

Formation doctorale Knowledge processing

Yannis Haralambous & Gábor Bella (IMT Atlantique)

May 6-7, 2026

<https://tinyurl.com/bddk74vp>

A CRASH COURSE IN KNOWLEDGE PROCESSING & THE SEMANTIC WEB!

DATA ≠ INFORMATION ≠ KNOWLEDGE!



I PREDICT THERE WILL BE MORE DATA!

WE HAVE DATA!
BUT DO WE KNOW WHAT IT MEANS?



0110101
110010

42



CHART

LOGIC LABYRINTH

$P \rightarrow Q$
 $\neg P \times Q$
 $(P \wedge Q) \rightarrow R$

$\forall x P(x)$
 $\exists x Q(x)$

$P \rightarrow Q$
 $\neg P \times Q$
 $(P \wedge Q) \rightarrow R$

COMPLETENESS THEOREM
BRIDGE: $\models \phi$ iff $\models \phi$

PROOF VS POSSIBLE WORLDS

PROOF FOREST (⊢)

POSSIBLE WORLDS SWAMP (⊨)

COMPLETENESS THEOREM
BRIDGE: $\models \phi$ iff $\models \phi$

KB $\models \alpha$??

I ONLY TRUST MODELS.

ALL MODELS

GÖDEL IS WATCHING...

THIS STATEMENT IS NOT PROVABLE.

NO SYSTEM CAN PROVE EVERYTHING!

OPEN WORLD / ASSUMPTION PIT

BOB HAS 2 CHILDREN

WHAT IS TRUE?!

YES! THEREFORE $KB \models \alpha$

THE SEMANTIC WEB CAKE (BAKED SINCE 2000)

TRUST & APPLICATIONS
TAKE IT TO THE USERS!

PROOF
TRUST, BUT VERIFY!

UNIFYING LOGIC
ALL THE WAY UP!

ONTOLOGIES
MAKING SENSE OF THE WORLD

SPARQL
ASK. SELECT. CONSTRUCT. CONVINCE.
SELECT ?x WHERE { ?x foaf:knows ?y }

RULES (SWRL / RIF)
IF ... THEN ...
hasParent(x,y) $\wedge$ hasBrother(y,z) $\Rightarrow$ hasUncle(x,z)

OWL / DESCRIPTION LOGIC
VERY EXPRESSIVE (BUT DECIDABLE!)

RDFS / TAXONOMIES
CLASSES & HIERARCHIES
Person $\sqsubseteq$ Agent Dog $\sqsubseteq$ Mammal

RDF
EVERYTHING IS A TRIPLE!
Gérald — loves — Alice

XML
TREES ALL THE WAY DOWN!
<tag> <more> <and, more>...

UNICODE / IRI
EVERY CHARACTER HAS AN IDENTITY!
A A A
SAME GLYPH, DIFFERENT CHARACTERS!

IT'S FRAGILE!

18 WHEELS... ARE THEY DISTINCT THOUGH??

JUST KIDDING, THERE'S ALWAYS MORE...

EVERYTHING LINKS TO SOMETHING!



DUBLIN CORE LIBRARIAN

TITLE ✓
CREATOR ✓
DATE ✓
SUBJECT ✓
FORMAT ✓
IDENTIFIER ✓

EVERYTHING NEEDS METADATA!

EVEN SANDWICHES!

SKOS LABEL CHAOS

Dog (en)
Chien (fr)
Hund (de)
Canine (en)

SAME CONCEPT? DEPENDS!



Data $\neq$ Information $\neq$ Knowledge

- We all use terms “data,” “information,” “knowledge” in daily life.
- But they all have a specific meaning used in AI:
- *data* are ordered sets of numbers, originating from sensors, simulation programs or random data generators, etc.;
- *information* is data to which we have attached meaning: we know what represents each number or sequence of bytes or character string;
- *knowledge* is motives or tendencies extracted from information in order to do predictions.
- Example: a train leaving from Brest railway station.

Pyramids, the Brest Railway Station...

DIKW

Wisdom

Knowledge

Information

Data

DIK

Knowledge

Information

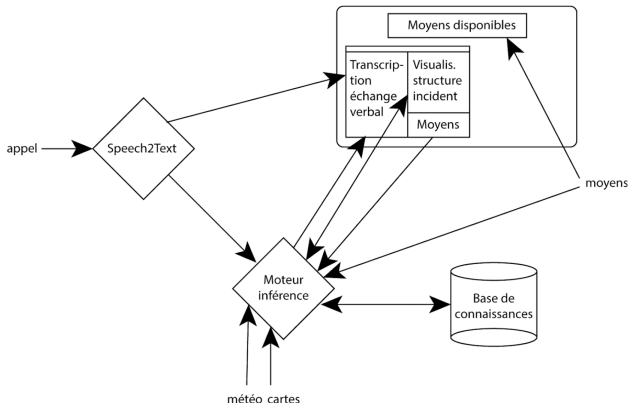
Data



Our Working Tools

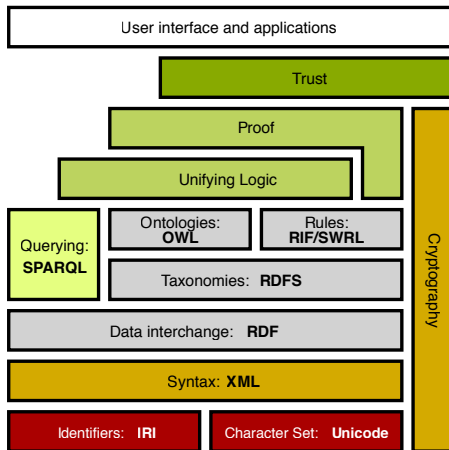
- Human intelligence operates through *reasoning processes* based on *(internal) knowledge representations*.
- In AI we create *knowledge bases*.
- They contain *statements* expressed in *knowledge representation languages*, like, e.g.,
 - different kinds of logic: propositional, first-order, second-order, description, modal, fuzzy, temporal, nonmonotonic, etc., but also
 - ontologies,
 - conceptual graphs,
 - some controlled languages, etc.
- We can interrogate a knowledge base: to obtain the answer to the query we also use the inference process.

A Real-World Example: The VAIATA Project



1. Nature of the incident 2. Identification of person and vessel 3. Location
4. Vessel AIS 5. Casualties (injuries, medical issues, etc.) 6. Material damage (hull, engine, sail, etc.) 7. Weather 8. Risk of pollution 9. Danger to other vessels 10. Level of urgency (Mayday, Pan-Pan, etc.) 11. Resources deployed / Outcome of the incident

The Semantic Web Cake (2000)



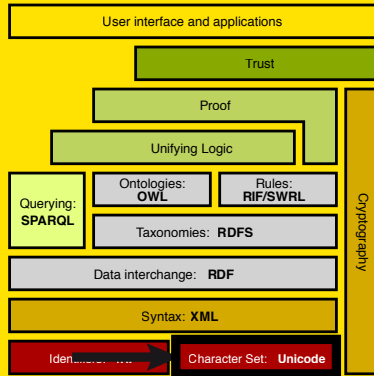
Tim Berners-Lee

Take a look at the very informative
<https://www.w3.org/2000/Talks/1206-xml2k-tbl>

Open World Assumption

- UML uses the *Closed-World Assumption*: if you say that “Bob is the father of Bill,” and “Bob is the father of John,” and there is no other relation of the kind, then Bob has exactly two children.
- In an *Open-World Assumption*, Bob would have between 1 and ∞ children.
- Natural language uses *implicatures* as hidden information: when I say that a book has 5 chapters, I (normally) mean “exactly five”.
- In the following we will admit the Open-World Assumption.
- Read this: S. Russell, *Unifying Logic and Probability* (on Moodle).

Part



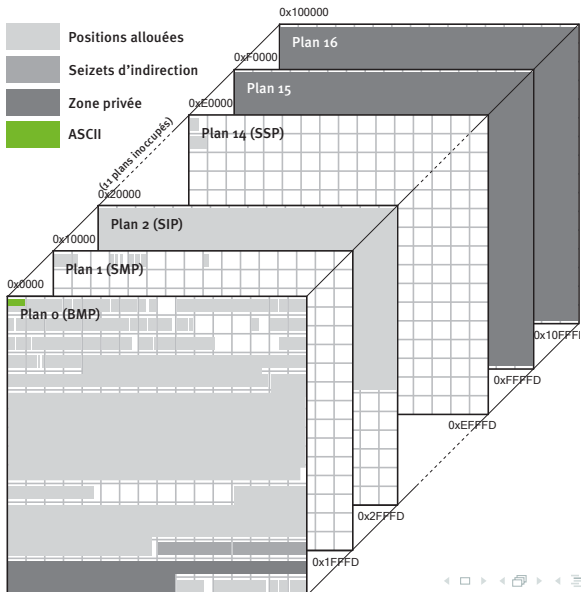
Unicode

- Unicode is (1) a *character set* and (2) an encoding, that is a set of *code points* for the members of the character set.
- A *character* is an atomic unit of written text or of a notation system.
- A *code point* is a number between 0 and $2^{23} - 1$.
- A *Unicode character* has a *name*, a *code point*, and *properties*. Most characters are visually represented by *glyphs*.
- E.g., characters GREEK CAPITAL LETTER ALPHA, CYRILLIC CAPITAL LETTER A and LATIN CAPITAL LETTER A have code points 0x0391, 0x0410 and 0x0041, they all share glyph “A”.
- The “semantics” of a character depend on the writing/notation system it belongs to.

Unicode Structure

- The first Unicode version came out in 1991, today we reached v. 13 (March 2020).
- Unicode has 1 114 111 *slots* ($0 \times 00 - 0 \times 10 \text{FFFF}$);
- it is divided in 17 *planes*, numbered 0 to 16;
- plane 0 («BMP») is the «basic multilingual plane»;
- plane 1 («SMP») is the «supplementary multilingual plane»;
- plane 2 («SIP») is the «supplementary ideographic plane»;
- planes 2–13 are empty (for the moment);
- plane 14 («SSP») is the «supplementary special-purpose plane»;
- planes 15 and 16 are private use planes.

Unicode Structure



Reading Character Lists

00CA

C1 Controls and Latin-1 Supplement

00ED

00CA Ê LATIN CAPITAL LETTER E WITH CIRCUMFLEX

≡ 0045 E 0302 ◌̂

00CB Ë LATIN CAPITAL LETTER E WITH DIAERESIS

≡ 0045 E 0308 ◌̈

00CC Ì LATIN CAPITAL LETTER I WITH GRAVE

≡ 0049 I 0300 ◌̀

00CD Í LATIN CAPITAL LETTER I WITH ACUTE

≡ 0049 I 0301 ◌́

00CE Î LATIN CAPITAL LETTER I WITH CIRCUMFLEX

≡ 0049 I 0302 ◌̂

00CF Ï LATIN CAPITAL LETTER I WITH DIAERESIS

≡ 0049 I 0308 ◌̈

00DD Ý LATIN CAPITAL LETTER Y WITH ACUTE

≡ 0059 Y 0301 ◌́

00DE Þ LATIN CAPITAL LETTER THORN

00DF ß LATIN SMALL LETTER SHARP S
= Eszett

- German

- uppercase is “SS”

- in origin a ligature of 017F f and 0073 s
→ 03B2 β greek small letter beta

00E0 à LATIN SMALL LETTER A WITH GRAVE

≡ 0061 a 0300 ◌̀

00E1 á LATIN SMALL LETTER A WITH ACUTE

≡ 0061 a 0301 ◌́

00E2 â LATIN SMALL LETTER A WITH CIRCUMFLEX

≡ 0061 a 0302 ◌̂

Reading Character Lists

00AE

C1 Controls and Latin-1 Supplement

00C9

00AE ® REGISTERED SIGN
= REGISTERED TRADE MARK SIGN

00AF ¯ MACRON
= overline, APL overbar
• this is a spacing character
→ 02C9 ¯ modifier letter macron
→ 0304 ¯ combining macron
→ 0305 ¯ combining overline
≈ 0020 [sp] 0304 ¯

00B0 ° DEGREE SIGN
• this is a spacing character
→ 02DA ° ring above
→ 030A ª combining ring above
→ 2070 ⁰ superscript zero
→ 2218 ∘ ring operator

00B1 ± PLUS-MINUS SIGN
→ 2213 ∓ minus-or-plus sign

00B9 ¹ SUPERSCRRIPT ONE
→ 00B2 ² superscript two
→ 00B3 ³ superscript three
≈ <super> 0031 1

00BA º MASCULINE ORDINAL INDICATOR
• Spanish
≈ <super> 006F o

00BB » RIGHT-POINTING DOUBLE ANGLE
QUOTATION MARK
= RIGHT POINTING GUILLEMET
• usually closing, sometimes opening
→ 226B » much greater-than
→ 300B » right double angle bracket

00BC ¼ VULGAR FRACTION ONE QUARTER
• bar may be horizontal or slanted
• other fraction characters: 2153 ⅓ –215E ⅔
≈ 0031 1 2044 / 0034 4

Information Strata

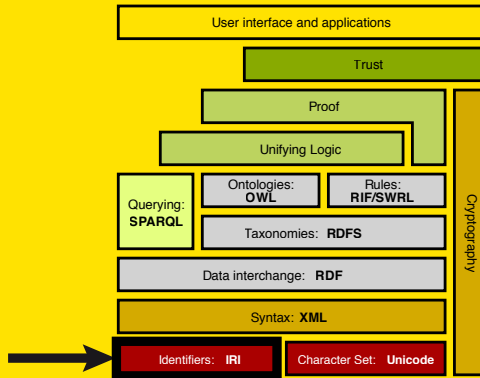
- Layer 1: *Abstract Character Repertoire*: descriptions;
- Layer 2: *Coded Character Set*: descriptions + slots;
- Layer 3: *Character Encoding Form*: computer memory representation (UTF-8, UTF-16, UTF-32, etc.);
- Layer 4: *Character Encoding Scheme*: low-level computer memory representation (UTF-16BE, UTF-16LE, etc.);
- Layer 5: *Transfer Encoding Syntax* (hexa, base 64, etc.).

UTF-8 Character Encoding Form

- **UTF-8** is a *form* of the Unicode encoding, invented by Ken Thompson (the inventor of C and Unix) and Rob Pike.
- Instead of coding all chars in 32 bits, we combine the nonzero bits to form a variable-length character code of 1 to 4 8-bit bytes (the “8” in UTF-8). ASCII characters are coded by single bytes and texts entirely written in ASCII do not change.

UTF-32	UTF-8			
	Byte 1	Byte 2	Byte 3	Byte 4
00000000 00000000 0xxxxxxx	0xxxxxxx			
00000000 00000yyy yyxxxxxx	110yyyyy	10xxxxxx		
00000000 zzzzyyyy yyxxxxxx	1110zzzz	10yyyyyy	10xxxxxx	
000uuuzz zzzzyyyy yyxxxxxx	11110uuu	10zzzzzz	10yyyyyy	10xxxxxx

Part



IRI (2005)

- An IRI *Internationalized Resource Identifier* is a Unicode-enabled *Uniform Resource Identifier*.
- An IRI has the following syntax:

```
scheme:[//[userinfo@]host[:port]]  
path[?query][#fragment]
```

E.g.,

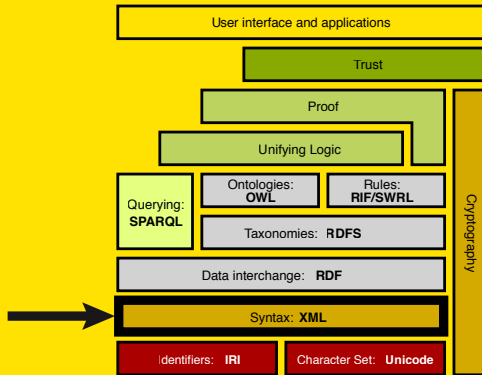
<http://www.iriofficial.com>

[https://en.wikipedia.org/wiki/
Internationalized_Resource_Identifier](https://en.wikipedia.org/wiki/Internationalized_Resource_Identifier)

- IRIs denote resources in an unambiguous way.
- The “semantics” of an IRI is the resource denoted.



Part



XML (1998)

- XML is a *markup language*.
- An *XML document* is a string of Unicode characters and IRIs with a tree structure.
- The nodes of this tree structure can be: *elements*, *namespaces*, *attributes*, *text content*, *processing instructions*. Edges denote the inclusion relation.
- E.g.,

```
<?xml version="1.1" encoding="utf-8"?>
<a xmlns:me="http://www.somewhere.org/somepath">
  <me:b id="32">
```

Some contents.....

```
    </me:b>
    <?php print("Hello world");?>
  </a>
```

- The “semantics” of an XML node are given by its namespace.

Compositionality in XML

```
<?xml version="1.1" encoding="utf-8"?>
<xhtml:body xmlns:xhtml="http://www.w3.org/1999/xhtml"
  xmlns:me="http://www.somewhere.org/somepath">
  <xhtml:h1>
```

Some Important Towns

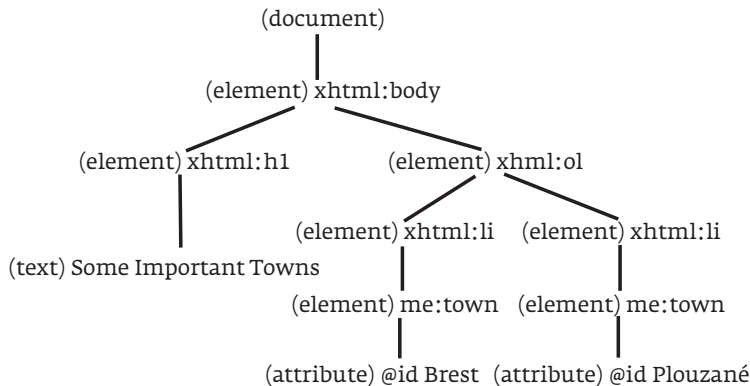
```
</xhtml:h1>
<xhtml:ol>

  <xhtml:li><me:town id="Brest"/></xhtml:li>
  <xhtml:li><me:town id="Plouzané"/></xhtml:li>

</xhtml:ol>
</xhtml:body>
```

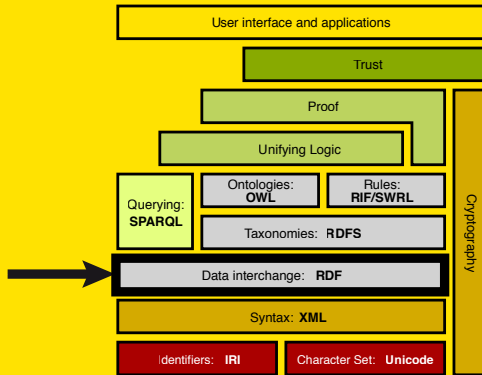
Compositionality in XML

- Just like natural language, an XML document has syntax.
- Applying the compositionality principle to this tree:



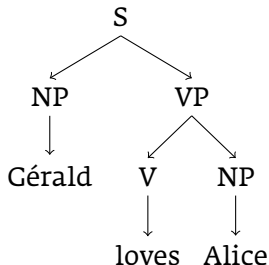
- ... we get the “semantics” of the XML document.

Part



RDF (1999)

- RDF is a method for making statements about resources.
- An RDF document is a set of *triples* of the kind:
(*subject*, *predicate*, *object*).
- In natural language syntax:



- In *First-Order Logic*, we would write the formula “loves(Gérald,Alice),” where “loves” is a predicate.
- For the corresponding RDF triple, see next slide.

RDF (1999)

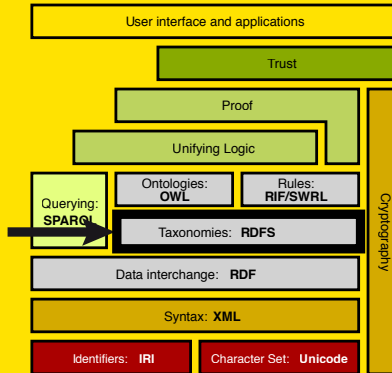
- In RDF, atomic items are IRIs or Unicode literal strings, and the syntax used to represent RDF triples can be XML or Turtle:

```
@prefix me: <http://www.myself.me/something> .  
@prefix foaf: <http://xmlns.com/foaf/0.1/> .  
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .  
me:Gérald me:loves me:Alice .  
me:Gérald rdf:type foaf:Person .  
me:Alice rdf:type foaf:Person .  
me:loves rdf:type rdf:Property .
```

- Period . for end of triple, Semicolon ; for multiple predicate+objects, Comma , for multiple objects.

```
me:Gérald me:loves me:Alice ;  
  rdf:type foaf:Person .  
me:Gérald me:loves me:Alice , me:Béatrice , me:Claudie .
```

Part



RDF Schema (1998)

- RDF Schema (RDFS) allows, among other things, the definition of classes and the relation of subclass, ranges and domains of properties:

```
@prefix me: <http://www.myself.me/something> .  
@prefix foaf: <http://xmlns.com/foaf/0.1/>  
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .  
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .  
me:Alice rdf:type me:LovelyPerson .  
me:LovelyPerson rdf:type rdfs:Class .  
me:LovelyPerson rdfs:subClassOf foaf:Person .  
me:loves rdfs:domain foaf:Person .  
me:loves rdfs:range me:LovelyPerson .
```

RDF Schema (1998)

- Some useful RDFS predicates:

```
me:Alice rdfs:label "Alice" .  
me:Alice rdfs:label "Alice"^^xsd:string .  
me:Alice rdfs:label "Alice"@fr .  
me:Alice rdfs:comment "this is bullshit!"@en .
```

- Empty nodes:

```
me:Alice me:loves [] .  
me:Alice me:loves [ foaf:name "G rald" ;  
  rdf:type foaf:Person ] .  
[ foaf:name "Alice" ; rdf:type foaf:Person ]  
  me:loves me:G rald .
```

RDF Schema (1998)

- A list structure:

`me:playlist me:tracks ( me:song1 me:song2 me:song3 ) .`

which actually means

```
:playlist :tracks [  
  rdf:first :song1 ;  
  rdf:rest [  
    rdf:first :song2 ;  
    rdf:rest [  
      rdf:first :song3 ;  
      rdf:rest rdf:nil  
    ]  
  ]  
] .
```

Reification in RDF (1998)

- In RDF one can use *reification*:

```
@prefix me: <http://www.myself.me/something> .  
@prefix foaf: <http://xmlns.com/foaf/0.1/>  
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .  
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .  
me:G-l-A rdf:type rdf:Statement .  
me:G-l-A rdf:subject me:Gérald .  
me:G-l-A rdf:predicate me:loves .  
me:G-l-A rdf:object me:Alice .  
me:G-l-A rdf:type me:Lie .
```

XSD—Overview

XML Schema Datatypes provide the datatype system reused by RDF for typed literals.

- W3C Recommendation (XSD 1.1, 2012).
- Namespace: <http://www.w3.org/2001/XMLSchema#>, prefix `xsd:`.
- In RDF, datatypes are attached to literals with the `^^` operator: `"42"^^xsd:integer`.
- Without an explicit datatype, plain literals are `xsd:string` (or `rdf:langString` when tagged).

XSD is what lets a triple store or reasoner know that

`"2024-03-15"^^xsd:date` and `"15/3/2024"^^xsd:string` are not the same thing.

XSD—Main Datatypes

Numeric

- `xsd:integer`, `xsd:decimal`
- `xsd:double`, `xsd:float`
- `xsd:int`, `xsd:long`,
`xsd:byte`
- `xsd:nonNegativeInteger`,
`xsd:positiveInteger`

Date / time

- `xsd:dateTime`, `xsd:date`,
`xsd:time`
- `xsd:duration`
- `xsd:gYear`, `xsd:gYearMonth`

String

- `xsd:string` (default)
- `xsd:normalizedString`,
`xsd:token`
- `xsd:language`

Other

- `xsd:boolean`
- `xsd:anyURI`
- `xsd:base64Binary`,
`xsd:hexBinary`

XSD—Facets and Examples

Custom datatypes are derived from base types by applying *facets*:
`minInclusive`, `maxInclusive`, `pattern`, `length`, `enumeration`, ...

@prefix ex: <http://example.org/> .

@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

```
ex:alice ex:age "30"^^xsd:integer ;
        ex:height "1.72"^^xsd:decimal ;
        ex:joinedOn "2024-03-15"^^xsd:date ;
        ex:active "true"^^xsd:boolean ;
        ex:website "https://alice.example"^^xsd:anyURI .
```

SPARQL filter expressions and OWL data-property restrictions both rely on XSD datatypes for ordering, comparison and validation.

FOAF—Overview

Friend of a Friend: a vocabulary for describing people, their activities, and the relations between them and to other entities.

- Started in 2000 by Dan Brickley and Libby Miller; one of the oldest and most reused Linked Data vocabularies.
- Namespace: <http://xmlns.com/foaf/0.1/>, prefix **foaf:**.
- Designed to be lightweight and openly extensible—ideal for personal profiles, author metadata, and social graphs.

FOAF predates schema.org's **Person** type by more than a decade and remains common in academic and Linked Data contexts.

FOAF—Main Classes

- `foaf:Agent`—the top class: anything that can act (people, organizations, groups, software).
- `foaf:Person` $\sqsubseteq$ `foaf:Agent` —a human being.
- `foaf:Organization` $\sqsubseteq$ `foaf:Agent` —companies, institutions.
- `foaf:Group` $\sqsubseteq$ `foaf:Agent` —a collection of agents.
- `foaf:Document`, `foaf:Image` —web resources and depictions.
- `foaf:OnlineAccount`—a user account on a service.
- `foaf:Project`—a collaborative undertaking.

FOAF—Main Properties and Example

Identification

name, givenName, familyName, nick, mbox, mbox_sha1sum

Web presence

homepage, weblog, account, depiction, img

Social

knows, member, maker

Topical

interest, topic_interest, currentProject

@prefix foaf: <http://xmlns.com/foaf/0.1/> .

@prefix ex: <http://example.org/> .

```
ex:alice a foaf:Person ;
  foaf:name  "Alice Martin" ;
  foaf:mbox  <mailto:alice@ex.org> ;
  foaf:homepage  <https://alice.ex/> ;
  foaf:knows  ex:bob , ex:carol .
```

Dublin Core—Overview

Dublin Core Metadata Initiative (DCMI): a small, general-purpose metadata vocabulary for describing any resource.

- Originated in 1995 at a workshop in Dublin, Ohio; the foundational metadata standard for digital libraries.
- Two namespaces:
 - **dc:**—legacy 15 elements at <http://purl.org/dc/elements/1.1/>
 - **dcterms:**—the modern, richer DCMI Terms at <http://purl.org/dc/terms/>
- Used in repositories, OAI-PMH, RSS, embedded HTML metadata.

Modern usage prefers **dcterms:** because it has formal domains, ranges, and refinements.

Dublin Core—The 15 Core Elements

Content

- title
- subject
- description
- type
- source
- relation
- coverage

Responsibility

- creator
- contributor
- publisher
- rights

Instantiation

- date
- format
- identifier
- language

DCMI Terms refines these and adds many more: `created`, `modified`, `issued`, `abstract`, `license`, `isPartOf`, `hasPart`, `references`, `conformsTo`, `spatial`, `temporal`, ...

It also adds classes: `Agent`, `BibliographicResource`, `FileFormat`, `Location`, `LinguisticSystem`.

Dublin Core—Example

```
@prefix dcterms: <http://purl.org/dc/terms/> .
@prefix foaf:   <http://xmlns.com/foaf/0.1/> .
@prefix ex:    <http://example.org/> .
```

```
ex:paper42 a dcterms:BibliographicResource ;
  dcterms:title    "On Knowledge Graphs"@en ;
  dcterms:creator  ex:alice , ex:bob ;
  dcterms:publisher ex:springer ;
  dcterms:issued   "2024-09-01"^^xsd:date ;
  dcterms:language "en" ;
  dcterms:subject  "Semantic Web" ;
  dcterms:license   <https://creativecommons.org/licenses/by/4.0/> ;
  dcterms:isPartOf ex:proceedings2024 .
```

Notice how Dublin Core composes naturally with FOAF (**ex:alice** is a **foaf:Person**) and XSD (the **xsd:date** literal). This kind of layering is the whole point of vocabulary reuse on the Semantic Web.

SKOS—Overview

Simple Knowledge Organization System: a model for representing controlled vocabularies—thesauri, taxonomies, classification schemes, subject heading lists, glossaries.

- W3C Recommendation, 2009.
- Namespace: <http://www.w3.org/2004/02/skos/core#>, prefix **skos:**.
- Lighter and more permissive than OWL: SKOS concepts are not formal classes, so contradictions in a thesaurus don't break a reasoner.
- Used by AGROVOC, MeSH, EuroVoc, the Library of Congress Subject Headings, and many digital library systems.

SKOS—Main Classes and Properties

Classes

- `skos:Concept`—a unit of thought (a topic, a term).
- `skos:ConceptScheme`—a thesaurus or vocabulary as a whole.
- `skos:Collection`, `skos:OrderedCollection` —meaningful groupings of concepts.

Properties

- *Labelling*: `prefLabel`, `altLabel`, `hiddenLabel` (multilingual).
- *Hierarchy*: `broader`, `narrower` (with transitive variants `broaderTransitive`, `narrowerTransitive`).
- *Association*: `related`.
- *Notes*: `definition`, `scopeNote`, `example`, `historyNote`.
- *Membership*: `inScheme`, `topConceptOf`.
- *Mapping across schemes*: `exactMatch`, `closeMatch`, `broadMatch`, `narrowMatch`, `relatedMatch`.

SKOS—Example

```
@prefix skos: <http://www.w3.org/2004/02/skos/core#> .
@prefix ex: <http://example.org/animals/> .
ex:scheme a skos:ConceptScheme ;
  skos:prefLabel "Animal Taxonomy"@en ;
  skos:hasTopConcept ex:Mammal .
ex:Mammal a skos:Concept ;
  skos:prefLabel "Mammal"@en , "Mammifère"@fr ;
  skos:definition "Warm-blooded vertebrate ..."@en ;
  skos:narrower ex:Dog , ex:Cat ;
  skos:topConceptOf ex:scheme .
ex:Dog a skos:Concept ;
  skos:prefLabel "Dog"@en , "Chien"@fr ;
  skos:altLabel "Canine"@en ;
  skos:broader ex:Mammal ;
  skos:related ex:Wolf ;
  skos:exactMatch <http://www.wikidata.org/entity/Q144> ;
  skos:inScheme ex:scheme .
```

Wikidata—Overview

Wikidata: a free, collaborative, multilingual knowledge graph hosted by the Wikimedia Foundation.

- Launched in 2012; the structured-data backbone behind Wikipedia infoboxes.
- All data licensed CC0 (public domain).
- Over 100 million items.
- Public SPARQL endpoint at <https://query.wikidata.org/sparql>.
- Acts as a Linked Data hub: many other vocabularies (FOAF, SKOS, schema.org, GeoNames, VIAF, ...) link to it.

Wikidata—Items, Properties, Statements

Two kinds of entity

- *Items* have Q-identifiers, e.g. `wd:Q42` (Douglas Adams), `wd:Q90` (Paris), `wd:Q102323864` (Yannis Haralambous).
- *Properties* have P-identifiers, e.g. `wdt:P31` (*instance of*), `wdt:P279` (*subclass of*), `wdt:P19` (*place of birth*).

Statements are reified to allow context:

- *Qualifiers*—start time, end time, point in time, applies to part ...
- *References*—citations and source URLs.
- *Ranks*—preferred / normal / deprecated.

Each item carries multilingual labels, descriptions, and aliases, plus sitelinks to the corresponding Wikipedia articles.

SPARQL (2008)

- SPARQL *Simple Protocol and RDF Query Language* is, similarly to SQL, a query language for RDF triple stores.

```
PREFIX me: <http://www.myself.me/something> .
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT DISTINCT ?personA me:loves ?personB
WHERE {
    ?personA rdf:type foaf:Person .
    ?personB rdf:type foaf:Person .
}
```

- A SPARQL engine can have inference support and hence one can find triples *which are not included in the triple store*.
- Like GNU, SPARQL has a recursive name: *SPARQL Protocol and RDF Query Language*.

Example: Wikidata SPARQL

Cities in France with more than one million inhabitants:

```
PREFIX wd:  <http://www.wikidata.org/entity/>
PREFIX wdt: <http://www.wikidata.org/prop/direct/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT DISTINCT ?city ?cityLabel ?population
WHERE {
    ?city wdt:P31/wdt:P279* wd:Q515 ;    # instance of (subclass of)
        wdt:P17 wd:Q142 ;              # country = France
        wdt:P1082 ?population .        # population
    FILTER (?population > 1000000)
    SERVICE wikibase:label {
        bd:serviceParam wikibase:language "fr,en" .
    }
}
ORDER BY DESC(?population)
```

The `wdt:P31/wdt:P279*` pattern—"instance of, then any chain of subclass-of"—is the Wikidata idiom for "is a kind of".

SPARQL (2008)

- Some features (functions, groups):

```
PREFIX ex: <http://example.org/>
```

```
SELECT ?author (COUNT(?book) AS ?nBooks)  
              (AVG(?price) AS ?avgPrice)
```

```
WHERE {  
    ?book a ex:Book ;  
        ex:author ?author ;  
        ex:price ?price .  
}
```

```
GROUP BY ?author  
HAVING (COUNT(?book) >= 3)  
ORDER BY DESC(?nBooks)
```

SPARQL (2008)

- Some more features (multiple hops, filters):

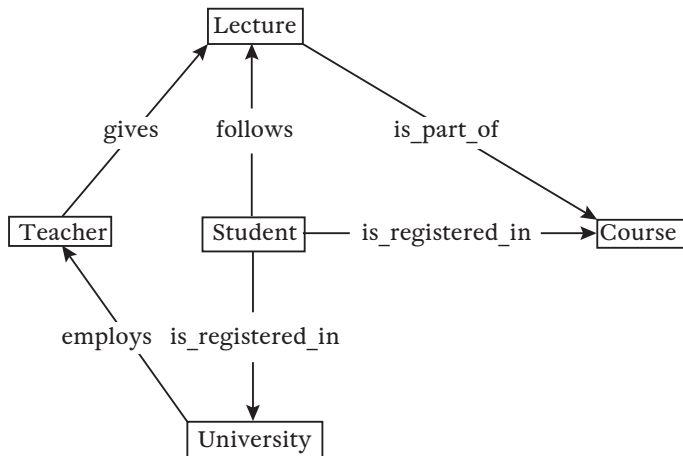
```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX ex: <http://example.org/>
SELECT ?ancestor
WHERE { ex:alice ex:parent+ ?ancestor . }
SELECT ?friendOfFriend
WHERE {
    ex:bob foaf:knows/foaf:knows ?friendOfFriend .
    FILTER (?friendOfFriend != ex:bob)
}
```

- A filter can be a regex(input, pattern, flags):

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name
WHERE { ?x foaf:name ?name
    FILTER regex(?name, "^ali", "i") }
```

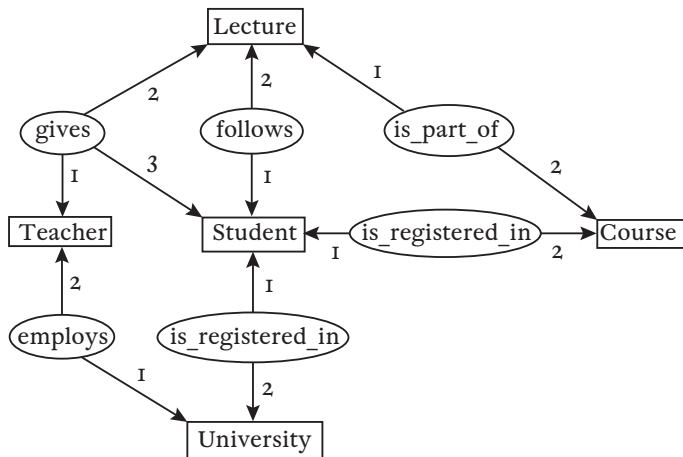
Ontologies

- *Ontologies* are conceptualizations of domain knowledge.



Ontologies compared to Conceptual Graphs

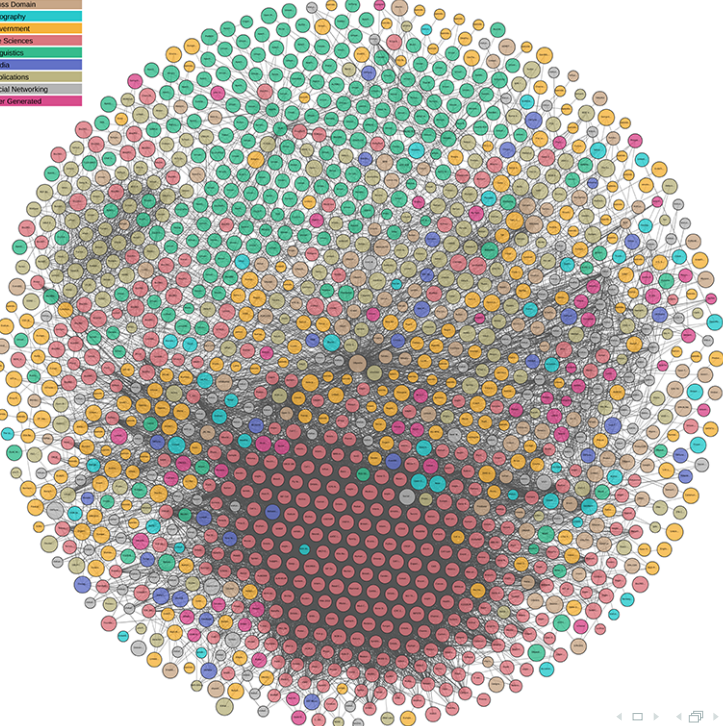
- *Conceptual Graphs* use a slightly different approach.



Linked Open Data

- The *Linked Open Data Cloud* initiative collects ontologies in the *Linked Data* format (RDF, RDFS, OWL).
- On the following slide you can see the status of LOD Cloud as of March 2019: 1,239 datasets with 16,147 links. The real image is here: <https://lod-cloud.net>.
- One of these ontologies is *DBpedia*, a dataset describing 3,4M entities (in 2010), descriptions taken from Wikipedia.





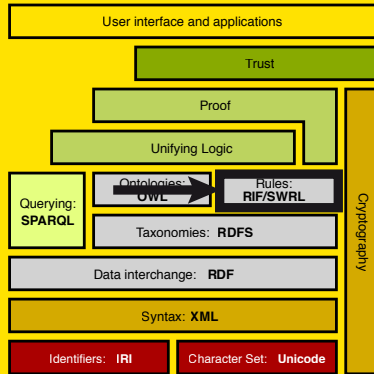
OWL (2004)

- OWL (*Web Ontology Language*) is an extension of RDFS for creating ontologies.
- There are many flavors of OWL, among which OWL-DL which is based on *description logics*.
- Description logics are a decidable fragment of FOL: no variables, no functions, predicates of arity 1 and 2 only.

(A formal language is decidable if there exists a Turing machine that, when given a finite sequence of symbols as input, accepts it if it belongs to the language and rejects it otherwise.)

(FOL is undecidable.)

Part



SWRL (2004)

- SWRL (*Semantic Web Rules Language*) defines *rules* with an antecedent and a subsequent:

$\text{hasParent}(\text{?x1}, \text{?x2}) \wedge \text{hasBrother}(\text{?x2}, \text{?x3}) \Rightarrow$
 $\text{hasUncle}(\text{?x1}, \text{?x3})$

FOL:

$\forall X, Y, Z \text{ hasParent}(X, Y) \wedge \text{hasBrother}(Y, Z) \rightarrow \text{hasUncle}(X, Z).$

SWRL (2004) in Turtle

```
@prefix swrl: <http://www.w3.org/2003/11/swrl#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix ex: <http://example.org/> .

[] a swrl:Imp ;
  swrl:body (
    [ a swrl:IndividualPropertyAtom ;
      swrl:propertyPredicate ex:hasParent ;
      swrl:argument1 [ a swrl:Variable ; rdf:about <urn:swrl#x> ] ;
      swrl:argument2 [ a swrl:Variable ; rdf:about <urn:swrl#p> ] ]
    [ a swrl:IndividualPropertyAtom ;
      swrl:propertyPredicate ex:hasBrother ;
      swrl:argument1 [ a swrl:Variable ; rdf:about <urn:swrl#p> ] ;
      swrl:argument2 [ a swrl:Variable ; rdf:about <urn:swrl#y> ] ]
  ) ;
  swrl:head (
    [ a swrl:IndividualPropertyAtom ;
      swrl:propertyPredicate ex:hasUncle ;
      swrl:argument1 [ a swrl:Variable ; rdf:about <urn:swrl#x> ] ;
      swrl:argument2 [ a swrl:Variable ; rdf:about <urn:swrl#y> ] ]
  ) .
```

Horn Clauses, Prolog

- A clause of the types
 $\neg P_1 \vee \dots \vee \neg P_n$ or
 $\neg P_1 \vee \dots \vee \neg P_n \vee Q$ (at most one positive literal) is called a *Horn Clause*;
- When it has no positive literal it is called a *fact*.
- When it has a positive literal it is called a *rule*.
- *Prolog* is a programming language based on Horn clauses. It is a *French invention* (Alain Colmerauer & Philippe Roussel, 1972, Luminy, Marseille).



SHACL—Overview

Shapes Constraint Language: a W3C Recommendation (2017) for validating RDF graphs against a set of conditions called *shapes*.

- Namespace: <http://www.w3.org/ns/shacl#>, prefix *sh*:.
- Closed-world, “find violations” semantics—the opposite of OWL’s open-world entailment.
- Inputs: a *data graph* (the RDF being validated) and a *shapes graph* (the rules). Output: a *validation report*, itself an RDF graph.
- Two layers:
 - *SHACL Core*—a declarative vocabulary covering the common cases (cardinality, datatype, value range, patterns, ...).
 - *SHACL-SPARQL*—arbitrary SPARQL-based constraints for anything Core can’t express.

Use SHACL when you’d otherwise reach for SWRL or hand-written SPARQL queries to validate data: it’s purpose-built and standardised.

SHACL—Node Shapes and Property Shapes

A *NodeShape* targets a set of nodes; inside it, *property shapes* declare constraints on the values of specific properties.

```
@prefix sh:    <http://www.w3.org/ns/shacl#> .
@prefix xsd:   <http://www.w3.org/2001/XMLSchema#> .
@prefix ex:    <http://example.org/> .

ex:PersonShape a sh:NodeShape ;
  sh:targetClass ex:Person ;
  sh:property [
    sh:path      ex:name ;
    sh:datatype  xsd:string ;
    sh:minCount  1 ;
    sh:maxCount  1 ;
  ] ;
  sh:property [
    sh:path      ex:age ;
    sh:datatype  xsd:integer ;
    sh:minInclusive 0 ;
    sh:maxInclusive 150 ;
  ] ;
```

SHACL—Node Shapes and Property Shapes

```
sh:property [  
  sh:path ex:email ;  
  sh:nodeKind sh:IRI ;  
  sh:pattern "^mailto:" ;  
] .
```

Targets: `sh:targetClass` (all instances of class), `sh:targetNode` (a specific instance), `sh:targetSubjectsOf` (all subjects of some predicate), `sh:targetObjectsOf` (all objects of some predicate).

Common constraints: `minCount`, `maxCount`, `datatype`, `class`, `nodeKind` (such as `sh:IRI`, `sh:BlankNode`, `sh:Literal`, `sh:BlankNode`, `sh:BlankNodeOrLiteral`, `sh:IRIOrLiteral`), `pattern` (a regex!!), `minInclusive`, `maxInclusive`, `in`, `hasValue`, `not`, `or`, `and`, `xone`.

SHACL—A Validation Example

Page-number constraint: consecutive pages must have consecutive numbers (using SHACL-SPARQL for the arithmetic):

```
ex:PageShape a sh:NodeShape ;
  sh:targetClass ex:Page ;
  sh:sparql [
    sh:message "Next page must have number = current + 1" ;
    sh:select """
      SELECT $this ?value
      WHERE {
        $this ex:nextPage ?next ;
              ex:pageNumber ?n1 .
        ?next ex:pageNumber ?n2 .
        FILTER (?n2 != ?n1 + 1)
        BIND (?n2 AS ?value)
      }
    """ ;
  ] .
```

SHACL—A Validation Example

Validation report (excerpt) when a violation is found:

```
[ ] a sh:ValidationReport ;  
    sh:conforms false ;  
    sh:result [  
      a sh:ValidationResult ;  
      sh:focusNode      ex:page_3 ;  
      sh:resultPath      ex:pageNumber ;  
      sh:value           5 ;  
      sh:resultMessage   "Next page must have number = current + 1  
      sh:severity         sh:Violation ;  
    ] .
```

(Instead of “every X must satisfy Y,” we ask “find every X that does not satisfy Y”.)

Unifying Logic, Proof, Trust

- These steps are still far in the future.
- It is about adding more and more logic to the existing tools, together with proofs, which will entail trust.
- According to Wikipedia:
 - Trust to derived statements will be supported by (a) verifying that the premises come from trusted source and by (b) relying on formal logic during deriving new information.
 - User interface is the final layer that will enable humans to use semantic web applications.

(See also <https://sites.ualberta.ca/~reformat/ece720w2012/papers/IEEEXplore-8.pdf>)

Section 7

Description Logics

Description Logics: The Basics

- *Description logics* are variants of FOL featuring two kinds of compromises:
 - ① they add new notation to improve the expressivity of language;
 - ② they are less powerful than FOL, so that they are decidable.
- We have three types of objects:
 - ① *individuals* (like constants in FOL),
 - ② *concepts* or *classes* (unary predicates, the set-theoretic interpretation of which corresponds to sets of individuals),
 - ③ *roles* or *properties* (binary predicates, the set-theoretic interpretation of which corresponds to sets of pairs of individuals).

Protégé: the *wildlife.owl* Ontology

Free software to process OWL ontologies.

The screenshot displays the Protégé ontology editor interface for the *wildlife.owl* ontology. The top navigation bar includes tabs for 'Active ontology', 'Entities', 'Individuals by class', and 'DL Query'. The main interface is divided into several panels:

- Ontology header:** Shows the 'Ontology IRI' as `file:/Users/yannis/Google_Drive/yannis/textmf/cours/ue-tlft/wildlife.owl` and the 'Ontology Version IRI' as `e.g. file:/Users/yannis/Google_Drive/yannis/textmf/cours/ue-tlft/wildlife.owl`. Below this, an 'Annotations' section shows 'owl:VersionInfo' with the text 'Version du 17 octobre 2002'.
- Ontology metrics:** A table providing counts for various ontology elements:

Metrics	
Axiom	58
Logical axiom count	21
Declaration axioms count	23
Class count	13
Object property count	3
Data property count	0
Individual count	3
Annotation Property count	5
Class axioms	
SubClassOf	8
EquivalentClasses	2
DisjointClasses	1
GCI count	0
Hidden GCI Count	0
Object property axioms	
- General class axioms:** A section for viewing general class axioms, currently showing no results.
- Ontology imports:** A section for viewing ontology imports, currently showing no results.
- Imported ontologies:** A section for viewing imported ontologies, currently showing no results.

Protégé: the *wildlife.owl* Ontology

```

@prefix : <http://www.mydomain.org/african> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix xml: <http://www.w3.org/XML/1998/namespace> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix wildlife: <file:wildlife.owl#> .
@base <http://www.mydomain.org/african> .

<file:wildlife.owl> rdf:type owl:Ontology ;
    owl:VersionInfo ""
    Version du 17 octobre 2002
    "" .

```


Protégé: Classes

In *wildlife.owl*, animals form a class.

The screenshot displays the Protégé ontology editor interface for the file `wildlife.owl`. The top navigation bar includes tabs for `Active ontology`, `Entities`, `Individuals by class`, `DL Query`, and `OntoGraf`. Below this, there are tabs for `Annotation properties`, `Datatypes`, and `Individuals`, with sub-tabs for `Classes`, `Object properties`, and `Data properties`.

The main workspace is divided into three panes:

- Class hierarchy: animal**: A tree view showing the hierarchy of classes. The `animal` class is selected and highlighted in blue. Its subclasses are `herbivore`, `giraffe`, `carnivore`, and `lion`. Other classes visible in the hierarchy include `branche`, `feuille`, `owl:allValuesFrom`, `owl:onProperty`, `owl:unionOf`, `plante`, `arbre`, and `plante-comestible`.
- Annotations: animal**: A pane showing annotations for the `animal` class. It displays the annotation `rdfs:comment` with the text "Les animaux forment une classe".
- Description: animal**: A pane showing the description of the `animal` class. It includes fields for `Equivalent To`, `SubClass Of`, `General class axioms`, `SubClass Of (Anonymous Ancestor)`, `Instances`, `Target for Key`, `Disjoint With` (showing `plante`), and `Disjoint Union Of`.

Protégé: Classes

Animals form a class.

```
wildlife:animal rdf:type owl:Class ;
    owl:disjointWith wildlife:branche ,
        wildlife:feuille ,
        wildlife:plante ;
    rdfs:comment ""
    Les animaux forment une classe
    "" .
```

In the formalism of description logics, we will simply write “Animal” to denote this class.

Giraffes are herbivores that are animals:

Giraffe $\sqsubseteq$ Herbivore $\sqsubseteq$ Animal

Protégé: Individuals

Gigi and Giginou are different giraffes.

The screenshot shows the Protégé ontology editor interface. The top bar indicates the active ontology is 'wildlife' (file:/Users/yannis/Google_Drive/yannis/textmf/cours/ue-tft/wildlife.owl). The left sidebar shows the 'Individuals' tab selected, with a list of individuals: Gigi, Giginou, and Léo. The main panel displays the details for the selected individual 'Gigi'. The 'Description' section shows 'Gigi' is of type 'giraffe'. The 'Property assertions' section shows 'Gigi' has the property 'Same Individual As' with the value 'Giginou'. The 'Annotations' section is empty.

Protégé: Individuals

Gigi and Giginou are different giraffes.

```
wildlife:Gigi rdf:type owl:NamedIndividual ,
               wildlife:giraffe ;
               owl:differentFrom wildlife:Giginou ;
               wildlife:poids "800.0"^^xsd:float .
```

```
wildlife:Giginou rdf:type owl:NamedIndividual ,
                  wildlife:giraffe .
```

```
wildlife:Léo rdf:type owl:NamedIndividual ,
               wildlife:lion ;
               wildlife:mange wildlife:Gigi .
```

Giraffe(Gigi)

Giraffe(Giginou)

Gigi $\neq$ Giginou

Protégé: Roles

Un animal can eat something. To eat is the inverse of being eaten.

The screenshot shows the Protégé ontology editor interface. The top bar indicates the active ontology is 'wildlife' (file:/Users/yannis/Google_Drive/yannis/textmf/cours/ue-tift/wildlife.owl). The left sidebar shows the 'Object property hierarchy: mange' with a tree structure: owl:topObjectProperty > is-part-of > mange > mangé-par. The main area displays the 'Annotations: mange' tab, which is currently empty. The bottom right panel shows the 'Description: mange' configuration, including checkboxes for Functional, Inverse functional, Transitive, Symmetric, Asymmetric, Reflexive, and Irreflexive. The 'Equivalent To' field is empty. The 'SubProperty Of' field is empty. The 'Inverse Of' field is set to 'mangé-par'. The 'Domains (Intersection)' field is set to 'animal'. The 'Ranges (Intersection)' field is empty. The 'Disjoint With' field is empty. The 'SuperProperty Of (Chain)' field is empty.

Protégé: Roles

Un animal can eat something. To eat is the inverse of being eaten.

```
wildlife:mange rdf:type owl:ObjectProperty ;
    owl:inverseOf wildlife:mangé-par ;
    rdfs:domain wildlife:animal .
wildlife:mangé-par rdf:type owl:ObjectProperty .
```

To-eat $\equiv$ Is-eaten-by⁻

Is-eaten-by $\equiv$ To-eat⁻

Roles can be

- functional / inverse functional,
- transitive,
- symmetric / asymmetric,
- reflexive / irreflexive,

or nothing of all that...

Protégé: Roles

Leo eats Gigi.

wildlife (file:/Users/yannis/Google_Drive/yannis/txmf/cours/ue-tlft/wildlife.owl) : [file:/Users/yannis/Google_Drive/yannis/txmf/cours/ue-tlft/wildlife.owl]

< > wildlife (file:/Users/yannis/Google_Drive/yannis/txmf/cours/ue-tlft/wildlife.owl) Search...

Active ontology x Entities x Individuals by class x DL Query x OntoGraf x

Annotation properties Datatypes Individuals
Classes Object properties Data properties

Individuals: Léo

☒ Gigi
☒ Giginou
☒ Léo

Usage: Léo

Show: ☒ this ☒ different

Found 6 uses of Léo

- ☒ Léo
 - ☒ Léo Type lion
 - ☒ Individual: Léo
 - ☒ Léo mange Gigi

Description: Léo

Types +

- lion

Same Individual As +

Different Individuals +

Property assertions: Léo

Object property assertions +

- mange Gigi

Data property assertions +

Negative object property assertions +

Negative data property assertions +

Protégé: Roles

Leo eats Gigi.

```
wildlife:Léo rdf:type owl:NamedIndividual ,  
             wildlife:lion ;  
wildlife:mange wildlife:Gigi .
```

Eats(Leo,Gigi)

Basics

- We define three types of formulas:
 - ① **ABox** formulas, “A” like “assertion,” describing concepts and roles (unary and binary predicates on constants);
 - ② **TBox** formulas, “T” like “terminology,” describing relations between concepts;
 - ③ **RBox** formulas, “R” like “relation,” describing a hierarchy of roles, compositions of roles and relations between roles, like mutual exclusion, reflexivity, symmetry, transitivity, etc.

ABox

- A *concept assertion* is a unary predicate of type Giraffe(Gigi).
- A *role assertion* is a binary predicate Eats(Léo,Gigi).
- In description logics we have no *name uniqueness hypothesis*: two individuals of different names can have the same referent, we write $\text{cloclo} \approx \text{claudeFrançois}$, et $\text{samsonFrançois} \neq \text{claudeFrançois}$.

TBox

- Since we can interpret *concepts* (unary predicates) as sets, we can also use set-theoretic relations:
- $\text{Mother} \sqsubseteq \text{Parent}$, equivalent to $\forall X \text{ Mère}(X) \rightarrow \text{Parent}(X)$;
- $\text{Person} \equiv \text{Human}$, equivalent to $\forall X \text{ Personne}(X) \leftrightarrow \text{Human}(X)$.

RBox

- *Concept hierarchy* induces a *role hierarchy*:
- $\text{ParentOf} \sqsubseteq \text{AncestorOf}$, equivalent to $\forall X, Y \text{ParentOf}(X, Y) \rightarrow \text{AncestorOf}(X, Y)$.
- A binary relation can be composed with another
- $\text{BrotherOf} \circ \text{ParentOf} \sqsubseteq \text{UncleOf}$, equivalent to $\forall X, Y, Z \text{FrèreDe}(X, Y) \wedge \text{ParentDe}(Y, Z) \rightarrow \text{OncleDe}(X, Z)$.
- Finally, we can give general characteristics of concepts:
 $\text{Disjoint}(\text{ParentOf}, \text{ChildOf})$, equivalent to $\forall X, Y \text{ParentDe}(X, Y) \rightarrow \neg \text{Parent}(Y, X)$.

Existential Restriction

- Always imitating set theory, we can write $\text{Mother} \equiv \text{Woman} \sqcap \text{Parent}$, equivalent to $\forall X \text{Mother}(X) \leftrightarrow \text{Woman}(X) \wedge \text{Parent}(X)$,
- or $\text{Parent} \equiv \text{Father} \sqcup \text{Mother}$, equivalent to $\forall X \text{Parent}(X) \leftrightarrow \text{Mother}(X) \vee \text{Father}(X)$,
- negation corresponding to the complementary of a set:
 $\neg \text{Bachelor}$ is equivalent to Married .

Existential Restriction

- The complete set corresponds to a predicate that is always true, we write $\top$. To represent a partition we write $\top \sqsubset \text{Even} \sqcup \text{Odd}$.
- Empty set corresponds to a predicate that is always false, we write $\perp$. To represent mutual exclusion we write $\text{Even} \sqcap \text{Odd} \sqsubset \perp$.
- In Protégé, $\top$ is written **owl:Thing** and $\perp$ is written **owl:Nothing**, these two classes are hardcoded.

Existential Restriction

- Let us imagine we have a $\text{Parent}(X, Y)$ role and that we want to characterize the X s that are parents.
So we are asking “I’m looking for X s for which $\exists Y$ such that $\text{Parent}(X, Y)$,”
this is written $\exists \text{Parent}.\top$;
- $\top$ means that we take X s for which there exist an Y , without filtering any further.
- If I were looking those who are parents of at least one daughter, I would write $\exists \text{Parent}.\text{Daughter}$.

Protégé: Existential Restriction

Eatable plants are plants that are eaten by herbivores and by carnivores.

The screenshot shows the Protégé ontology editor interface. The top bar indicates the file path: `wildlife (file:/Users/yannis/Google_Drive/yannis/textmf/cours/ue-tlft/wildlife.owl)`. The main window is divided into several panes:

- Left Pane (Class Hierarchy):** Displays a tree of classes. The hierarchy is:
 - owl:Thing
 - animal
 - herbivore
 - giraffe
 - carnivore
 - lion
 - branche
 - feuille
 - owl:allValuesFrom
 - owl:onProperty
 - owl:unionOf
 - plante
 - arbre
 - plante-comestible** (selected)

- Top Bar:** Includes tabs for "Active ontology", "Entities", "Individuals by class", "DL Query", and "OntoGraf". Below these are tabs for "Annotation properties", "Datatypes", and "Individuals". The "Classes" tab is active, showing sub-tabs for "Object properties", "Data properties", "Annotations", and "Usage".
- Class hierarchy: plante-comestible:** Shows the selected class and its hierarchy.
- Annotations: plante-comestible:** Displays the annotation `rdfs:comment` with the text: "Les plantes comestibles sont des plantes qui sont mangées par des herbivores et par des carnivores".
- Description: plante-comestible:** Shows the class description, including:
- Equivalent To: (empty)
- SubClass Of:
 - mangé-par some carnivore
 - mangé-par some herbivore
 - plante
- General class axioms: (empty)
- SubClass Of (Anonymous Ancestor): (empty)
- Instances: (empty)
- Target for Key: (empty)

Protégé: Existential Restriction

Eatable plants are plants that are eaten by herbivores and by carnivores.

```
wildlife:plante-comestible rdf:type owl:Class ;
    rdfs:subClassOf wildlife:plante ,
        [ rdf:type owl:Restriction ;
          owl:onProperty wildlife:mangé-par ;
          owl:someValuesFrom wildlife:carnivore
        ] ,
        [ rdf:type owl:Restriction ;
          owl:onProperty wildlife:mangé-par ;
          owl:someValuesFrom wildlife:herbivore
        ] ;
    rdfs:comment """
    Les plantes comestibles sont des plantes qui sont mangées
    par des herbivores et par des carnivores
    """ .
```

Eatable-plant $\sqsubset \exists$ Is-eaten-by.Herbivore

Universal Restriction

- Similarly, those that
 - ① have only daughters, or
 - ② are not parentsare denoted by: $\forall \text{Parent.Daughter}$.
- But then what does $\forall \text{Parent}.\top$ mean?

Universal Restriction

- To find out, let us develop formal definitions of the semantics of these notations.
- Let R be a role, and R^I its interpretation. Recall that from a set-theoretic point of view, R^I becomes a set of pairs of elements $R^I = \{(x^I, y^I), \dots\}$ of domain Δ . We will say y^I is a R -*successor* of x^I if the pair (x^I, y^I) belongs to R^I .
- Let C be a concept (so that C^I is a subset of Δ). The interpretation of $\exists R.C$ is $\{x^I \mid \text{some successors of } x^I \text{ are in } C^I\}$.
- And the one of $\forall R.C$ is $\{x^I \mid \text{all the successors of } x^I \text{ are in } C^I\}$.
- So what is the interpretation of $\forall \text{Parent}.\top$?

Protégé: Universal Restriction

Giraffes are herbivores that eat only leaves.

The screenshot shows the Protégé ontology editor interface. The top bar indicates the active ontology is 'wildlife' (file:/Users/yannis/Google_Drive/yannis/textmf/cours/ue-tlft/wildlife.owl). The left pane displays the 'Class hierarchy: giraffe' tree, where 'giraffe' is highlighted under 'herbivore'. The right pane shows the 'Annotations: giraffe' and 'Description: giraffe' sections.

Class hierarchy: giraffe

- owl:Thing
 - animal
 - herbivore
 - giraffe**
 - carnivore
 - lion
 - branche
 - feuille
 - owl:allValuesFrom
 - owl:onProperty
 - owl:unionOf
 - plante
 - arbre
 - plante-comestible

Annotations: giraffe

- rdfs:comment**: Les giraffes sont herbivores et ne mangent que des feuilles
- rdfs:subClassOf**

Description: giraffe

- Equivalent To**
- SubClass Of**
 - herbivore
 - mange **only** feuille
- General class axioms**
- SubClass Of (Anonymous Ancestor)**
- Instances**
 - Gigi
 - Giginou
- Target for Key**

Protégé: Universal Restriction

Giraffes are herbivores that eat only leaves.

```
wildlife:giraffe rdf:type owl:Class ;
  rdfs:subClassOf wildlife:herbivore ,
    [ rdf:type owl:Restriction ;
      owl:onProperty wildlife:mange ;
      owl:allValuesFrom wildlife:feuille
    ] ;
  rdfs:comment """
  Les giraffes sont herbivores et ne mangent que des feuilles
  """ ;
  rdfs:subClassOf "" .
```

Giraffe $\sqsubset$ Herbivore $\sqcap \forall \text{Eats.Leave}$

Restrictions At-Most and At-Least, Reflexivity

- We can specify “at least n successors”: $\geq nR.C$, as well as “at most n successors”: $\leq nR.C$.
- Another notation allows to describe the set of elements for which a role R is reflexive: $\exists R.Self$.
- One way to describe a concept is by enumerating its members, as in $\{jacques\} \sqsubseteq \exists Parent.\{julie\}$.

Restrictions At-Most and At-Least, Reflexivity

- We denote R^- the *inverse role* of R , e.g., Parent^- is equivalent to Child since $\forall X, Y \text{Parent}(X, Y) \leftrightarrow \text{Child}(Y, X)$.
- We denote U the *universal role*, i.e., the one that maps any elements to any other elements, including itself.
- Finally, we denote R^+ the *transitive closure* of R , the smallest transitive role containing R .

SR $\mathcal{OIQ}$ Logic

- *SR $\mathcal{OIQ}$* is a description logic, such that: if N_C is a concept, N_R a role and N_I an individual, then we define iteratively:
 - 1 a *concept expression* $\mathbf{C}$ as $\mathbf{C} ::= N_C \mid (\mathbf{C} \sqcap \mathbf{C}) \mid \mathbf{C} \sqcup \mathbf{C} \mid \neg \mathbf{C} \mid \top \mid \perp \mid \exists \mathbf{R}.\mathbf{C} \mid \forall \mathbf{R}.\mathbf{C} \mid \geq n \mathbf{R}.\mathbf{C} \mid \leq n \mathbf{R}.\mathbf{C} \mid \exists \mathbf{R}.\text{Self} \mid \{N_I\}$,
 - 2 a *role expression* $\mathbf{R}$ as $\mathbf{R} ::= U \mid N_R \mid N_R^-$.
- Axioms of an *SR $\mathcal{OIQ}$* ontology are as follows:
 - 1 ABox: $\mathbf{C}(N_I)$, $\mathbf{R}(N_I, N_I)$, $N_I \approx N_I$, $N_I \not\approx N_I$,
 - 2 TBox: $\mathbf{C} \sqsubseteq \mathbf{C}$, $\mathbf{C} \equiv \mathbf{C}$,
 - 3 RBox: $\mathbf{R} \sqsubseteq \mathbf{R}$, $\mathbf{R} \equiv \mathbf{R}$, $\mathbf{R} \circ \mathbf{R} \sqsubseteq \mathbf{R}$, *Disjoint*($\mathbf{R}, \mathbf{R}$).

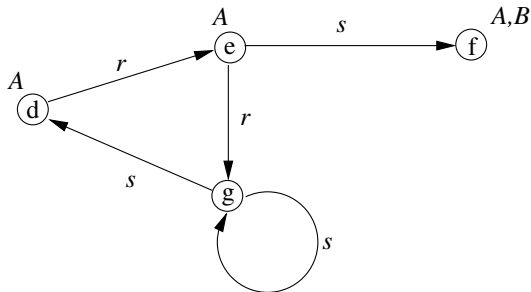
Properties of $\mathcal{SROIQ}$

- Here are the properties of description logic $\mathcal{SROIQ}$:
 - ① $\mathcal{S}$ (also named $\mathcal{ALCR}^+$): ($\mathcal{AL}$) availability of concept and role names, of $\top$, of constructor $\sqcap$ (conjunction), of the universal quantifier, ($\mathcal{C}$) of the concept negation, ($\mathcal{R}^+$) of transitive closure;
 - ② $\mathcal{R}$: role inclusion, reflexivity, irreflexivity, role exclusion;
 - ③ $\mathcal{O}$: possibility of describing a concept extensionally ($\{N_{I,1}, \dots, N_{I,n}\}$);
 - ④ $\mathcal{I}$: possibility of having inverse roles ($\mathcal{R}^-$);
 - ⑤ $\mathcal{Q}$: possibility of having fully quantified restrictions ($\leq n, \geq n$).
- The [OWL 2](#) standard of the Web consortium corresponds to logic $\mathcal{SROIQ}$.

Description Logics: Exercise 1

(Our interpretations are using the “closed world” assumption, i.e., the hypothesis that every true statement is also known to be true.)

In interpretation I below (in domain $\Delta = \{d, e, f, g\}$):



enumerate elements
of the following concepts:

$A \sqcup B$
 $\exists s.A$
 $\exists s.\neg A$
 $\forall s.A$

$\exists s.\exists s.\exists s.\exists s.A$
 $\forall r.A \sqcap \forall r.\neg A$
 $\neg \exists r.(\neg A \sqcap \neg B)$
 $\exists r.(A \sqcap \forall s.\neg B) \sqcap \neg \forall r.\exists r.(A \sqcup \neg A)$

Description Logic: Exercise 2

- Modelize in description logic the assertion “All trucks with trailers (= camion remorque) have exactly 18 wheels”.
- We will use the role “Has” and concepts “Wheel,” “Trailer,” “Truck”.

Description Logic: Exercise 2

- Modelize in description logic the assertion “All trucks with trailers (= camion remorque) have exactly 18 wheels”.
- We will use the role “Has” and concepts “Wheel,” “Trailer,” “Truck”.
- In description logic:
$$(1\text{Has.Trailor} \sqcap \text{Truck}) \sqsubseteq 18\text{Has.Wheel}.$$

Description Logic: Exercise 2

- Modelize in description logic the assertion “All trucks with trailers (= camion remorque) have exactly 18 wheels”.
- We will use the role “Has” and concepts “Wheel,” “Trailer,” “Truck”.
- In description logic:

$$(1\text{Has.Trailor} \sqcap \text{Truck}) \sqsubseteq 18\text{Has.Wheel.}$$
- Do you see the trap?

Description Logic: Exercise 2

- Modelize in description logic the assertion “All trucks with trailers (= camion remorque) have exactly 18 wheels”.
- We will use the role “Has” and concepts “Wheel,” “Trailer,” “Truck”.
- In description logic:

$$(1\text{Has.Trailor} \sqcap \text{Truck}) \sqsubseteq 18\text{Has.Wheel.}$$
- Do you see the trap?
- 1Has.Wheel means that there exist exactly 18 individuals of class Wheel belonging to the truck. But can we guarantee that these individuals represent different elements in any interpretation? We need (a) to name them, and (b) to write pairwise inequalities.