } Y \) is used to specify that all members of a class \( X \) are also members of a class \( Y \)

This allows to arrange classes in hierarchies.

And allows expressing that two classes are the same by mutual inclusion: \( C_1 \text{ subClassOf } C_2 \) and \( C_2 \text{ subClassOf } C_1 \).

**Deduction Rule**

<table>
<thead>
<tr>
<th>If</th>
<th>Then infer</th>
</tr>
</thead>
<tbody>
<tr>
<td>( u \text{ rdfs:subClassOf } x )</td>
<td>( y \text{ rdf:type } x )</td>
</tr>
<tr>
<td>And ( y \text{ rdf:type } u )</td>
<td>( y \text{ rdf:type } x )</td>
</tr>
</tbody>
</table>

**Example**

<table>
<thead>
<tr>
<th>Example Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>:mary</td>
</tr>
<tr>
<td>:mary</td>
</tr>
<tr>
<td>:Woman</td>
</tr>
<tr>
<td>:Person</td>
</tr>
</tbody>
</table>
Classes stand for sets of things (in RDF: sets of URIs)

- Use `rdf:type` to indicate class membership
- A URI can belong to several classes.
- `X rdfs:subClassOf Y` is used to specify that all members of a class `X` are also members of a class `Y`
- This allows to arrange classes in hierarchies.
- And allows expressing that two classes are the same by mutual inclusion: `C_1 subClassOf C_2` and `C_2 subClassOf C_1`

**Deduction Rule**

If \( u \ rdfs:subClassOf x \).
And \( y \ rdf:type u \).
Then infer \( y \ rdf:type x \).

**Example**

- `:mary rdf:type :Woman`.
- `:mary rdf:type :Student`.
- `:Woman rdfs:subClassOf :Person`.
- `:Person rdfs:subClassOf :Animal`.
- `:mary rdf:type :Woman`.
• Classes stand for sets of things (in RDF: sets of URIs)
• Use `rdf:type` to indicate class membership
• A URI can belong to several classes.
• \( X \text{ rdfs:subClassOf } Y \) is used to specify that all members of a class \( X \) are also members of a class \( Y \)
• This allows to arrange classes in hierarchies.
• And allows expressing that two classes are the same by mutual inclusion: \( C_1 \text{ subClassOf } C_2 \) and \( C_2 \text{ subClassOf } C_1 \)

**Deduction Rule**

If \( u \text{ rdfs:subClassOf } x \).
And \( y \text{ rdf:type } u \).
Then infer \( y \text{ rdf:type } x \).

**Example**

```
:mary rdf:type :Woman .
:mary rdf:type :Student .
:Woman rdfs:subClassOf :Person .
:Person rdfs:subClassOf :Animal .
:mary rdf:type :Woman .
```
Everything that occurs in the middle of a triple is a property.

$\text{X rdfs:subPropertyOf Y}$ is used to specify that all resources with property X to some object also have property Y to that object.

This allows to arrange properties also in hierarchies.

And expressing that two properties are the same (by mutual inclusion).

**Deduction Rule**

If $a \text{ rdfs:subPropertyOf } b$.

And $x a y$.

Then add $x b y$.

**Example**

```
:john :happilyMarriedTo :mary .
:happilyMarriedTo rdfs:subPropertyOf :marriedTo .
```
Properties and Property Hierarchies

- Everything that occurs in the middle of a triple is a property.
- \( X \texttt{rdfs:subPropertyOf} Y \) is used to specify that all resources with property \( X \) to some object also have property \( Y \) to that object.
- This allows to arrange properties also in hierarchies.
- And expressing that two properties are the same (by mutual inclusion).

**Deduction Rule**

If \( a \texttt{rdfs:subPropertyOf} b \).
And \( x a y \).
Then add \( x b y \).

**Example**

\[
\text{:john} \quad \text{:happilyMarriedTo} \quad \text{:mary}.
\text{:happilyMarriedTo} \texttt{rdfs:subPropertyOf} \text{:marriedTo}.
\text{:john} \quad \text{:marriedTo} \quad \text{:mary}.
\]
Property restrictions allow expressing that a certain property can only be between things of a certain rdf:type.

- P rdfs:domain C is used to specify that all subjects with (outgoing) property P belong to class C.
- P rdfs:range C is used to specify that all objects with (incoming) property P belong to class C.

**Deduction Rule**

If \( a \) rdfs:domain \( x \).
And \( u \) a y.
Then add \( u \) rdf:type \( x \).

If \( a \) rdfs:range \( x \).
And \( u \) a v.
Then add \( v \) rdf:type \( x \).

**Example**

\[ :\text{john} :\text{isMarriedTo} :\text{mary} . \]
\[ :\text{isMarriedTo} \text{ rdfs:domain } :\text{Person} . \]
\[ :\text{isMarriedTo} \text{ rdfs:range } :\text{Person} . \]
• **Property restrictions** allow expressing that a certain property can only be between things of a certain rdf:type.

• **P rdfs:domain C** is used to specify that all subjects with (outgoing) property $P$ belong to class $C$

• **P rdfs:range C** is used to specify that all objects with (incoming) property $P$ belong to class $C$

---

**Deduction Rule**

If $a$ rdfs:domain $x$.
And $u$ a $y$.
Then add $u$ rdf:type $x$.

If $a$ rdfs:range $x$.
And $u$ a $v$.
Then add $v$ rdf:type $x$.

**Example**

:john :isMarriedTo :mary .
:isMarriedTo rdfs:domain :Person .
:isMarriedTo rdfs:range :Person .
:john rdf:type :Person .
PROPERTY RESTRICTION

- **Property restrictions** allow expressing that a certain property can only be between things of a certain rdf:type.
- **P rdfs:domain C** is used to specify that all subjects with (outgoing) property $P$ belong to class $C$
- **P rdfs:range C** is used to specify that all objects with (incoming) property $P$ belong to class $C$

**Deduction Rule**

If $a$ rdfs:domain $x$.
And $u$ a $y$.
Then add $u$ rdf:type $x$.

If $a$ rdfs:range $x$.
And $u$ a $v$.
Then add $v$ rdf:type $x$.

**Example**

<table>
<thead>
<tr>
<th>:john</th>
<th>:isMarriedTo</th>
<th>:mary</th>
</tr>
</thead>
<tbody>
<tr>
<td>:isMarriedTo</td>
<td>rdfs:domain</td>
<td>:Person</td>
</tr>
<tr>
<td>:isMarriedTo</td>
<td>rdfs:range</td>
<td>:Person</td>
</tr>
<tr>
<td>:john</td>
<td>rdf:type</td>
<td>:Person</td>
</tr>
<tr>
<td>:mary</td>
<td>rdf:type</td>
<td>:Person</td>
</tr>
</tbody>
</table>
Consider the following example. What new triples are inferred?

```reasoning
:MarriedWoman rdfs:subClassOf :Woman .
:maidenName rdfs:domain :MarriedWoman .
:Karen :maidenName "Stephens" .
```
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 .
```
Beware of the following when modeling using RDFs:
• multiple property restrictions are allowed.

Example

ex:authorOf rdfs:range ex:Storybook .

States that everything in the range of ex:authorOf is both a textbook and a storybook!
Beware of the following when modeling using RDFs:

- multiple property restrictions are allowed.
- property restrictions do not depend on the context

Example

```reasonml
ex:isMarriedTo rdfs:domain ex:Person .
ex:isMarriedTo rdfs:range ex:Person .
ex:instULB rdf:type ex:Institution .
ex:john ex:isMarriedTo ex:instULB .
```
MODELING PITFALLS

Beware of the following when modeling using RDFs:

- multiple property restrictions are allowed.
- property restrictions do not depend on the context

**Example**

```
ex:isMarriedTo rdfs:domain ex:Person .
ex:isMarriedTo rdfs:range ex:Person .
ex:instULB rdf:type ex:Institution .
ex:john ex:isMarriedTo ex:instULB .
ex:instULB rdf:type :Person .
```

Now we have ex:instULB is both an institute and a Person.
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)

<table>
<thead>
<tr>
<th>triple</th>
<th>triple</th>
<th>triple</th>
</tr>
</thead>
<tbody>
<tr>
<td>rdfs:subClassOf</td>
<td>rdfs:domain</td>
<td>rdfs:Class</td>
</tr>
<tr>
<td>rdfs:subClassOf</td>
<td>rdfs:range</td>
<td>rdfs:Class</td>
</tr>
<tr>
<td>rdfs:subClassOf</td>
<td>rdf:type</td>
<td>rdfs:Class</td>
</tr>
<tr>
<td>rdfs:subPropertyOf</td>
<td>rdfs:domain</td>
<td>rdfs:Property</td>
</tr>
<tr>
<td>rdfs:subPropertyOf</td>
<td>rdfs:range</td>
<td>rdfs:Property</td>
</tr>
<tr>
<td>rdfs:subPropertyOf</td>
<td>rdf:type</td>
<td>rdfs:Property</td>
</tr>
<tr>
<td>rdfs:domain</td>
<td>rdf:type</td>
<td>rdfs:Property</td>
</tr>
<tr>
<td>rdfs:range</td>
<td>rdf:type</td>
<td>rdfs:Property</td>
</tr>
<tr>
<td>rdfs:Resource</td>
<td>rdf:type</td>
<td>rdfs:Class</td>
</tr>
<tr>
<td>rdfs:Class</td>
<td>rdfs:subClassOf</td>
<td>rdfs:Resource</td>
</tr>
<tr>
<td>rdfs:Class</td>
<td>rdf:type</td>
<td>rdfs:Class</td>
</tr>
</tbody>
</table>
For modeling open lists, RDF schema introduces:

- **rdfs:Container** as the superclass of **rdf:Seq**, **rdf:Bag**, **rdf:Alt**
- **rdfs:ContainerMembershipProperty** as the property that contains all properties used with containers (**rdf:_1**, **rdf:_2**, ...are all of this type).
In addition, it introduces

- a new property, `rdfs:member`
- which is the superproperty of all properties contained in `rdfs:ContainerMembershipProperty`
- this allows one to introduce their own properties to describe membership in a container.
- and to easily determine whether something is a member of a container without knowing the property used (just look for `u rdfs:member x`).

If

\[ a \text{ rdf:type rdfs:ContainerMembershipProperty} . \]
And

\[ u a x . \]
Then add

\[ a \text{ rdfs:member } x . \]
In addition, RDFs proposes a standard set of URIs to document RDF or RDFs. These are not part of the actual facts or ontology, but are for the human reader/user/developer.

- **rdfs:label** to give, e.g., a human-readable name for a URI;
- **rdfs:comment** used for lengthy commentary/explanatory text;
- **rdfs:seeAlso** and **rdfs:definedBy**: properties pointing to URIs where further information or definitions can be found.
A SIMPLE ONTOLOGY

```
ex:vegetableThaiCurry ex:thaiDishBasedOn ex:coconutMilk .
ex:sebastian rdf:type ex:AllergicToNuts .
ex:sebastian ex:eats ex:vegetableThaiCurry .
ex:AllergicToNuts rdfs:subClassOf ex:Pitiable .
ex:thaiDishBasedOn rdfs:domain ex:Thai .
ex:thaiDishBasedOn rdfs:range ex:Nutty .
ex:thaiDishBasedOn rdfs:subPropertyOf ex:hasIngredient .
ex:hasIngredient rdf:type rdfs:containerMembershipProperty .
```
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 will remedy this.

- RDF 1.1. Primer ([http://www.w3.org/TR/rdf11-primer/](http://www.w3.org/TR/rdf11-primer/))
- RDF 1.1. Turtle ([http://www.w3.org/TR/turtle/](http://www.w3.org/TR/turtle/)), section 1, 2, 3.
- RDF 1.1. N-Triples ([http://www.w3.org/TR/n-triples/](http://www.w3.org/TR/n-triples/)), section 1, 2