Lecture 4: Reasoning and Data Exchange

TIES452 Practical Introduction to Semantic Technologies
Autumn 2014

University of Jyväskylä

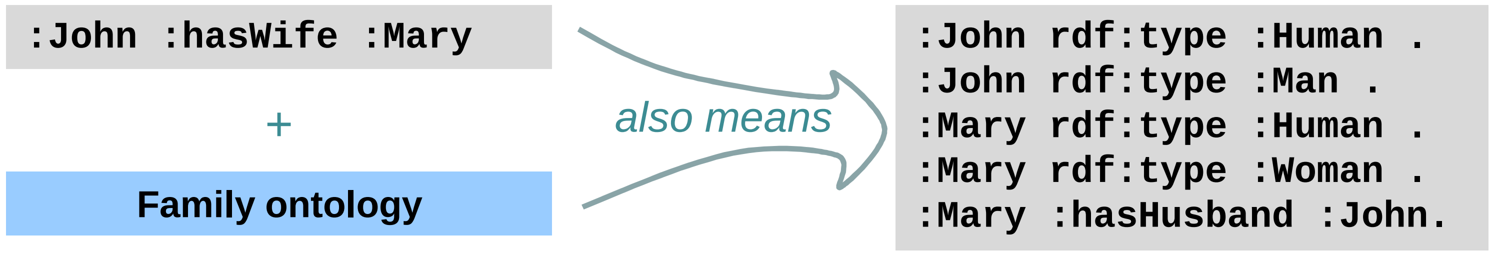
Khriyenko Oleksiy

Part 1

Reasoning and rules

Reasoning types

- Two basic types:
  - Ontology-based reasoning
    - Classification-based inference (e.g. RDF-S, OWL reasoning)
    - The inference rules for RDF-S or OWL are ***fixed.*** Therefore: No need for rule engine -> procedural algorithm sufficient



```
:John :hasWife :Mary
```

```
:John rdf:type :Human .
:John rdf:type :Man .
:Mary rdf:type :Human .
:Mary rdf:type :Woman .
:Mary :hasHusband :John.
```

  - Rule-based reasoning
    - General rule-based inference (semantic rules)
    - Further classification: forward-chaining and backward-chaining

Some rules of RDF Schema

- If a resource is an instance of a class, it is also an instance of any super-class of that class (*any human is a mammal*).

```
:Mammal rdf:type owl:Class.
:Human rdf:type owl:Class.
:Human rdfs:subClassOf :Mammal.
:John rdf:type :Human.
```

also means

```
:John rdf:type :Mammal.
```

- If a statement with a property is made, the statement with any super-property is also true (*if you love something, you also like it*).

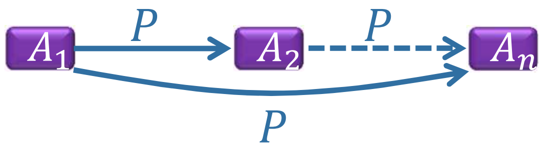
```
:like rdf:type owl:ObjectProperty.
:love rdf:type owl:ObjectProperty.
:love rdfs:subPropertyOf :like.
:John :love :Mary.
```

also means

```
:John :like :Mary.
```

Some rules of RDF Schema

- Transitive property:
  - If class A is a sub-class of B, while B is a sub-class of C, then A is a sub-class of C (*mother is woman, woman is human, therefore mother is human*).
  - Also applies to sub-properties
- Example: *rdfs:subClassOf* and *rdfs:subPropertyOf* are transitive properties



$$\begin{tikzcd} A_1 \arrow[r, "P"] \arrow[rr, bend right, "P"'] & A_2 \arrow[r, dashed, "P"] & A_n \end{tikzcd}$$

```
:Human rdf:type owl:Class.
:Woman rdf:type owl:Class.
:Mother rdf:type owl:Class.
:Woman rdfs:subClassOf :Human.
:Mother rdfs:subClassOf :Woman.
:prefer rdf:type owl:ObjectProperty.
:like rdf:type owl:ObjectProperty.
:love rdf:type owl:ObjectProperty.
:like rdfs:subPropertyOf :prefer.
:love rdfs:subPropertyOf :like.
```

also means

```
:Mother rdfs:subClassOf :Human.
:love rdfs:subPropertyOf :prefer.
```

Some rules of RDF Schema

- If a property is defined to have class A as its domain, and a statement with that property is made, the subject of the statement must be an instance of A.
  - The same for the range of a property and the object of a statement.

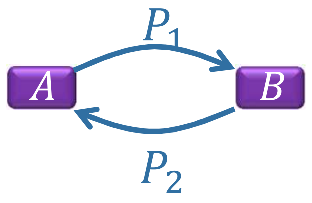
```
:Human rdf:type owl:Class .
:love rdf:type owl:ObjectProperty ;
      rdfs:domain :Human ;
      rdfs:range :Human .
:John :love :Mary .
```

also means

```
:John rdf:type :Human .
:Mary rdf:type :Human .
```

Some rules of OWL

- Some of the property characteristics allow reasoners to infer new knowledge about instances and their relations:
  - *owl:inverseOf*



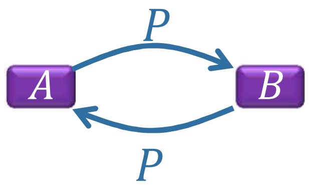
```
:Human rdf:type owl:Class .
:hasChild rdf:type owl:ObjectProperty .
:hasParent rdf:type owl:ObjectProperty .
:hasChild owl:inverseOf :hasParent .
:John rdf:type :Human .
:Mary rdf:type :Human .
:John :hasChild :Mary .
```

also means

```
:Mary :hasParent :John .
```

Some rules of OWL

- Some of the property characteristics allow reasoners to infer new knowledge about instances and their relations:
  - *owl:SymmetricProperty*



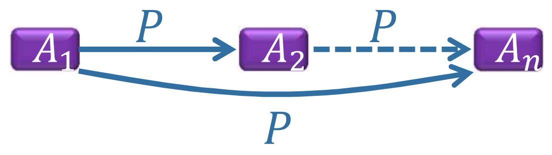
```
:Human rdf:type owl:Class .
owl:inverseOf rdf:type owl:SymmetricProperty .
:hasChild rdf:type owl:ObjectProperty .
:hasParent rdf:type owl:ObjectProperty .
:hasChild owl:inverseOf :hasParent .
:John rdf:type :Human .
:Mary rdf:type :Human .
:Mary :hasParent :John .
```

```
:hasParent owl:inverseOf :hasChild .
```

```
:John :hasChild :Mary .
```

Some rules of OWL

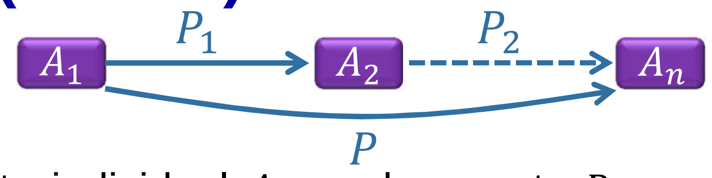
- Some of the property characteristics allow reasoners to infer new knowledge about instances and their relations:
  - *owl:TransitiveProperty*



```
:Human rdf:type owl:Class .
:bossOf rdf:type owl:TransitiveProperty .
:John rdf:type :Human .
:Michael rdf:type :Human .
:Mary rdf:type :Human .
:John :bossOf :Mary .
:Mary :bossOf :Michael .
```

```
:John :bossOf :Michael .
```

Property chains (OWL-2)



- *owl:propertyChainAxiom* ***(OWL-2)***

  Simply: If the property P_1 relates individual A_1 to individual A_2, and property P_2 relates individual A_2 to individual A_n, then property P relates individual A_1 to individual A_n;

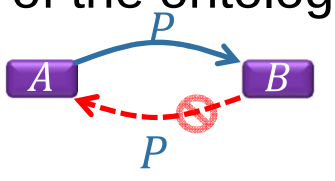
```
:hasParent rdf:type owl:ObjectProperty .
:hasGrandparent rdf:type owl:ObjectProperty ;
                owl:propertyChainAxiom ( :hasParent :hasParent ) .
:hasGrandGrandparent rdf:type owl:ObjectProperty .
[ rdf:type owl:ObjectProperty ;
  owl:propertyChainAxiom (:hasGrandparent :hasParent)] rdfs:subPropertyOf :hasGrandGrandparent.
[ rdf:type owl:ObjectProperty ;
  owl:propertyChainAxiom (:hasParent :hasGrandparent)] rdfs:subPropertyOf :hasGrandGrandparent.
:Human rdf:type owl:Class .
:John rdf:type :Human .
:Michael rdf:type :Human .
:Mary rdf:type :Human .
:Katarina rdf:type :Human ;
          :hasParent :Mary .
:Mary :hasParent :Michael .
:Michael :hasParent :John .
```

```
:Katarina :hasGrandparent :Michael .
:Mary :hasGrandparent :John .
```

```
:Katarina :hasGrandGrandparent :John .
```

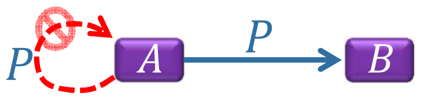
Some rules of OWL

- Some of the property characteristics set certain conditions and allow reasoners to detect inconsistency of the ontology:
  - *owl:AsymmetricProperty*



```
:Human rdf:type owl:Class .
:isChildOf rdf:type owl:AsymmetricProperty .
:John rdf:type :Human .
:Mary rdf:type :Human .
:John :isChildOf :Mary .
…
:Mary :isChildOf :John .  inconsistency
```

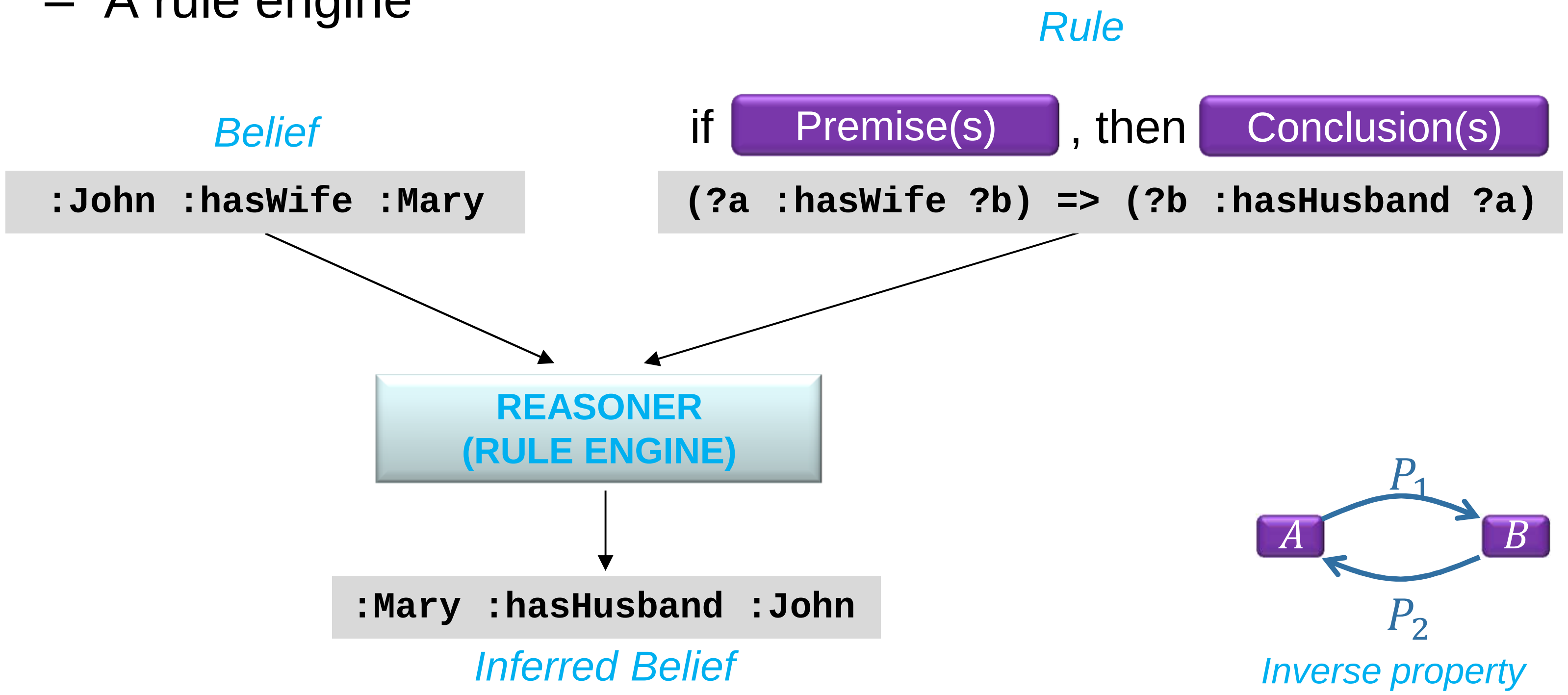
  - *owl:IrreflexiveProperty*



```
:Human rdf:type owl:Class .
:motherOf rdf:type owl:IrreflexiveProperty .
:John rdf:type :Human .
:Mary rdf:type :Human .
:Mary :motherOf :John .
…
:Mary :motherOf :Mary .  inconsistency
```

Rule-based reasoning

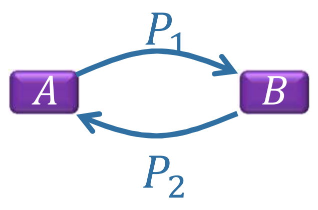
- The inference based on free-form rules always requires:
  - A language for representing the rules
  - A rule engine



```
:John :hasWife :Mary
```

```
(?a :hasWife ?b) => (?b :hasHusband ?a)
```

```
:Mary :hasHusband :John
```



Inverse property

Rule-based reasoning

- The OWL language is not able to express all relations (ex: it cannot express the relation “*child of married parents*“).
- The expressivity of OWL can be extended by adding rules to an ontology.
- Need for rule definition language:
  - ***SWRL*** (Semantic Web Rule Language)
  - ***Notation 3 (N3)*** logic
  - ***RIF*** (Rule Interchange Format)

SWRL

- ***SWRL*** (Semantic Web Rule Language):
  - Part of many tools (e.g. Pellet)
  - Basic form is XML, but also available in human-readable form
  - unary predicates for describing classes and data types,
  - binary predicates for properties,
  - some special built-in n-ary predicates.
- SWRL rules are supported:
  - Protege OWL editor.
  - reasoners Pellet and Hermit.

SWRL

- SWRL predicates:
  - Class expressions: Class atom: `Person(?x)` `Man(Fred)`

```
Man(?p) -> Person(?p)
```

  - Property expressions:
  - Individual Property atom: `hasBrother(?x,?y)` `hasSibling(Fred,?y)`
  - Data Valued Property atom: `hasAge(?x,?age)` `hasAge(?x,232)` `hasName(Fred,”Fred”)`

```
Person(?p), hasSibling(?p,?s), Man(?s) -> hasBrother(?p,?s)
```

  - Data range restrictions

```
Person(?p), integer[>= 18,<= 65](?age), hasAge(?p, ?age) -> hasDriverAge(?p, true)
```

  - OWL Class expressions in SWRL Rules

```
Person(?x), hasChild min 1 Person(?x) -> Parent(?x)
```

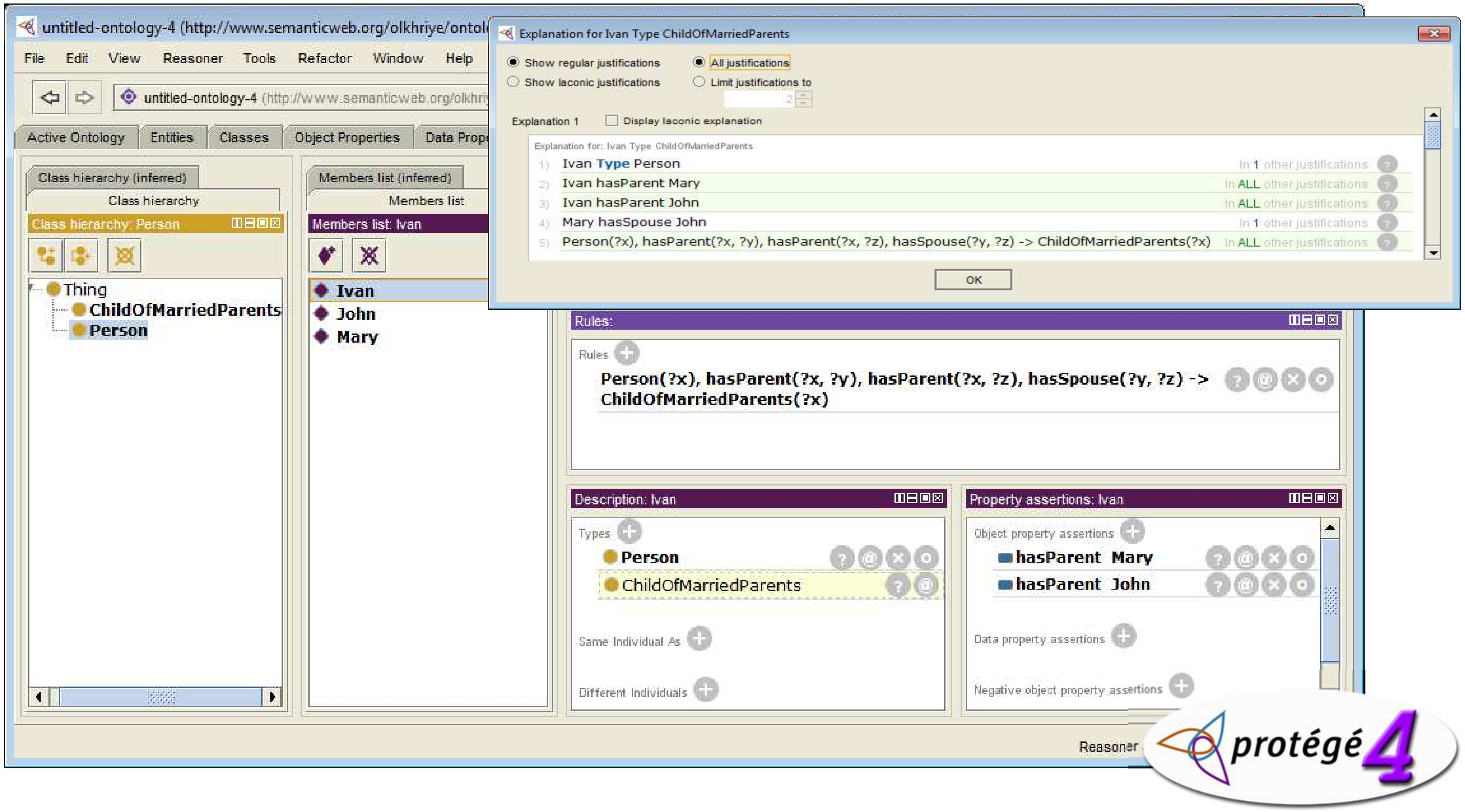
  - Core SWRL built-ins (http://www.daml.org/rules/proposal/builtins.html)

```
Person(?p), hasAge(?p, ?age), swrlb:greaterThan(?age, 18) -> Adult(?p)
```

```
Person(?p), bornOnDate(?p, ?date), xsd:date(?date),
swrlb:date(?date, ?year, ?month, ?day, ?timezone) -> bornInYear(?p, ?year)
```

SWRL

- Rule definition in Protégé 4.x (5.x)
  - Open rule tub from the menu: *Window – Views – Ontology Views – Rules*


untitled-ontology-4 (http://www.semanticweb.org/olkhriye/ontol
File Edit View Reasoner Tools Refactor Window Help
Active Ontology Entities Classes Object Properties Data Prope
Class hierarchy (inferred)
Class hierarchy
Class hierarchy: Person
Thing
ChildOfMarriedParents
Person
Members list (inferred)
Members list
Members list: Ivan
Ivan
John
Mary
Reasoner
protégé 4

Explanation for Ivan Type ChildOfMarriedParents

(● Show regular justifications) ○ Show laconic justifications
(● All justifications) ○ Limit justifications to 2

Explanation 1 ☐ Display laconic explanation

Explanation for: Ivan Type ChildOfMarriedParents

1) Ivan **Type** Person — In 1 other justifications
2) Ivan hasParent Mary — In ALL other justifications
3) Ivan hasParent John — In ALL other justifications
4) Mary hasSpouse John — In 1 other justifications
5) Person(?x), hasParent(?x, ?y), hasParent(?x, ?z), hasSpouse(?y, ?z) -> ChildOfMarriedParents(?x) — In ALL other justifications

OK

Rules:

Rules

Person(?x), hasParent(?x, ?y), hasParent(?x, ?z), hasSpouse(?y, ?z) -> ChildOfMarriedParents(?x)

Description: Ivan

Types
- **Person**
- ChildOfMarriedParents

Same Individual As

Different Individuals

Property assertions: Ivan

Object property assertions
- **hasParent Mary**
- **hasParent John**

Data property assertions

Negative object property assertions

Notation 3 (N3) logic rules

- *Notation 3 (N3)* logic rule expression
  - graph definition ({}) give a possibility to write formulas in rules

```
@prefix log: <http://www.w3.org/2000/10/swap/log#>.
@forAll :x, :y, :z.
{ :x family:parent :y . :y family:brother :z } log:implies { :x family:uncle :z }.
```

- Shorthand symbols:
  - *?* – for universal variables "*@forAll*";
  - *_:* or better *[ ]* – for existential variables "*@forSome*" (blank node);
  - **=>** – for implies (*log:implies*);
  - **<=>** – for meaning (*log:means*);
  - **=** – for equivalents (*owl:equivalentTo*);

```
{?x family:parent ?y. ?y family:brother ?z} => {?x family:uncle ?z}.
```

- Built-in Functions: used by CWM (http://www.w3.org/2000/10/swap/doc/CwmBuiltins)

```
{ ?x f:age ?ag . ?ag math:lessThan 30} => { ?x rdf:type f:YoungPerson } .
```

```
{ ex:d test:point ?x.  ?x math:sin ?y } =>  {...} .
```

```
{ ?x a ex:TestData.
    ( ?x 1 ) math:sum ?y.
    (  ?y  " is one more than " ?x ) string:concatenation ?s
} => { ?s a ex:Result }.
```

N3 logic: Some RDFS rules

```
{ ?A rdfs:subClassOf ?B .
  ?S rdf:type ?A } => { ?S rdf:type ?B } .

{ ?P rdfs:subPropertyOf ?R .
  ?S ?P ?O } => { ?S ?R ?O } .

{ ?P rdfs:domain ?C. ?S ?P ?O } => { ?S rdf:type ?C } .

{ ?P rdfs:range ?C. ?S ?P ?O } => { ?O rdf:type ?C } .

{ ?B rdfs:subClassOf ?C .
  ?A rdfs:subClassOf ?B } => { ?A rdfs:subClassOf ?C } .

{ ?Q rdfs:subPropertyOf ?R.
  ?P rdfs:subPropertyOf ?Q } => { ?P rdfs:subPropertyOf ?R } .
```

N3 logic: Some OWL rules

```
owl:inverseOf rdf:type owl:SymmetricProperty .
```

```
{ ?P rdf:type owl:SymmetricProperty. ?S ?P ?O } => { ?O ?P ?S } .

{ ?P owl:inverseOf ?Q . ?S ?P ?O } => { ?O ?Q ?S } .

{ ?P rdf:type owl:TransitiveProperty.
  ?S ?P ?X. ?X ?P ?O } => { ?S ?P ?O } .

{ ?P rdf:type owl:InverseFunctionalProperty.
  ?X ?P ?O. ?Y ?P ?O } => { ?X owl:sameAs ?Y }.

{ ?P rdf:type owl:FunctionalProperty.
  ?S ?P ?X. ?S ?P ?Y } => { ?X owl:sameAs ?Y }.

{ ?A rdfs:subPropertyOf ?B.
  ?B rdfs:subPropertyOf ?A } => { ?A owl:equivalentProperty ?B }.
```

See http://www.agfa.com/w3c/euler/owl-rules.n3 for all OWL rules.

RIF

- ***RIF*** (Rule Interchange Format) is W3C Recommendation June 2010

RIF is part of the infrastructure for the semantic web. The design of RIF is based on the observation that there are many "rules languages" in existence, and what is needed is to exchange rules between them.

- RIF includes three dialects:
  - Core dialect (which is extended into others)

```
Document(  Prefix(ppl <http://example.com/people#>)
           Prefix(cpt <http://example.com/concepts#>)
           Prefix(bks <http://example.com/books#>)
  Group(   Forall ?Buyer ?Item ?Seller (
                  cpt:buy(?Buyer ?Item ?Seller) :- cpt:sell(?Seller ?Item ?Buyer))
                  cpt:sell(ppl:John bks:LeRif ppl:Mary)))
```

  - Basic Logic Dialect (BLD)

```
Document(  Base(<http://example.com/people#>)
           Prefix(cpt <http://example.com/concepts#>)
           Prefix(bks <http://example.com/books#>)
  Group(   Forall ?Buyer ?Item ?Seller (
                  cpt:buy(?Buyer ?Item ?Seller) :- cpt:sell(?Seller ?Item ?Buyer))
                  cpt:sell(<John> bks:LeRif "Mary"^^rif:iri)))
```

  - Production Rule Dialect (PRD)

```
Prefix(ex <http://example.com/2008/prd1#>)
(* ex:rule_1 *)
Forall ?customer ?purchasesYTD (
 If   And( ?customer#ex:Customer
           ?customer[ex:purchasesYTD->?purchasesYTD]
           External(pred:numeric-greater-than(?purchasesYTD 5000)) )
 Then Do( Modify(?customer[ex:status->"Gold"]) ) )
```

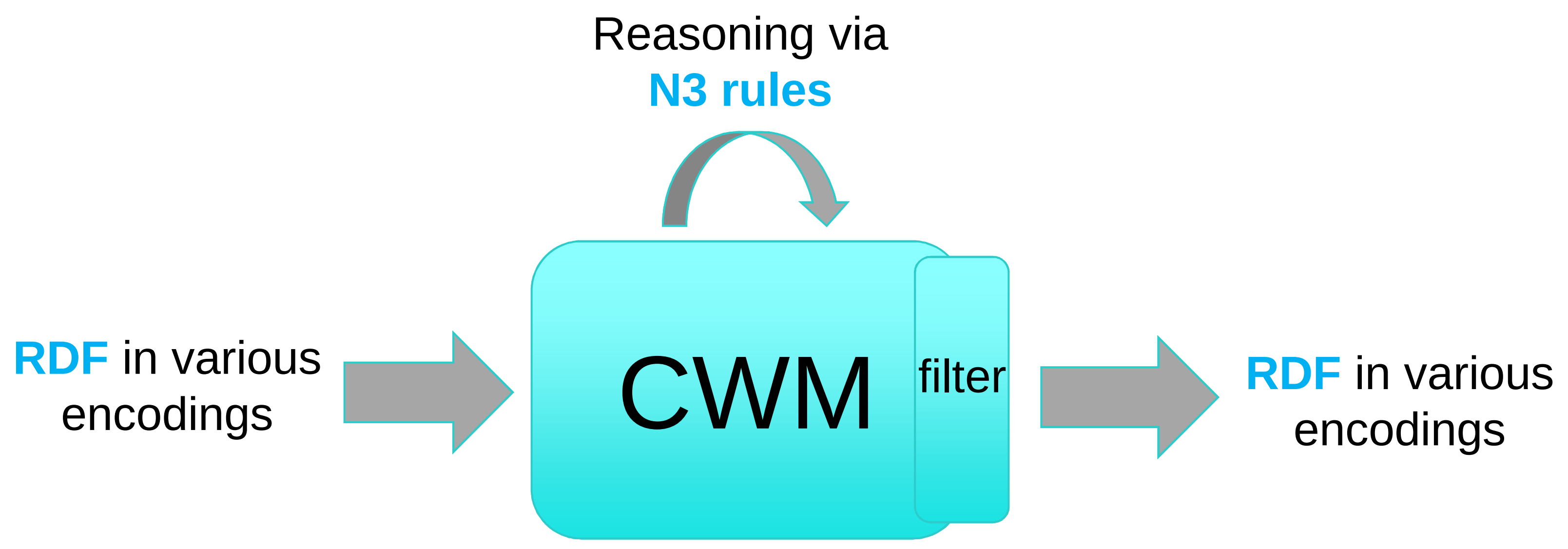
Forward vs. backward-chaining reasoning

- Forward
  - Input: rules + data
  - Output: extended data
  - Starts with available facts
  - Uses rules to derive new facts (which can be stored)
  - Stops when there is nothing else to be derived
- Backward
  - Input: rules + data + hypothesis (statement)
  - Output: Statement is true / Statement is false
  - Goes backwards from the hypothesis to the set of axioms (our data)
  - If it can find the path to the original axioms, then the hypothesis is true (otherwise false)

CWM

- ***Forward-chaining reasoner*** written in Python
- Originally to show capabilities of N3
- Link: http://www.w3.org/2000/10/swap/doc/cwm
- ”Cwm (*pronounced coom*) is a general-purpose data processor for the semantic web”. It can be used for:
  - *querying,*
  - *checking,*
  - *transforming,*
  - *filtering information…*
- Deals with open worlds!
- CWM’s function:
  - Loads data in N3 or RDF/XML + rules in N3
  - Applies rules to data
  - Output result in N3 or RDF/XML

CWM


Reasoning via
N3 rules
RDF in various encodings
CWM
filter
RDF in various encodings

CWM usage practical

- You must have command **python** in your PATH variable

```
e.g. set PATH=%PATH%;c:\Python27
```

- Basic usage:

```
python cwm <COMMAND> <OPTIONS> <STEPS>
```

- By default the output goes to standard output
  - If you want to store it in a file, use redirect, e.g.:

```
python cwm input.n3 --think --data --rdf > result.rdf
```

- Useful use cases:

source format *source file* *destination format*

```
  python cwm --n3 data.n3 --filter=rules.n3 --n3
```

Show only new reasoned facts by applying rules in rules.n3

```
python cwm --n3 data.n3 --apply=rules.n3 --n3
```

Show both the old data from data.n3 together with new reasoned data

```
python cwm --n3 data.n3 --think=rules.n3 --n3
```

As –apply, but continue until no more rule matches (or forever!)

CWM usage: Example

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
```

Data

```
:Human rdf:type owl:Class .
:dan rdf:type :Human .
:peter rdf:type :Human .
:mary rdf:type :Human .
:jon rdf:type :Human .
:betty rdf:type :Human .

:ancestorOf rdf:type owl:TransitiveProperty.
:hasSpouse rdf:type owl:SymmetricProperty.
:brotherOf rdf:type owl:ObjectProperty.
:sisterOf rdf:type owl:ObjectProperty.
owl:inverseOf rdf:type owl:SymmetricProperty .

:brotherOf owl:inverseOf :sisterOf .

:dan :ancestorOf :peter .
:peter :ancestorOf :jon .
:peter :hasSpouse :mary .
:betty :sisterOf :jon .
```

```
:dan :ancestorOf :jon .
:mary :hasSpouse :peter .
:sisterOf owl:inverseOf :brotherOf .
```

```
:jon :brotherOf :betty .
```

+

Rules

```
{ ?P rdf:type owl:SymmetricProperty .
  ?S ?P ?O
} => {?O ?P ?S} .

{ ?P owl:inverseOf ?Q .
  ?S ?P ?O
} => {?O ?Q ?S} .

{ ?P rdf:type owl:TransitiveProperty .
  ?S ?P ?X .
  ?X ?P ?O
} => {?S ?P ?O} .
```

CWM practical tips

practical

- When you write a file for CWM, *always put dot after the last statement!*
- You don’t have to separate data and rules into two files
  - If you use N3 as your notation, then they can be in one file
  - Example:

```
python cwm --n3 dataAndRules.n3 --rules
python cwm --n3 dataAndRules.n3 --think
```

- CWM can be used to convert files without reasoning
  - Example:

```
python cwm --rdf source.rdf --n3 > destination.n3
```

- More CWM command line arguments are available at (http://www.w3.org/2000/10/swap/doc/CwmHelp) or using `python cwm --help`

Euler/EYE

- Originally ***backward-chaining reasoner*** for N3 logic – inference engine ***Euler***
- ***Euler YAP Engine (EYE)*** – a backward-forward-backward chaining reasoner design enhanced with Euler path detection (reasoning is grounded in First Order Logic).
- Home: http://www.agfa.com/w3c/euler/
- Download: http://sourceforge.net/projects/eulersharp/files/eulersharp/
- Implemented in several languages: *Java, C#, Python, Javascript and Prolog*
- Input: *rules + data + hypothesis*
- Output: *Chain of rules that lead to the hypothesis (if the hypothesis is true)*

Other reasoners

- RacerPro by Racer Systems (commercial)
- http://www.racer-systems.com/

- Jena by Hewlett-Packard (open-source)
- http://jena.sourceforge.net/

- Pellet by Clark & Parsia (open-source)
- http://pellet.owldl.com/

- FaCT++ by University of Manchester (open-source, in C++)
- http://owl.man.ac.uk/factplusplus/

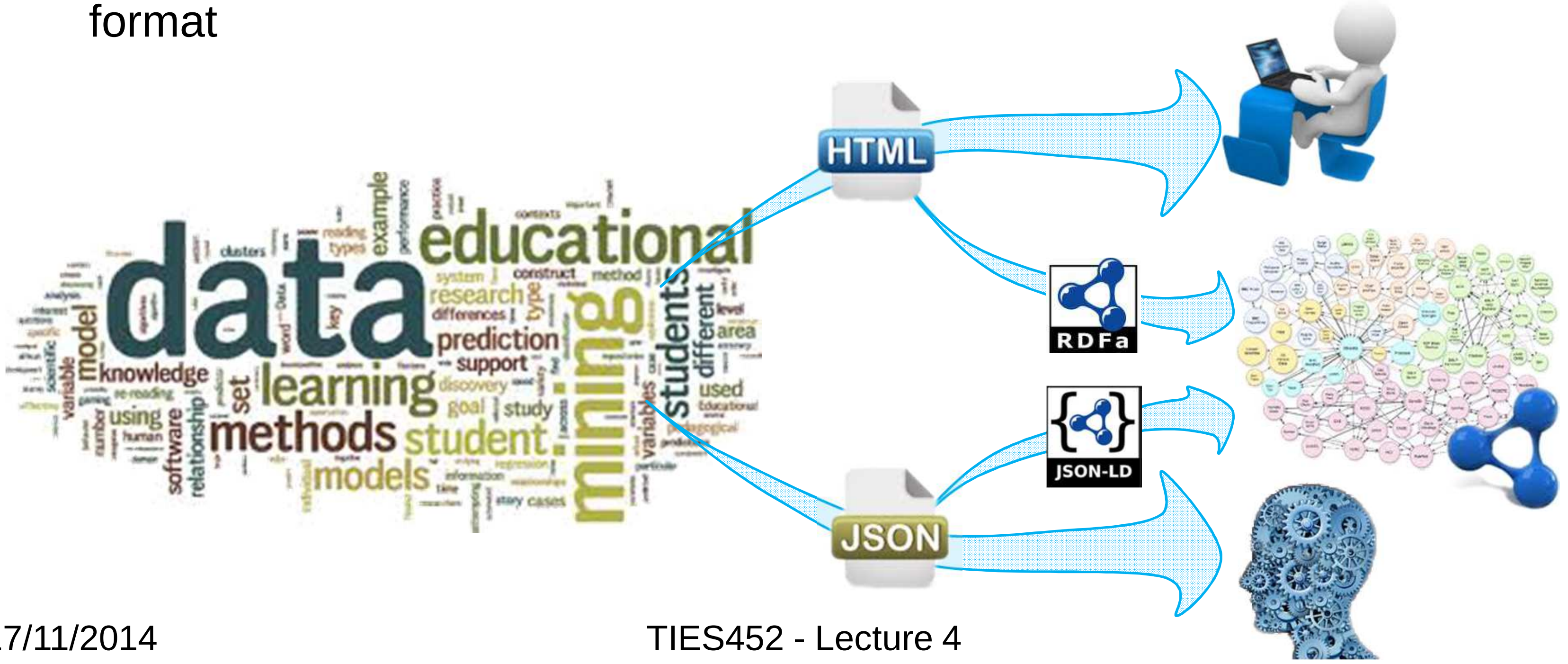
- Sesame supports RDFS reasoning as well

Part 2

Data Exchange

Machine readable data exchange

- ***RDFa*** – Resource Description Framework in attributes (W3C Recommendation). It is a domain-independent way to explicitly embed RDF data in attributes of a web page to:
  - transfer data from an application to another through the web;
  - write data only once for web users and web applications.
- ***JSON-LD*** - JavaScript Object Notation for Linked Data. Extension of JSON - simple property-value type machine readable data exchange format



RDFa

- ***RDFa 1.1*** is for XHTML and HTML5, also works for any XML-based languages like SVG. (You can use HTML+RDFa, but it won't be officially valid HTML file)
  - ***RDFa Lite 1.1*** - is a minimal subset of RDFa (http://www.w3.org/TR/rdfa-lite/)
  - ***RDFa 1.1 Prime*** – is Rich Structured Data Markup for Web Documents (http://www.w3.org/TR/rdfa-primer/)
  - ***RDFa Core 1.1*** – is complete specification of RDFa (http://www.w3.org/TR/2013/REC-rdfa-core-20130822/)
- Useful links:
  - Basic presentation: http://www.slideshare.net/fabien_gandon/rdfa-in-a-nutshell-v1
  - RDFa materials for users and developers: http://http://rdfa.info/ , http://rdfa.info/dev/
  - Real-time RDFa 1.1 editor: http://rdfa.info/play/
  - RDFa Online Parser: http://rdf-in-html.appspot.com/
  - RDFa 1.1 Distiller: http://www.w3.org/2012/pyRdfa/

RDFa

```
<div vocab=“http://schema.org/”
     prefix=“ex: http://example.com/”
     resource="ex:alice/posts/trouble_with_bob“
     typeof=“Article”>
   <h2 property="title">The trouble with Bob</h2>
    ...
   The trouble with Bob is that he takes much better photos than I do:
    ...
   <div resource="ex:bob/photos/sunset.jpg"
        prefix=“dc: http://purl.org/dc/terms/” >
      <img src="http://example.com/bob/photos/sunset.jpg" />
      <span property=“title">Beautiful Sunset</span>
      by <span property=“dc:creator">Bob</span>.
   </div>
</div>
```

The trouble with Bob

…
The trouble with Bob is that he takes much better photos than I do:
…

Beautiful Sunset by Bob

RDFa

```
<div vocab=“http://schema.org/”
     prefix=“ex: http://example.com/”
     resource="ex:alice/posts/trouble_with_bob“
     typeof=“Article”>
   <h2 property="title">The trouble with Bob</h2>
    ...
   The trouble with Bob is that he takes much better photos than I do:
    ...
   <div resource="ex:bob/photos/sunset.jpg"
        prefix=“dc: http://purl.org/dc/terms/” >
      <img src="http://example.com/bob/photos/sunset.jpg" />
      <span property=“title">Beautiful Sunset</span>
      by <span property=“dc:creator">Bob</span>.
   </div>
</div>
```

```
@prefix  sc: <http://schema.org/> .
@prefix  ex: <http://example.com/> .
@prefix  dc: < http://purl.org/dc/terms/> .

ex:alice/posts/trouble_with_bob a sc:Article; sc:title ”The trouble with Bob”.
ex:bob/photos/sunset.jpg sc:title ”Beautiful Sunset” ; dc:creator “Bob” .
```

RDFa Lite 1.1

- ***RDFa Lite*** consists of five simple attributes: ***vocab*** (default vocabulary, applied until be redefined), ***typeof***, ***property***, ***resource*** and ***prefix***.

```
<div vocab="http://schema.org/"
     prefix="ex: http://example.com/"
     resource="ex:alice/posts/trouble_with_bob"
     typeof="Article">
   <h2 property="title">The trouble with Bob</h2>
    ...
   The trouble with Bob is that he takes much better photos than I do:
    ...
   <div resource="ex:bob/photos/sunset.jpg"
        prefix="dc: http://purl.org/dc/terms/" >
      <img src="http://example.com/bob/photos/sunset.jpg" />
      <span property="title">Beautiful Sunset</span>
      by <span property="dc:creator">Bob</span>.
   </div>
</div>
```

- A full list of pre-declared prefixes: http://www.w3.org/2011/rdfa-context/rdfa-1.1

```
@prefix  sc: <http://schema.org/> .
@prefix  ex: <http://example.com/> .
@prefix  dc: < http://purl.org/dc/terms/> .

ex:alice/posts/trouble_with_bob a sc:Article; sc:title "The trouble with Bob".
ex:bob/photos/sunset.jpg sc:title "Beautiful Sunset" ; dc:creator "Bob" .
```

More of RDFa

- It is possible to define a *blank node* by named blank node (e.g. “*_:name*”) or just by mentioning a type of the node.
- If the element contains the *href* (or *src*) attribute, property is automatically associated with the value of the attribute rather than the textual content of the <*a*> element.

```
<div vocab="http://xmlns.com/foaf/0.1/">
   <ul>
      <li typeof="Person">
        <a property="homepage" href="http://example.com/bob/">Bob</a>
      </li>
      <li typeof="Person">
        <a property="homepage" href="http://example.com/eve/">Eve</a>
      </li>
   </ul>
</div>
```

```
<div vocab="http://xmlns.com/foaf/0.1/" typeof="Person">
 <span property="name">Alice Birpemswick</span>,
 <ul>
  <li property="knows" typeof="Person">
   <a property="homepage" href="http://example.com/bob/"><span property="name">Bob</span></a>
  </li>
  <li property="knows" typeof="Person">
   <a property="homepage" href="http://example.com/eve/"><span property="name">Eve</span></a>
  </li>
 </ul>
</div>
```

More of RDFa

- HTML+RDFa allows "*Property copying*" in case you have repeating set of data. It is possible to collect a number of statements as a pattern (***rdfa:Pattern***) and refer to it using the property ***rdfa:copy***.

```
<body vocab="http://purl.org/dc/terms/">
   <div resource="/alice/posts/trouble_with_bob">
      <h2 property="title">The trouble with Bob</h2>
      /some repeating part/
   </div>
   <div resource="/alice/posts/jims_concert">
      <h2 property="title">I was at Jim's concert the other day</h2>
      /some repeating part/
   </div>
</body>
```

```
<body vocab="http://purl.org/dc/terms/">
   <div resource="/alice/posts/trouble_with_bob">
      <h2 property="title">The trouble with Bob</h2>
      <link property="rdfa:copy" href="#cpattern"/>
   </div>
   <div resource="/alice/posts/jims_concert">
      <h2 property="title">I was at Jim's concert the other day</h2>
      <link property="rdfa:copy" href="#cpattern"/>
   </div>
   <div resource="#cpattern" typeof="rdfa:Pattern">
      /some repeating part/
   </div>
</body>
```

More of RDFa

- RDFa allows the value of a ***property*** and ***typeof*** attributes to be a list of values

```
...
<div resource="ex:me" typeof="foaf:Person schema:Person" >
   ...
   <h3 property="dc:creator schema:creator" resource="ex:me">John</h3>
</div>
...
```

- Human readability vs. unambiguity for machine readability (*RDFa Core*)

```
...
<p>Date: <span property="http://purl.org/dc/terms/created">2014-11-17</span></p>   ...
<p>Date: <span property="http://purl.org/dc/terms/created">17th of November, 2014</span></p>
...
```

  – RDFa makes it possible to re-use the ***content*** attribute of HTML

```
...
<p>Date: <span property="http://purl.org/dc/terms/created" content="2014-09-10">10th
of September, 2014</span></p>
...
```

  – also, ***content*** attribute can be useful when we define some statements through *meta* element (that may have no text content) in the header of the document

```
<head prefix="og: http://ogp.me/ns#" >
   ...
   <meta property="og:title" content="The Trouble with Bob" />
   ...
</head>
```

More of RDFa

- *RDFa Core* introduces attribute ***about*** that can be used as an alternative to ***resource*** in setting the context (the *subject* of the statement)
  - Attribute *resource* may be used to present subject or object of a statement

```
<div resource="/alice/posts/trouble">
   <h2 property="title">The trouble ...</h2>
   <h3 property="creator" resource="#me">Alice</h3>
</div>
```

Example: We need to set up a separate index page for all different blogs

```
<ul resource="/alice/posts/trouble">
  <li resource="/alice/posts/trouble" property="title">The trouble ...</li>
  <li resource="/alice/posts/jos" property="title">Jo’s Barbecue</li>
</ul>
```

The combination of *property* and *resource* inside the same element would be considered as *predicate* and *object* and would generate a different statement than originally intended.

```
<ul>
  <li resource="/alice/posts/trouble"><span property="title">The trouble ...</span></li>
  <li resource="/alice/posts/jos"><span property="title">Jo's Barbecue</span></li>
</ul>
```

… Current solution becomes a little bit complicated. Therefore ***about*** could be used…

```
<ul>
  <li about="/alice/posts/trouble" property="title">The trouble ...</li>
  <li about="/alice/posts/jos" property="title">Jo's Barbecue</li>
  ...
</ul>
```

More of RDFa

- *RDFa Core* allows definition of datatypes and language tag

```
<span property="dc:date" datatype="xsd:gYear">2011</span>
<span property="dc:name" xml:lang="en">John</span>
```

- *RDFa Core* attribute ***rel*** can be used as an alternative to ***property***
  – In contrast to *property*, *rel* *never* considers the textual content of an element (or the value of the *content* attribute). Instead, if no clear target has been specified for a link via, e.g., a *resource* or an *href* attribute, the processor is supposed to go “down” and find one or more targets in the hierarchy and use those.

```
<div vocab="http://xmlns.com/foaf/0.1/" resource="#me">
   <ul>
      <li property="knows" resource="http://example.com/bob/#me" typeof="Person">
         ...
      </li>
      <li property="knows" resource="http://example.com/eve/#me" typeof="Person">
         ...
      </li>
   </ul>
</div>
```

```
<div vocab="http://xmlns.com/foaf/0.1/" resource="#me">
   <ul rel="knows">
      <li resource="http://example.com/bob/#me" typeof="Person">
         ...
      </li>
      <li resource="http://example.com/eve/#me" typeof="Person">
         ...
      </li>
   </ul>
</div>
```

More of RDFa

```
<html xmlns="http://www.w3.org/1999/xhtml"
      prefix="dbp: http://dbpedia.org/property/"
      prefix="dbp-owl: http://dbpedia.org/ontology/"
      prefix="dbr: http://dbpedia.org/resource/"
      prefix="foaf: http://xmlns.com/foaf/0.1/"
      prefix="xsd: http://www.w3.org/2001/XMLSchema#">
  <head>
    <title>Albert Einstein</title>
  </head>
  <body>
    <div about="dbr:Albert_Einstein">
      <span property="foaf:name">Albert Einstein</span>
      <span property="dbp:dateOfBirth" datatype="xsd:date">1879-03-14</span>
      <div rel="dbp:birthPlace" resource="dbp-res:German_Empire">
        <span property="dbp:conventionalLongName">the German Empire</span>
        <span rel="dbp-owl:capital" resource="dbr:Berlin" />
      </div>
    </div>
  </body>
</html>
```

```
@prefix …

dbr:Albert_Einstein foaf:name "Albert Einstein" .
dbr:Albert_Einstein dbp:dateOfBirth "1879-03-14"^^xsd:date .
dbr:Albert_Einstein dbp:birthPlace dbr:German_Empire .

dbr:German_Empire dbp:conventionalLongName "the German Empire" .
dbr:German_Empire dbp-owl:capital dbr:Berlin .
```

JSON-LD

- ***JSON*** (JavaScript Object Notation) is a lightweight data-interchange format that is completely language independent but uses conventions that are familiar to programmers of most of the programming languages (an *object* is an unordered set of name/value pairs).
- ***JSON-LD*** is a lightweight Linked Data format that extends JSON:
  - easy for humans to read and write
  - it is based on the already successful JSON format
  - provides a way to help JSON data interoperate at Web-scale

```
{
  "@context": "http://json-ld.org/contexts/person.jsonld",
  "@id": "http://dbpedia.org/resource/John_Lennon",
  "name": "John Lennon",
  "born": "1940-10-09",
  "spouse": "http://dbpedia.org/resource/Cynthia_Lennon"
}
```

- JSON for Linked Data: http://json-ld.org
- JSON-LD Playground: http://json-ld.org/playground/index.html
- Latest specification list: http://json-ld.org/spec/latest/
- Basic specification: www.w3.org/TR/json-ld/
- JSON-LD parser/serializer: https://github.com/RDFLib/rdflib-jsonld
- JSON-LD Processor and API implementation in JavaScript: https://npmjs.org/package/jsonld

JSON-LD

- Ambiguity

```
{    "name":"Oleksiy",
     "homepage":"http://users.jyu.fi/~olkhriye" }
```

```
{    "name":"olkhriye",
     "homepage":"http://users.jyu.fi/~olkhriye" }
```

- To be specific

```
{    "http://ex1.com/name":"Oleksiy",
     "http://ex1.com/homepage":"http://users.jyu.fi/~olkhriye" }
```

- To be very concise use ***JSON-LD Context***
  - used to define the short-hand names

```
{    "@Context":"http://ex1.com/myApp.jsonld",
     "name":"Oleksiy",
     "homepage":"http://users.jyu.fi/~olkhriye" }
```

```
{    "@Context":{ "schema":"http://schema.org/",
                  "name":"schema:givenName",
                  "homepage":"schema:url" },
     "name":"Oleksiy",
     "homepage":"http://users.jyu.fi/~olkhriye" }
```

JSON-LD

- ***JSON-LD Identifiers***
  - uniquely identifies *things*

```
{   “@Context” {
       “name”:“http://schema.org/givenName”,
       “homepage”:“http://schema.org/url”
    },
    “@id”:“http://people.com/OleksiyKhriyenko”,
    “name”:“Oleksiy”,
    “homepage”:{“@id”:“http://users.jyu.fi/~olkhriye”}
}
```

```
{   “@Context” {
       “name”:“http://schema.org/givenName”,
       “homepage”:{
          “@id”:“http://schema.org/url”,
          “@type”:“@id” }
    },
    “@id”:“http://people.com/OleksiyKhriyenko”,
    “name”:“Oleksiy”,
    “homepage”:“http://users.jyu.fi/~olkhriye”
}
```

JSON-LD

- ***JSON-LD Type***
  - sets the data type of a *node* or *typed value*

```
{   “@Context” {
       “name”:“http://schema.org/givenName”,
       “homepage”:{“@id”:“http://schema.org/url”,“@type”:“@id”}
    },
    “@id”:“http://people.com/OleksiyKhriyenko”,
    “@type”:“http://schema.org/Person”,
    “name”:“Oleksiy”,
    “homepage”:“http://users.jyu.fi/~olkhriye”
}
```

- ***JSON-LD Value***
  - specifies the data that is associated with a particular *property* in the graph

```
{   “@Context”:“http://context-jsonld.com/person”,
    “@id”:“http://people.com/OleksiyKhriyenko”,
    “@type”:“http://schema.org/Person”,
    “name”:[
             {“@value”:“Oleksiy”},
             …
           ] ,
    “birthDate”:{“@value”:“1981-08-13”, “@type”:“xsd:data”}
}
```

JSON-LD

- ***JSON-LD language***
  - specifies the language for a particular string value or the default language of a JSON-LD document

```
{  “@Context” {
      “schema”:“http://schema.org/”,
      “name_ua”:{“@id”:“schema:givenName”, “@language”:“ua”},
      “name_ru”:{“@id”:“schema:givenName”, “@language”:“ru”},
      “birthDate”:{“@id”:“schema:birthData”, “@type”:“xsd:date”}
   },
   “@id”:“http://people.com/OleksiyKhriyenko”,
   “@type”:“http://schema.org/Person”,
   “name_ua”:“Oleksiy”,
   “name_ru”:“Aleksey”,
   “birthDate”:“1981-08-13”
}
```

JSON-LD

- ***JSON-LD arrays***
  - ***@container*** is used to set the default container type for a *term*
  - ***@list*** represents *ordered* collection of values
  - ***@set*** describes *unordered* set of values

```
{  "@context":{ ...
                "nick":{ "@id": "http://xmlns.com/foaf/0.1/nick",
                         "@container": "@list“ },
                "name":{ "@id": "http://xmlns.com/foaf/0.1/name",
                         "@container": "@set“ }
              },
   "@id": "http://example.org/people#joebob",
   "nick": [ "joe", "bob", "jaybee" ],
   "name": [ { “@value”:“John”,
               “@language”:“en” },
             { “@value”:“Jonie”,
               “@language”:“fr” }
           ],
   “homepage”: [ “http://users.jyu.fi/~joe”,
                 “http://examplepage.com/~bob”,
                 “http://myPage.org/~jaybee”
               ],
   ...
}
```

JSON-LD

- ***JSON-LD reverse property*** *allows bidirection in directed graph*

```
[ { "@id": "#john",
    "http://myontology.com/onto#name": “John“ },
  { "@id": "#peter",
    "http://myontology.com/onto#name": “Peter",
    "http://myontology.com/onto#parent": { "@id": "#John" } },
  { "@id": "#mary",
    "http://myontology.com/onto#name": “Mary",
    "http://myontology.com/onto#parent": { "@id": "#John" } }
]
```

```
{ "@id": "#john",
  "http://myontology.com/onto#name": “John",
  "@reverse": { "http://myontology.com/onto#parent": [
                    { "@id": "#peter",
                      "http://myontology.com/onto#name": “Peter" },
                    { "@id": "#mary",
                      "http://myontology.com/onto#name": “Mary" } ] }
}
```

```
{ "@context": { "name": "http://myontology.com/onto#name",
                "children": { "@reverse": "http://myontology.com/onto#parent" } },
  "@id": "#john",
  "name": “John",
  "children": [ { "@id": "#peter", "name": “Peter" },
                { "@id": "#mary", "name": “Mary" } ]
}
```

JSON-LD

- **@base** sets the base IRI against which *relative IRIs* are resolved

http://myJSON.com/document.jsonld

```
{ "@context": {
                  "label": "http://www.w3.org/2000/01/rdf-schema#label"
              },
  "@id": "",
  "label": "A simple document"
}
```

```
{ "@context": {   "@base": "http://myJSON.com/document.jsonld"   },
  "@id": "",
  "label": "A simple document"
}
```

- **@vocab** expands *properties* and *values* in `@type` with a common prefix IRI.

If certain keys *should not be expanded* using the vocabulary IRI, a term can be explicitly set to *null* in the context.

```
{ "@context": {
                  "@vocab": "http://schema.org/",
                  "dbID": null
              },
  "@id": "http://example.org/places#SalsaOrchidea",
  "@type": "Restaurant",
  "name": "Salsa Orchidea",
  "dbID": "12345678"
}
```

JSON-LD

- **`JSON-LD graph`** - used to group a set of *nodes*. make statements about a *graph* itself, rather than just a single *node*.

```
{ "@context": { "generatedAt": {
                    "@id": "http://www.w3.org/ns/prov#generatedAtTime",
                    "@type": "http://www.w3.org/2001/XMLSchema#date"
                 }
              },
  "@id": "http://example.org/graphs/13",
  "generatedAt": "2013-11-26",
  "@graph": [ { "@id": "http://example.org/about#manu",
                ... },
              { "@id": "http://http://example.org/foaf#me",
                ... }
            ]
}
```

- Explicit expression of default graph

```
{ "@context": ...,
  "@graph": [ {... },
              {... } ]
}
```

```
[ { "@context": ...,
    ... },
  { "@context": ...,
    ... } ]
```

JSON-LD

- ***JSON-LD compact IRI*** - expressing an IRI using a *prefix* and *suffix* separated by a colon (:):
- *prefix* matches a term defined within the *active context;*
- *suffix* does not begin with two slashes (//);
- if the *prefix* is not defined in the active context, or the *suffix* begins with two slashes (e.g. *http://example.com*), the value is interpreted as *absolute IRI* instead;
- if the *prefix* is an underscore (_), the value is interpreted as *blank node* identifier instead.

```
{
  "@context":
  {
    "foaf": "http://xmlns.com/foaf/0.1/"
  ...
  },
  "@type": "foaf:Person",
  "foaf:name": "Dave Longley",
  ...
}
```

JSON-LD and RDF

- *JSON-LD*

```
{   “@Context”:{
                  “schema”:“http://schema.org/”
               },
    “@id”:“http://people.com/OleksiyKhriyenko”,
    “@type”:“schema:Person”,
    “schema:name”:“Oleksiy”,
    “schema:knows”:{
          “@id”:“http://people.com/JohnDou”,
          “@type”:“schema:Person”,
          “schema:name”:“John”,
          “schema:knows”:“http://people.com/OleksiyKhriyenko”
     }
}
```

- *RDF*

```
@prefix schema: <http://schema.org/>.
<http://people.com/OleksiyKhriyenko> a schema:Person ;
          schema:name “Oleksiy” ;
          schema:knows <http://people.com/JohnDou> .
<http://people.com/JohnDou> a schema:Person ;
          schema:name “John” ;
          schema:knows <http://people.com/OleksiyKhriyenko> .
```

JSON-LD and HTML

- ***JSON-LD*** content can be easily embedded in HTML by placing it in a script element with the *type* attribute set to *application/ld+json*.

```
<script type="application/id+json">
{
    "@Context":"http://context-jsonld.com/person",
    "@id":"http://people.com/OleksiyKhriyenko",
    "@type":"Person",
    "name":"Oleksiy",
    "knows":{
              "@id":"http://people.com/JohnDou",
              "@type":"Person",
              "name":"John",
              "knows":"http://people.com/OleksiyKhriyenko"
            }
}
</script>
```

JSON-LD and RDFa

```
<div prefix="foaf: http://xmlns.com/foaf/0.1/">
   <ul>
      <li typeof="foaf:Person">
        <a rel="foaf:homepage" href="http://example.com/bob/"
                                            property="foaf:name">Bob</a>
      </li>
      <li typeof="foaf:Person">
        <a rel="foaf:homepage" href="http://example.com/eve/"
                                            property="foaf:name">Eve</a>
      </li>
      <li typeof="foaf:Person">
        <a rel="foaf:homepage" href="http://example.com/manu/"
                                            property="foaf:name">Manu</a>
      </li>
   </ul>
</div>
```

```
{ "@context":{ "foaf": "http://xmlns.com/foaf/0.1/" },
  "@graph":[ { "@type": "foaf:Person",
               "foaf:homepage": "http://example.com/bob/",
               "foaf:name": "Bob" },
             { "@type": "foaf:Person",
               "foaf:homepage": "http://example.com/eve/",
               "foaf:name": "Eve" },
             { "@type": "foaf:Person",
               "foaf:homepage": "http://example.com/manu/",
               "foaf:name": "Manu"}
           ]
}
```

Homework

- Using Protégé 4.x (5.x) and ontology (created in the previous homework), write SWRL rules and run a reasoner
- Download CWM and try it out
- Create a simple html web page about yourself
  - Use RDFa with XHTML 1.1 to provide some machine-readable information about yourself. You can use Dublin Core or FOAF
  - Create a JSON-LD file to transfer the same data