Wikidata: Temporal qualifiers/data quality

- quality is better than DBpedia, but not perfect:

```
SELECT *
WHERE
{ ?uni rdfs:label ?val .
  ?uni wdt:P31/wdt:P279* wd:Q3918 .     ## Q3918: University
  ?uni p:P17 ?stmt .       ## P17: located in country
    ?stmt ps:P17 ?C ;
          optional { ?stmt pq:P580 ?start }
          optional { ?stmt pq:P582 ?end } .
  ?C rdfs:label ?cl
  filter (lang(?cl)="en")
  filter (lang(?val)="en")
  filter (str(?val) = "Moscow State University")
}
```

[Filename: RDF/lomonossow.sparql]

- Countries: Russia, Soviet Union, Russian Empire
  the “start” and “end” entries are often missing.

Chapter 9
Ontologies and the Web Ontology Language – OWL

- *vocabularies* can be defined by RDFS
  - not so much stronger than the ER Model or UML
    (even weaker: no cardinalities)
  - not only a conceptual model, but a “real language” with a close connection to the data level (RDF)
  - *incremental* world-wide approach
  - “global” vocabulary can be defined by autonomous partners
- but: still restricted when *describing* the vocabulary.

Ontologies/ontology languages further extend the expressiveness:

- Description Logics
- Topic Maps (in SGML) since early 90s, XTM (XML Topic Maps)
- Ontolingua – non-XML approach from the Knowledge Representation area
- OIL (Ontology Inference Layer): initiative funded by the EU programme for Information Society Technologies (project: On-To-Knowledge, 1.2000-10.2002); based on RDF/RDFS
- DAML (Darpa Agent Markup Language; 2000) ... first ideas for a Semantic Web language
- DAML+OIL (Jan. 2001)
- developed into OWL (1st version March 02, finalized Feb. 04)

Three Variants of OWL

Several expressiveness/complexity/decidability levels:

- OWL Full: extension of RDF/RDFS
  - classes can also be regarded as individuals (have properties, classes of classes etc.)
- OWL DL
  - fragment of OWL that fits into the Description Logics Framework:
    - ⋆ the sets of classes, properties, individuals and literals are disjoint
    - ⇒ only individuals can have arbitrary user-specified properties;
      classes and properties have only properties from the predefined RDFS and OWL vocabularies.
  - decidable reasoning
  - OWL 1.0 (2004), OWL 2.0 (2009)
- OWL Lite
  - subset of OWL DL
  - easier migration from frame-based tools (note: F-Logic is a frame-based framework)
  - easier reasoning (translation to Datalog)

9.1 Description Logics

- Focus on the description of *concepts*, not of instances
- Terminological Reasoning
- Origin of DLs: Semantic Networks (graphical formalism)

Notions

- Concepts (= classes),
  note: literal datatypes (string, integer etc.) are not classes in DL and OWL, but *data ranges*
  (cf. XML Schema: distinction between simpleTypes and complexTypes)
- Roles (= relationships),
- A Description Logic alphabet consists of a finite set of concept names (e.g. Person, Cat, LivingBeing, Male, Female, ...) and a finite set of role names (e.g., hasChild, marriedTo, ...),
- constructors for derived concepts and roles,
- axioms for asserting facts about concepts and roles.

Comparison with other Logics

Syntax and semantics defined different but similar from first-order logic

- formulas over an alphabet and a small set of additional symbols and combinators
- semantics defined via *interpretations* of the combinators
- set-oriented, no instance variables
  (FOL: instance-oriented with domain quantifiers)
- family of languages depending on what combinators are allowed.

The base: $\mathcal{AL}$

The usual starting point is $\mathcal{AL}$:

- "attributive language"
- Manfred Schmidt-Schauss and Gert Smolka: *Attributive Concept Descriptions with Complements*. In *Artificial Intelligence* 48(1), 1991, pp. 1–26.
- extensions (see later: $\mathcal{ALC}$, $\mathcal{ALCQ}$, $\mathcal{ALCQ}(D)$, $\mathcal{ALCQI}$, $\mathcal{ALCN}$ etc.)

ATOMIC, NAMED CONCEPTS

- atomic concepts, e.g., Person, Male, Female
- the "universal concept" $\top$ (often called "Thing" – everything is an instance of Thing)
- the empty concept $\bot$ ("Nothing"). There is no thing that is an instance of $\bot$.

CONCEPT EXPRESSIONS USING SET OPERATORS

- intersection of concepts: $A \sqcap B$
  Adult $\sqcap$ Male
- negation: $\neg A$
  $\neg$Italian , Person $\sqcap$ $\neg$Italian
- union (disjunctive concept): $A \sqcup B$
  Cat $\sqcup$ Dog – things where it is known that they are cats or dogs, but not necessarily which one.

CONCEPT EXPRESSIONS USING ROLES

Concepts (as an intensional characterization of sets of instances) can be described implicitly by their properties (wrt. *roles*).

Let R be a role, C a concept. Then, the expressions $\exists R.C$ and $\forall R.C$ also describe concepts (intensionally defined concepts) by constraining the roles:

- Existential quantification: $\exists R.C$ – all things that have a *filler* for the role R that is in C.
  $\exists$hasChild.Male means "all things that have a male child".
  Syntax: the whole expression is the "concept expression", i.e., $\exists$hasChild.Male(john) stands for ($\exists$hasChild.Male)(john).
- Range constraints: $\forall R.C$
  $\forall$hasChild.Male means "all things that have only male children (including those that have no children at all)".
- Note that $\bot$ can be used to express non-existence: $\forall R.\bot$: all things where all fillers of role R are of the concept $\bot$ (= Nothing) – i.e., all things that do not have a filler for the role R.
  $\forall$hasChild.$\bot$ means "all things that have no children".

SEMANTICS OF CONCEPT CONSTRUCTORS

As usual: by interpretations.

An interpretation $\mathcal{I} = (\mathcal{I}, \mathcal{D})$ consists of the following:

- a domain $\mathcal{D}$,
- for every concept name C: $I(C) \subseteq \mathcal{D}$ is a subset of the domain,
- for every role name R: $I(R) \subseteq \mathcal{D} \times \mathcal{D}$ is a binary relation over the domain.

Structural Induction

- $I(A \sqcup B) = I(A) \cup I(B)$
- $I(A \sqcap B) = I(A) \cap I(B)$
- $I(\neg A) = \mathcal{D} \setminus I(A)$
- $I(\exists R.C) = \{x \mid$ there is an y such that $(x, y) \in I(R)$ and $y \in I(C)\}$
- $I(\forall R.C) = I(\neg\exists R.(\neg C)) = \{x \mid$ for all y such that $(x, y) \in I(R)$, $y \in I(C)\}$

Example

Male $\sqcap$ $\forall$hasChild.Male is the set of all men who have only sons.

STRUCTURE OF A DL KNOWLEDGE BASE

DL Knowledge Base

TBox (schema)

Statements/Axioms about concepts

Man $\equiv$ Person $\sqcap$ Male

Parent $\equiv$ Person $\sqcap$ ($\exists \geq 1$ hasChild.$\top$)

ParentOfSons $\equiv$ Person $\sqcap$ ($\exists \geq 1$ hasChild.Male)

ParentOfOnlySons $\equiv$ Person $\sqcap$ ($\forall$ hasChild.Male)

ABox (data)

Statements/Facts about individuals

Person(john), (Adult $\sqcap$ Male)(john), ($\neg$Italian)(john)

hasChild(john,alice), age(alice,10), Female(alice)

hasChild(john,bob), age(bob,8), Male(bob)

$\forall$hasChild.$\bot$ (alice), $\neg\exists$hasChild.$\top$ (bob)

THE TBOX: TERMINOLOGICAL AXIOMS

Definitions and assertions (to be understood as constraints, and as knowledge that can be used for deduction, e.g. of class membership) about concepts:

- concept subsumption: $C \sqsubseteq D$; defining a concept hierarchy.
  $\mathcal{I} \models C \sqsubseteq D \;\; :\Leftrightarrow \;\; I(C) \subseteq I(D)$.
- concept equivalence: $C \equiv D$; often used for defining the left-hand side concept.
  Semantics: $\mathcal{I} \models C \equiv D \;\; :\Leftrightarrow \;\; C \sqsubseteq D$ and $D \sqsubseteq C$.

TBox Reasoning

- is a concept C satisfiable?
- is $C \sqsubseteq D$ implied by a TBox
- given the definition of a new concept D, classify it wrt. the given concept hierarchy.

THE ABOX: ASSERTIONAL AXIOMS

- contains the facts about instances (using names for the instances) in terms of the basic concepts and roles:
  Person(john), Male(john), hasChild(john,alice)
- contains also knowledge in terms of intensional concepts, e.g., ∃hasChild.Male(john)

TBox + ABox Reasoning

- check consistency between ABox and a given TBox
- ask whether a given instance satisfies a concept C
- ask for all instances that have a given property
- ask for the most specific concepts that an instance satisfies

Note: instances are allowed only in the ABox, not in the TBox.

If instances should be used in the definition of concepts (e.g., “European Country” or “Italian City”), *Nominals* must be used (see later).

THE BASIS: $\mathcal{AL}$

Concept expressions can be composed as follows:

- intersection of concepts, negation of *atomic* concepts: $C \sqcap D$, $\neg A$
- restricted existential quantification: $\exists R.\top$
  $\exists$hasChild.$\top$ means "all things that have a child (... that belongs to the concept "Thing")".
- universal restriction: $\forall R.C$
  $\forall$hasChild.Person means "if some thing is a "filler" of a "hasChild" role, (of another thing), it must be a person."

Properties of $\mathcal{AL}$

- $\mathcal{AL}$ has no "branching" in its tableaux (no union, or any kind of disjunction); so proofs in $\mathcal{AL}$ are linear.
- Note that all notions of RDFS can be expressed already in $\mathcal{AL}$:
  - C rdfs:subClassOf D: $C \sqsubseteq D$
  - p rdfs:domain C: $\exists p.\top \sqsubseteq C$
  - p rdfs:subPropertyOf q: $p \sqsubseteq q$
  - p rdfs:range C: $\top \sqsubseteq \forall p.C$
  - where C and D can be composite concept expressions over the $\mathcal{AL}$ constructors listed above.

FAMILY OF DL LANGUAGES UP TO $\mathcal{ALC}$

- $\mathcal{U}$: "union"; e.g. Parent $\equiv$ Father $\sqcup$ Mother.
- $\mathcal{C}$: negation ("complement") of non-atomic concepts.
  Childless $\equiv$ Person $\sqcap$ $\neg\exists$hasChild.$\top$ characterizes the set of persons who have no children (note: open-world semantics of negation!)
  Note: the FOL equivalent would be expressed via variables:
  $\forall x$(Childless(x) $\leftrightarrow$ (Person(x) $\wedge$ $\neg\exists y$(hasChild(x, y))))
- $\mathcal{U}$ and $\mathcal{E}$ can be expressed by $\mathcal{C}$.
- $\mathcal{ALC}$ is the "smallest" Description Logic that is closed wrt. the set operations.
- A frequently used restriction of $\mathcal{AL}$ is called $\mathcal{FL}^-$ (for "Frame-Language"), which is obtained by disallowing negation completely (i.e., having only positive knowledge).

Family of DL Languages: Extensions to $\mathcal{ALC}$

- $\mathcal{E}$: (unrestricted) existential quantification of the form $\exists R.C$ (recall that $\mathcal{AL}$ allows only $\exists R.\top$).
  HasSon $\equiv$ $\exists$hasChild.Male , for persons who have at least one male child,
  GrandParent $\equiv$ $\exists$hasChild.(hasChild.$\top$) for grandparents.
  Note: the FOL equivalent uses variables:
  hasSon$(x) \leftrightarrow \exists y($hasChild$(x,y) \wedge$ Male$(y))$,
  grandparent$(x) \leftrightarrow \exists y($hasChild$(x,y) \wedge \exists x :$ hasChild$(y,x))$.
- Exercise: show why unrestricted existential quantification $\exists R.C$ in contrast to $\exists R.\top$ leads to branching.
- $\mathcal{N}$: (unqualified) cardinalities of roles ("number restrictions").
  $(\geq 3$ hasChild.$\top)$ for persons who have at least 3 children.
- $\mathcal{Q}$: qualified role restrictions:
  $(\leq 2$ hasChild.Male$)$
  $\mathcal{F}$: like $\mathcal{Q}$, but restricted to cardinalites 0, 1 and "arbitrary".

Complexity and Decidability: Overview

- Logic $\mathcal{L}^2$, i.e., FOL with only two (reusable) variable symbols is decidable.
- Full FOL is undecidable.
- DLs: incremental, modular set of semantical notions.
- only part of FOL is required for concept reasoning.
- $\mathcal{ALC}$ can be *expressed* by FOL, but then, the inherent semantics is lost $\rightarrow$ full FOL reasoner required.
- Actually, $\mathcal{ALC}$ can be encoded in FOL by only using two variables $\rightarrow$ $\mathcal{ALC}$ is decidable.
- Consistency checking of $\mathcal{ALC}$-TBoxes and -ABoxes is PSPACE-complete (proof by reduction to *Propositional Dynamic Logic* which is in turn a special case of propositional multimodal logics).
  There are algorithms that are efficient in the average case.
- $\mathcal{ALCN}$ goes beyond $\mathcal{L}^2$ and PSPACE. Reduction to $\mathcal{C}^2$ (including "counting" quantifiers) yields decidability, but now in NEXPTIME. There are algorithms for $\mathcal{ALCN}$ and even $\mathcal{ALCQ}$ in PSPACE.

Further Extensions

- Role hierarchy ($\mathcal{H}$; role subsumption and role equivalence, union/intersection of roles): hasSon $\sqsubseteq$ hasChild , hasChild $\equiv$ hasSon $\sqcup$ hasDaughter
- *Role Constructors* similar to regular expressions: concatenation (hasGrandchild $\equiv$ hasChild $\circ$ hasChild), transitive closure (hasDescendant $\equiv$ hasChild$^+$) (indicated by e.g. $\mathcal{H}_{reg}$ or $\mathcal{R}$), and inverse (isChildOf $\equiv$ hasChild$^-$) ($\mathcal{I}$).
- *Data types* (indicated by "(D)"), e.g. integers. Adult $\equiv$ Person $\sqcap$ $\exists$age. ≥ 18.
- *Nominals* ($\mathcal{O}$) allow to use individuals from the ABox also in the TBox.
  Enumeration Concepts: BeNeLux $\equiv$ {Belgium, Netherlands, Luxemburg},
  HasValue-Concepts: GermanCity $\equiv$ $\exists$inCountry.Germany.
- Role-Value-Maps:
  Equality Role-Value-Map: $(R_1 \equiv R_2)(x) \Leftrightarrow \forall y : R_1(x,y) \leftrightarrow R_2(x,y)$.
  Containment Role-Value-Map: $(R_1 \sqsubseteq R_2)(x) \equiv \forall y : R_1(x,y) \rightarrow R_2(x,y)$.
  (knows $\sqsubseteq$ likes) describes the set of people who like all people they know; i.e., (knows $\sqsubseteq$ likes)(john) denotes that John likes all people he knows.

Formal Semantics of Expressions

- $I(\geq nR.C) = \{x \mid \#\{y \mid (x,y) \in I(R)$ and $y \in I(C)\} \geq n\}$,
- $I(\leq nR.C) = \{x \mid \#\{y \mid (x,y) \in I(R)$ and $y \in I(C)\} \leq n\}$,
- $I(nR.C) = \{x \mid \#\{y \mid (x,y) \in I(R)$ and $y \in I(C)\} = n\}$,
- $I(R \sqcup S) = I(R) \cup I(S)$, $I(R \sqcap S) = I(R) \cap I(S)$,
- $I(R \circ S) = \{(x,z) \mid \exists y : (x,y) \in I(R)$ and $(y,z) \in I(S)\}$,
- $I(R^-) = \{(y,x) \mid (x,y) \in I(R)\}$,
- $I(R^+) = (I(R))^+$.
- If nominals are used, $\mathcal{I}$ also assigns an element $I(x) \in \mathcal{D}$ to each nominal symbol x (similar to constant symbols in FOL). With this,
  $I(\{x_1,\ldots,x_n\}) = \{I(x_1),\ldots,I(x_n)\}$, and
  $I(R.y) = \{x \mid \{z \mid (x,z) \in I(R)\} = \{y\}\}$,
- $I(R_1 \equiv R_2) = \{x \mid \forall y : R_1(x,y) \leftrightarrow R_2(x,y)\}$,
  $I(R_1 \sqsubseteq R_2) = \{x \mid \forall y : R_1(x,y) \rightarrow R_2(x,y)\}$.

Overview: Complexity of Extensions

- $\mathcal{ALC}_{reg}$, $\mathcal{ALCHIQ}_{\mathcal{R}^+}$, $\mathcal{ALCIO}$ are ExpTime-complete, $\mathcal{ALCHIQO}_{\mathcal{R}^+}$ is NExpTime-Complete.,
- Combining *composite* roles with cardinalities becomes undecidable (encoding in FOL requires 3 variables).
- Encoding of Role-Value Maps with composite roles in FOL is undecidable (encoding in FOL requires 3 variables; the logic loses the *tree model property*).
- $\mathcal{ALCQI}_{reg}$ with role-value maps restricted to boolean compositions of *basic* roles remains decidable. Decidability is also preserved when role-value-maps are restricted to functional roles.

Description Logic Model Theory

The definition is the same as in FOL:

- an interpretation is a model of an ABox A if
  - for every atomic concept C and individual x such that $C(x) \in A$, $I(x) \in I(C)$, and
  - for every atomic role R and individuals x, y such that $R(x,y) \in A$, $(I(x), I(y)) \in I(R)$.
- note: the interpretation of the non-atomic concepts and roles is given as before,
- all axioms ϕ of the TBox are satisfied, i.e., $\mathcal{I} \models \phi$.

Based on this, DL entailment is also defined as before:

- a set Φ of formulas entails another formula Ψ (denoted by $\Phi \models \psi$), if $\mathcal{I}(\Psi) =$ true in all models $\mathcal{I}$ of Φ.

DECIDABILITY, COMPLEXITY, AND ALGORITHMS

Many DLs are decidable, but in high complexity classes.

- decidability is due to the fact that often *local* properties are considered, and the verification proceeds tree-like through the graph without connections between the branches.
- This locality does not hold for cardinalities over composite roles, and for role-value maps – these lead to undecidability.
- Reasoning algorithms for $\mathcal{ALC}$ and many extensions are based on tableau algorithms, some use model checking (finite models), others use tree automata.

Three types of Algorithms

- restricted (to polynomial languages) and complete
- expressive logics with complete, worst-case EXPTIME algorithms that solve realistic problems in "reasonable" time. (Fact, HermiT, Racer, Pellet)
- more expressive logics with incomplete reasoning.

EXAMPLE

- Given facts: Person $\equiv$ Male $\sqcup$ Female and Person(unknownPerson).
- Query ?-Male(X) yields an empty answer
- Query ?-Female(X) yields an empty answer
- Query ?-(Male $\sqcup$ Female)(X) yields unknownPerson as an answer
- for query answering, *all* models of the TBox+ABox are considered.
- in some models, the unknownPerson is Male, in the others it is female.
- in all models it is in (Male $\sqcup$ Female).

Summary and Comparison with FOL

Base Data (DL atomic concepts and atomic roles $\sim$ RDF)

- unary predicates (concepts/classes): Person(john),
- binary predicates (roles/properties): hasChild(john,alice)

Expressions

Concept/Role Expressions act as unary/binary predicates:

- ($\exists$ hasChild.Male)(john), (Adult $\sqcap$ Parent)(john),
- (hasChild $\circ$ hasChild)(jack,alice), (neighbor*)(portugal,germany)

$\Rightarrow$ disjunction, conjunction and quantifiers *only* in the restricted contexts of expressions

$\Rightarrow$ implications *only* in the restricted contexts of TBox Axioms:

- $C_1 \sqsubseteq C_2$ Parent $\sqsubseteq$ Person
- $C_1 \equiv C_2$ Parent $\equiv$ $\exists$hasChild.$\top$
- $R_1 \sqsubseteq R_2$ capital $\sqsubseteq$ hasCity
- $R_1 \equiv R_2$ neighbor $\equiv$ (neighbor $\sqcup$ neighbor$^-$)

$\Rightarrow$ ABox/TBox (=database) is a conjunctive set of atoms.

$\Rightarrow$ No formulas with $\wedge$, $\vee$, $\neg$, $\forall x$, $\exists x$!

9.2 OWL

- the OWL versions use certain DL semantics:
- Base: $\mathcal{ALC}_{\mathcal{R}^+}$: (i.e., with transitive roles). This logic is called $\mathcal{S}$ (reminiscent to its similarity to the modal logic S).
- roles can be ordered hierarchically (rdfs:subPropertyOf; $\mathcal{H}$).
- OWL Lite: $\mathcal{SHIF}(D)$, Reasoning in ExpTime.
- OWL DL: $\mathcal{SHOIN}(D)$, decidable.
  Pellet (2007) implements $\mathcal{SHOIQ}(D)$. Decidability is in NExpTime (combined complexity wrt. TBox+ABox), but the actual complexity of a given task is constrained by the maximal used cardinality and use of nominals and inverses and behaves like the simpler classes.
  (Ian Horrocks and Ulrike Sattler: A Tableau Decision Procedure for SHOIQ(D); In IJCAI, 2005, pp. 448-453; available via `http://dblp.uni-trier.de`)
- OWL 2.0 towards $\mathcal{SROIQ}(D)$ and more datatypes ...

OWL Notions; OWL-DL vs. RDF/RDFS; Model vs. Graph

- OWL is defined based on (Description Logics) model theory,
- OWL ontologies can be represented by RDF graphs,
- Only *certain* RDF graphs are allowed OWL-DL ontologies: those, where class names, property names, individuals etc. occur in a well-organized way.
- Reasoning works on the (Description Logic) model, the RDF graph is only a means to represent it.
  (recall: RDF/RDFS "reasoning" works on the graph level)

OWL Vocabularies

- An OWL-DL vocabulary $\mathcal{V}$ is a 7-tuple (= a sorted vocabulary)
  $\mathcal{V} = (\mathcal{V}_{cls}, \mathcal{V}_{objprop}, \mathcal{V}_{dtprop}, \mathcal{V}_{annprop}, \mathcal{V}_{indiv}, \mathcal{V}_{DT}, \mathcal{V}_{lit})$:
- $\mathcal{V}_{cls}$ is the set of URIs denoting class names,
  `<http://.../mondial/10/meta#Country>`
- $\mathcal{V}_{objprop}$ is the set of URIs denoting object property names,
  `<http://.../mondial/10/meta#capital>`
- $\mathcal{V}_{dtprop}$ is the set of URIs denoting datatype property names,
  `<http://.../mondial/10/meta#population>`
- ($\mathcal{V}_{annprop}$ is the set of URIs denoting annotation property names,)
- $\mathcal{V}_{indiv}$ is the set of URIs denoting individuals, `<http://.../mondial/10/countries/D>`
- $\mathcal{V}_{DT}$ is the set of URIs denoting datatype names,
  `<http://www.w3.org/2001/XMLSchema#int>`
- $\mathcal{V}_{lit}$ is the set of literals;
- the builtin notions (=URIs) from RDF, RDFS, OWL namespaces do not belong to the vocabulary of the ontology (they are only used for describing the ontology in RDF).

OWL INTERPRETATIONS

Since DL is a subset of FOL, the interpretation of an OWL-DL vocabulary can be given as a FOL interpretation

$$\mathcal{I} = (I_{indiv} \cup I_{cls} \cup I_{objprop} \cup I_{dtprop} \cup I_{annprop} \cup I_{DT}\ ,\ \mathcal{U}_{obj} \cup \mathcal{U}_{DT})$$

where I interprets the vocabulary as

- I_{indiv} constant symbols (individuals),
- I_{cls}, I_{DT} unary predicates (classes and datatypes),
- $I_{objprop}$, I_{dtprop}, $I_{annprop}$ binary predicates (properties),

and the universe $\mathcal{U}$ is partitioned into

- an *object domain* $\mathcal{U}_{obj}$
- and a data domain $\mathcal{U}_{DT}$ (of all values of datatypes).

OWL INTERPRETATIONS

The interpretation I is as follows:

I_{indiv}: each individual $a \in \mathcal{V}_{indiv}$ to an object $I(a) \in \mathcal{U}_{obj}$,
(e.g., $I(\texttt{<http://.../mondial/10/countries/D>}) =$ *germany*)

I_{cls}: each class $C \in \mathcal{V}_{cls}$ to a set $I(C) \subseteq \mathcal{U}_{obj}$,
(e.g., *germany* $\in I(\texttt{<http://.../mondial/10/meta\#Country>})$)

I_{DT}: each datatype $D \in \mathcal{V}_{DT}$ to a set $I(D) \subseteq \mathcal{U}_{DT}$,
(e.g., $I(\texttt{<http://www.w3.org/2001/XMLSchema\#int>}) = \{\ldots, -2, -1, 0, 1, 2, \ldots\}$)

$I_{objprop}$: each object property $p \in \mathcal{V}_{objprop}$ to a binary relation $I(p) \subseteq \mathcal{U}_{obj} \times \mathcal{U}_{obj}$,
(e.g., (*germany*, *berlin*) $\in I(\texttt{<http://.../mondial/10/meta\#capital>})$)

I_{dtprop}: each datatype property $p \in \mathcal{V}_{dtprop}$ to a binary relation $I(p) \subseteq \mathcal{U}_{obj} \times \mathcal{U}_{D}$,
(e.g., (*germany*, $83536115) \in I(\texttt{<http://.../mondial/10/meta\#population>})$)

$I_{annprop}$: each annotation property $p \in \mathcal{V}_{annprop}$ to a binary relation $I(p) \subseteq \mathcal{U} \times \mathcal{U}$.

OWL Class Definitions and Axioms (Overview)

- owl:Class
- The properties of an owl:Class (including owl:Restriction) node describe the properties of that class.
  An owl:Class is required to satisfy the conjunction of all constraints (implicit: intersection) stated about it.
  These characterizations are roughly the same as discussed for DL class definitions:
  - Set-theory constructors: owl:unionOf, owl:intersectionOf, owl:complementOf ($\mathcal{ALC}$)
  - Enumeration constructor: owl:oneOf (enumeration of elements; $\mathcal{O}$)
  - Axioms rdfs:subClassOf, owl:equivalentClass,
  - Axiom owl:disjointWith (also expressible in $\mathcal{ALC}$: C disjoint with D is equivalent to $C \sqsubseteq \neg D$)

OWL NOTIONS (CONT'D)

OWL Restriction Classes (Overview)

- owl:Restriction is a subclass of owl:Class, allowing for specification of a constraint on *one* property.
- one property is restricted by an owl:onProperty specifier and a constraint on this property:
  - ($\mathcal{N}$, $\mathcal{Q}$, $\mathcal{F}$) owl:cardinality, owl:minCardinality or owl:maxCardinality,
  - owl:allValuesFrom ($\forall R.C$), owl:someValuesFrom ($\exists R.C$),
  - owl:hasValue ($\mathcal{O}$),
  - including datatype restrictions for the range (D)
- by defining intersections of owl:Restrictions, classes having multiple such constraints can be specified.

OWL Notions (Cont'd)

OWL Property Axioms (Overview)

- Distinction between owl:ObjectProperty and owl:DatatypeProperty
- from RDFS: rdfs:domain/rdfs:range assertions, rdfs:subPropertyOf
- Axiom owl:equivalentProperty
- Axioms: subclasses of rdf:Property:
  owl:TransitiveProperty, owl:SymmetricProperty, owl:FunctionalProperty,
  owl:InverseFunctionalProperty (see Slide 380)

OWL Individual Axioms (Overview)

- Individuals are modeled by unary classes
- owl:sameAs, owl:differentFrom, owl:AllDifferent($o_1,\ldots,o_n$).

First-Order Logic Equivalents

OWL : $x \in C$	DL Syntax	FOL
C	C	$C(x)$
intersectionOf(C_1, C_2)	$C_1 \sqcap \ldots \sqcap C_n$	$C_1(x) \wedge \ldots \wedge C_n(x)$
unionOf(C_1, C_2)	$C_1 \sqcup \ldots \sqcup C_n$	$C_1(x) \vee \ldots \vee C_n(x)$
complementOf(C_1)	$\neg C_1$	$\neg C_1(x)$
oneOf$(x_1, \ldots, x_n)$	$\{x_1\} \sqcup \ldots \sqcup \{x_n\}$	$x = x_1 \vee \ldots \vee x = x_n$

OWL : $x \in C$, Restriction on P	DL Syntax	FOL
someValuesFrom(C')	$\exists P.C'$	$\exists y : P(x,y) \wedge C'(y)$
allValuesFrom(C')	$\forall P.C'$	$\forall y : P(x,y) \rightarrow C'(y)$
hasValue(y)	$\exists P.\{y\}$	$P(x,y)$
maxCardinality(n)	$\leq n.P$	$\exists^{\leq n} y : P(x,y)$
minCardinality(n)	$\geq n.P$	$\exists^{\geq n} y : P(x,y)$
cardinality(n)	$n.P$	$\exists^{=n} y : P(x,y)$

FIRST-ORDER LOGIC EQUIVALENTS (CONT'D)

OWL Class Axioms for C	DL Syntax	FOL
rdfs:subClassOf(C_1)	$C \sqsubseteq C_1$	$\forall x : C(x) \rightarrow C_1(x)$
equivalentClass(C_1)	$C \equiv C_1$	$\forall x : C(x) \leftrightarrow C_1(x)$
disjointWith(C_1)	$C \sqsubseteq \neg C_1$	$\forall x : C(x) \rightarrow \neg C_1(x)$

OWL Individual Axioms	DL Syntax	FOL
x_1 sameAs x_2	$\{x_1\} \equiv \{x_2\}$	$x_1 = x_2$
x_1 differentFrom x_2	$\{x_1\} \sqsubseteq \neg\{x_2\}$	$x_1 \neq x_2$
AllDifferent($x_1, \ldots, x_n$)	$\bigwedge_{i \neq j} \{x_i\} \sqsubseteq \neg\{x_j\}$	$\bigwedge_{i \neq j} x_i \neq x_j$

FIRST-ORDER LOGIC EQUIVALENTS (CONT'D)

OWL Properties	DL Syntax	FOL
P	P	$P(x, y)$

OWL Property Axioms for P	DL Syntax	FOL
rdfs:range(C)	$\top \sqsubseteq \forall P.C$	$\forall x, y : P(x, y) \rightarrow C(y)$
rdfs:domain(C)	$C \sqsupseteq \exists P.\top$	$\forall x, y : P(x, y) \rightarrow C(x)$
subPropertyOf(P_2)	$P \sqsubseteq P_2$	$\forall x, y : P(x, y) \rightarrow P_2(x, y)$
equivalentProperty(P_2)	$P \equiv P_2$	$\forall x, y : P(x, y) \leftrightarrow P_2(x, y)$
inverseOf(P_2)	$P \equiv P_2^-$	$\forall x, y : P(x, y) \leftrightarrow P_2(y, x)$
TransitiveProperty	$P^+ \equiv P$	$\forall x, y, z : ((P(x, y) \wedge P(y, z)) \rightarrow P(x, z))$
		$\forall x, z : ((\exists y : P(x, y) \wedge P(y, z)) \rightarrow P(x, z))$
FunctionalProperty	$\top \sqsubseteq \leq 1 P.\top$	$\forall x, y_1, y_2 : P(x, y_1) \wedge P(x, y_2) \rightarrow y_1 = y_2$
InverseFunctionalProperty	$\top \sqsubseteq \leq 1 P^-.\top$	$\forall x, y_1, y_2 : P(y_1, x) \wedge P(y_2, x) \rightarrow y_1 = y_2$

SYNTACTICAL REPRESENTATION

- OWL specifications can be represented by graphs: OWL constructs have a straightforward representation as triples in RDF/XML and Turtle.
- there are several logic-based representations (e.g. *Manchester OWL Syntax*); TERP (which can be used with pellet) is a combination of Turtle and Manchester syntax.
- OWL in RDF/XML format: usage of class, property, and individual names:
  - as @rdf:about when used as identifier of a subject (owl:Class, rdf:Property and their subclasses),
  - as @rdf:resource as the object of a property.
- some constructs need auxiliary structures (collections):
  owl:unionOf, owl:intersectionOf, and owl:oneOf are based on Collections
  - representation in RDF/XML by rdf:parseType="Collection".
  - representation in Turtle by (x_1 x_2 $\ldots x_n$)
  - as RDF lists: rdf:List, rdf:first, rdf:rest

REQUIREMENT

- every entity in an OWL ontology must be explicitly typed (i.e., as a class, an object property, a datatype property, . . . , or an instance of some class).
  (for reasons of space this is not always done in the examples; in general, it may lead to incomplete results)

QUERYING OWL DATA

- queries are atomic and conjunctive DL queries against the underlying OWL-DL model.
- this model can still be seen as a graph:
  - many of the edges are those known from the basic RDF graph
  - some edges (and collections) are only there for encoding OWL stuff (describing owl:unionOf, owl:propertyChain etc.) – these should not be queried
- SPARQL-DL is a subset of SPARQL: not every SPARQL query pattern is allowed for use on an OWL ontology
  (but the reasonable ones are, so in practice this is not a problem.)
- the query language SPARQL-DL allows exactly such well-sorted patterns using the notions of OWL.

SOME TBOX-ONLY REASONING EXAMPLES ON SETS

Example: A Simple Paradox

```
@prefix : <foo://bla/>.
@prefix owl: <http://www.w3.org/2002/07/owl#>.
:Paradox owl:complementOf :Paradox.
```

[Filename: RDF/paradox.n3]

- without reasoner:
  jena -t -ol rdf/xml -if paradox.n3
  Outputs the same RDF facts in RDF/XML without checking consistency.
- with reasoner:
  jena -e -pellet -if paradox.n3
  reads the RDF file, creates a model (and checks consistency) and in this case reports that it is not consistent:
  "There is an anonymous individual which is forced to belong to class foo://bla/Paradox and its complement"
- Note: the reasoner invents an anonymous individual for checking consistency.
  The empty interpretation (with empty domain!) would be a model of $P \equiv\neq P$.

UNION AS $A \sqcup B \equiv \neg((\neg A) \sqcap (\neg B))$ (DE MORGAN'S RULE)

```
@prefix : <foo://bla/>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix owl: <http://www.w3.org/2002/07/owl#>.
:A rdf:type owl:Class.        :B rdf:type owl:Class.
:Union1 owl:equivalentClass [ owl:unionOf (:A :B) ].
:CompA owl:complementOf :A.   :CompB owl:complementOf :B.
:IntersectComps owl:equivalentClass [ owl:intersectionOf (:CompA :CompB)].
:Union2 owl:complementOf :IntersectComps.
:x rdf:type :A.               :x rdf:type :B.
:y rdf:type :CompA. # a negative assertion y not in A would be better -> OWL 2
:y rdf:type :CompB.
```

[Filename: RDF/union.n3]

```
prefix owl: <http://www.w3.org/2002/07/owl#>
prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
prefix : <foo://bla/>
select ?X ?C ?D
from <file:union.n3>
where {{?X rdf:type ?C} UNION {:Union1 owl:equivalentClass ?D}}
```

[Filename: RDF/union.sparql]

EXAMPLE: UNION AND SUBCLASS

```
@prefix : <foo://bla/>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix owl: <http://www.w3.org/2002/07/owl#>.
:Male a owl:Class.      ## if these lines are missing,
:Female a owl:Class.    ## the reasoner complains
:Person owl:equivalentClass [ owl:unionOf (:Male :Female) ].
:EqToPerson owl:equivalentClass [ owl:unionOf (:Female :Male) ].
:unknownPerson a [ owl:unionOf (:Female :Male) ].
```

[Filename: RDF/union-subclass.n3]

- print class tree (with jena -e -pellet -if union-subclass.n3):

```
owl:Thing
    bla:Person = bla:EqToPerson - (bla:unknownPerson)
        bla:Female
        bla:Male
```

- Male and Female are derived to be subclasses of Person.
- Person and EqToPerson are equivalent classes.
- unknownPerson is a member of Person and EqToPerson.

Example (Cont'd)

```
prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>
prefix owl: <http://www.w3.org/2002/07/owl#>
prefix : <foo://bla/>
select ?SC ?C ?T ?CC ?CD
from <file:union-subclass.n3>
where {{?SC rdfs:subClassOf ?C} UNION
       {:unknownPerson rdf:type ?T} UNION
       {?CC owl:equivalentClass ?CD}}
```

[Filename: RDF/union-subclass.sparql]

- Note: OWLizations of DL class expressions are always handled as blank nodes, and used with "owl:equivalentClass", "rdf:subClassOf", "rdfs:domain", "rdfs:range" or "a".

Aside: the same in RDF/XML (usage of rdf:parseType="Collection")

```
<?xml version="1.0"?>
<rdf:RDF xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:f="foo://bla/"
    xml:base="foo://bla/">
 <owl:Class rdf:about="Person">
  <owl:equivalentClass>
   <owl:Class>
    <owl:unionOf rdf:parseType="Collection">
     <owl:Class rdf:about="Male"/>
     <owl:Class rdf:about="Female"/>
    </owl:unionOf>
   </owl:Class>
  </owl:equivalentClass>
 </owl:Class>
 <owl:Class rdf:about="EqToPerson">
  <owl:equivalentClass>
   <owl:Class>
    <owl:unionOf rdf:parseType="Collection">
     <owl:Class rdf:about="Female"/>
     <owl:Class rdf:about="Male"/>
    </owl:unionOf>
   </owl:Class>
  </owl:equivalentClass>
 </owl:Class>
 <f:Person rdf:about="unknownPerson"/>
</rdf:RDF>
```

[Filename: RDF/union-subclass.rdf]

EXERCISE

Consider

```
<owl:Class rdf:about="C1">
 <owl:equivalentClass>
  <owl:Class>
   <owl:intersectionOf rdf:parseType="Collection">
    <owl:Class rdf:about="A"/>
    <owl:Class rdf:about="B"/>
   </owl:intersectionOf>
  </owl:Class>
 </owl:equivalentClass>
</owl:Class>
```

and

```
<owl:Class rdf:about="C2">
 <rdfs:subClassOf rdf:resource="A"/>
 <rdfs:subClassOf rdf:resource="B"/>
</owl:Class>
```

- give mathematical characterizations of both cases.
- discuss whether both fragments are equivalent or not.

DISCUSSION

- Two classes are *equivalent* (wrt. the knowledge base) if they have the same interpretation in every *model* of the KB.
- C_1 is characterized to be the intersection of classes A and B.
- for C_2, it is asserted that C_2 is a subset of A and that it is a subset of B.
- Thus there can be some c that is in A, B, C_1, but not in C_2.
- Thus, C_1 and C_2 are not equivalent.
- C_1 is a definition, the statements about C_2 are just two constraints (C_2 might be empty); but it can be derived that it must be a subclass of C_1.

Discussion: Formal Notation

The DL equivalent to the knowledge base (TBox) is

$$\mathcal{T} = \{C_1 \equiv (A \sqcap B)\ , \quad C_2 \sqsubseteq A\ , \quad C_2 \sqsubseteq B\}$$

The First-Order Logic equivalent is

$$\mathcal{KB} = \{\forall x : A(x) \land B(x) \leftrightarrow C_1(x)\ , \quad \forall x : C_2(x) \to A(x) \land B(x)\}$$

Thus, $\mathcal{KB} \models \forall x : C_2(x) \to A(x) \land B(x)$.

Or, in DL: $\mathcal{T} \models C_2 \sqsubseteq C_1$.

On the other hand, $\mathcal{M} = (\mathcal{D}, \mathcal{I})$ with $\mathcal{D} = \{c\}$ and

$$\mathcal{I}(A) = \{c\},\ \ \mathcal{I}(B) = \{c\},\ \ \mathcal{I}(C_1) = \{c\},\ \ \mathcal{I}(C_2) = \emptyset$$

is a model of $\mathcal{KB}$ (wrt. first-order logic) and $\mathcal{T}$ (wrt. DL) that shows that C_1 and C_2 are not equivalent.

Subclasses of Properties

Triple syntax: *some property* rdf:type *a specific type of property*

According to their ranges

- owl:ObjectProperty – subclass of rdf:Property; object-valued (i.e. rdfs:range must be an Object class)
- owl:DatatypeProperty – subclass of rdf:Property; datatype-valued (i.e. its rdfs:range must be an rdfs:Datatype)

⇒ OWL ontologies require each property to be typed in such a way!
(for reasons of space sometimes omitted in examples)

According to their Cardinality

- specifying n:1 or 1:n cardinality:
  owl:FunctionalProperty, owl:InverseFunctionalProperty

⇒ useful for deriving that objects must be different from each other.

According to their Properties

- owl:TransitiveProperty, owl:SymmetricProperty see later ...

FUNCTIONAL CARDINALITY SPECIFICATION

property rdf:type owl:FunctionalProperty

- not a constraint, but
- if such a property results in two things ... these things are inferred to be the same.

```
@prefix : <foo://bla/meta#>.
@prefix persons: <foo://bla/persons/>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix owl: <http://www.w3.org/2002/07/owl#>.
  :world :has_pope persons:jorgebergoglio .
  :world :has_pope [ :name "Franziskus" ] .
  :has_pope rdf:type owl:FunctionalProperty.
```

[Filename: RDF/popes.n3]

```
prefix : <foo://bla/meta#>
prefix persons: <foo://bla/persons/>
select ?N  from <file:popes.n3>
where { persons:jorgebergoglio :name ?N }
```

[Filename: RDF/popes.sparql]

OWL:RESTRICTION – EXAMPLE

- owl:Restriction for $\exists p.C$ and $\forall p.C$. (cf. earlier examples)
- Definition of "Parent" as $\mathsf{Parent} \equiv \mathsf{Person} \sqcap \exists \mathsf{hasChild}.\top$
  (can be used for conclusions in both directions),
- Range axiom as constraint: $\mathsf{Parent} \sqsubseteq \forall \mathsf{hasChild}.\mathsf{Person}$
  (use only in the "$\Rightarrow$" direction)

```
@prefix : <foo://bla/meta#>.
@prefix p: <foo://bla/persons/>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix owl: <http://www.w3.org/2002/07/owl#>.
:Parent owl:equivalentClass
  [ owl:intersectionOf ( :Person
                         [ a owl:Restriction;
                             owl:onProperty :hasChild;  owl:minCardinality 1 ] ) ] .
:Parent rdfs:subClassOf [ a owl:Restriction;
                             owl:onProperty :hasChild;  owl:allValuesFrom :Person ] .
p:john a :Person; :hasChild p:alice .
p:sue a :Parent .
```

[Filename: RDF/restriction.n3]

owl:Restriction – Example (cont'd)

```
prefix : <foo://bla/meta#>
select ?X ?CC ?Y ?C
from <file:restriction.n3>
where {{?X a :Person; a ?CC} union {?Y :hasChild ?C}}
```

[File: RDF/restriction.sparql]

- How to check whether it knows that Sue has a child?
  - ... only *implicitly* known resources are never contained in SPARQL answers (impedance mismatch between SPARQL and DL).
  - they are only known *inside* the reasoner.
  - for looking inside the reasoner's "private" knowledge, appropriate auxiliary classes have to be defined in the OWL ontology which are then queried by SPARQL (as in many later examples)
- note also the separation of the domain into notions (<foo://bla/meta#>) and instances (<foo://bla/persons/>).
  This will not be cleanly done in the subsequent examples because it costs space.

Aside: owl:Restriction as RDF/XML

```
<?xml version="1.0"?>
<rdf:RDF xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns="foo://bla/meta#"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xml:base="foo://bla/persons/">
 <owl:Class rdf:about="Parent">
  <owl:equivalentClass>
   <owl:Class>
    <owl:intersectionOf rdf:parseType="Collection">
     <owl:Class rdf:about="Person"/>
     <owl:Restriction>
      <owl:onProperty rdf:resource="hasChild"/>
      <owl:minCardinality>1</owl:minCardinality>
     </owl:Restriction>
    </owl:intersectionOf>
   </owl:Class>
  </owl:equivalentClass>
 </owl:Class>
 <Person rdf:about="john">
   <hasChild><Person rdf:about="alice"/></hasChild>
 </Person>
</rdf:RDF>
```

[Filename: RDF/restriction.rdf]

Restrictions (and Other Class Specifications) as Separate Blank Nodes

Consider the following (bad) specification:

```
@prefix : <foo://bla/meta#>.   @prefix p: <foo://bla/persons/>.
@prefix owl: <http://www.w3.org/2002/07/owl#>.
:BadIdea a owl:Class; a owl:Restriction;
          owl:onProperty :hasChild; owl:minCardinality 1.
p:john :hasChild p:alice.
:cl a owl:Class.
```

[Filename: RDF/restrictionWrong.n3]

This is not allowed in OWL-DL.

- Note [13.6.2019]: specifications of that form, not using a blank node, are ignored by jena3.10/pellet (the class BadIdea does not exist):
  jena -e -if restriction.n3 -if restrictionWrong.n3

Correct specification:

```
:goodIdea owl:equivalentClass
 [a owl:Restriction; owl:onProperty :hasChild; owl:minCardinality 1].
```

Why? ... there are many reasons, for one of them see next slide.

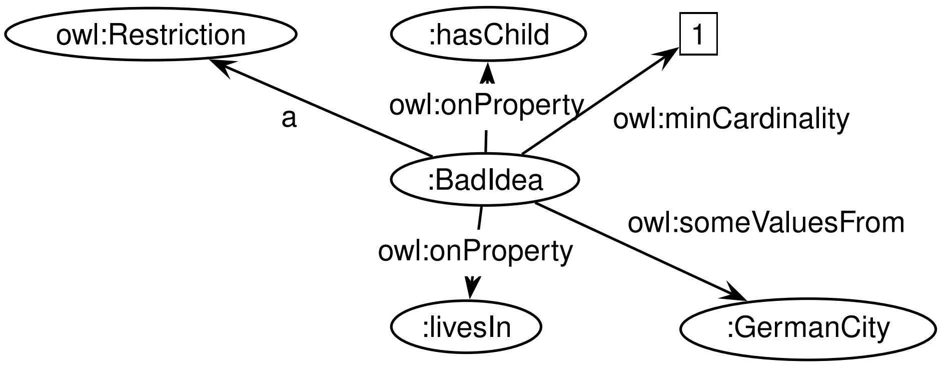
Restrictions Only as Blank Nodes (Cont'd)

A class with two such specifications:

```
@prefix : <foo://bla/meta#>.
@prefix owl: <http://www.w3.org/2002/07/owl#>.
:BadIdea a owl:Restriction; owl:onProperty :hasChild; owl:minCardinality 1 .
:BadIdea a owl:Restriction; owl:onProperty :livesIn; owl:someValuesFrom :GermanCity.
```

[Filename: RDF/badIdea.n3]

- call jena -t -pellet -if badIdea.n3:



The two restriction specifications are messed up.

Restrictions Only as Blank Nodes (Cont'd)

- Thus specify each Restriction specification with a separate blank node:

```
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix : <foo://bla/meta#>.
:TwoRestrictions owl:equivalentClass
 [ owl:intersectionOf
   ( [ a owl:Restriction; owl:onProperty :hasChild; owl:minCardinality 1]
     [ a owl:Restriction; owl:onProperty :livesIn; owl:someValuesFrom :GermanCity] ) ].
```

[Filename: RDF/twoRestrictions.n3]

The DL equivalent: TwoRestrictions $\equiv$ ($\exists$ hasChild.$\top$) $\sqcap$ ($\exists$ livesIn.GermanCity)

Another reason:

```
:BadSpecOfParent a owl:Restriction;
   owl:onProperty :hasChild; owl:minCardinality 1;
   rdfs:subClassOf :Person.
```

... mixes the *definition* of the Restriction with an assertive axiom:
BSOP $\equiv \exists \geq 1$ hasChild.$\top$ $\wedge$ ABDE $\sqsubseteq$ Person
(This expression probably does not meet the original intention – is *derives* that anything that has a child is made an instance of class "Person"; cf. Slide 377)

MULTIPLE RESTRICTIONS ON A PROPERTY

- "All persons that have at least two children, and one of them is male"
- first: a straightforward wrong attempt

```
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix : <foo://bla/meta#>.   @prefix p: <foo://bla/persons/>.
### Test: multiple restrictions: the owl:someValuesFrom-condition is then ignored
:HasTwoChildrenOneMale owl:intersectionOf (:Person
   [ a owl:Restriction; owl:onProperty :hasChild;
     owl:someValuesFrom :Male; owl:minCardinality 2]).
:name a owl:FunctionalProperty.
:Male rdfs:subClassOf :Person; owl:disjointWith :Female.
:Female rdfs:subClassOf :Person.
:kate a :Female; :name "Kate"; :hasChild :john.
p:john a :Male; :name "John";
  :hasChild [a :Female; :name "Alice"], [a :Male; :name "Bob"].
p:sue a :Female; :name "Sue";
  :hasChild [a :Female; :name "Anne"], [a :Female; :name "Barbara"].
```

[Filename: RDF/restriction-double.n3]

```
prefix : <foo://bla/meta#>
select ?X
from <file:restriction-double.n3>
where {?X a :HasTwoChildrenOneMale}
```

[Filename: RDF/restriction-double.sparql]

- The the owl:someValuesFrom-condition is ignored in this case (Result: John and Sue).

Multiple Restrictions on a Property

- "All persons that have at least two children, and one of them is male"
- to expressed as an *intersection* of two separate restrictions:

```
prefix : <foo://bla/meta#>
select ?X
from <file:intersect-restrictions.n3>
where {?X a :HasTwoChildrenOneMale}
```

[Filename: RDF/intersect-restrictions.sparql]

```
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix : <foo://bla/meta#>.      @prefix p: <foo://bl
:HasTwoChildrenOneMale owl:equivalentClass
 [ owl:intersectionOf (:Person
   [ a owl:Restriction; owl:onProperty :hasChild; owl:someValuesFrom :Male]
   [ a owl:Restriction; owl:onProperty :hasChild; owl:minCardinality 2] ) ].
:name a owl:FunctionalProperty.
:Male rdfs:subClassOf :Person; owl:disjointWith :Female.
:Female rdfs:subClassOf :Person.
p:kate a :Female; :name "Kate"; :hasChild p:john.
p:john a :Male; :name "John";
  :hasChild [a :Female; :name "Alice"], [a :Male; :name "Bob"].
p:sue a :Female; :name "Sue";
  :hasChild [a :Female; :name "Anne"], [a :Female; :name "Barbara"].
```

[Filename: RDF/intersect-restrictions.n3]

- Note: this is different from Qualified Range Restrictions such as "All persons that have at least two male children" – see Slide 451.

Use of a Derived Class

```
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix : <foo://bla/meta#>.      @prefix p: <foo://bla/persons/>.
p:kate :name "Kate"; :hasChild p:john.
p:john :name "John"; :hasChild p:alice.
p:alice :name "Alice".
:Parent a owl:Class; owl:equivalentClass
 [ a owl:Restriction; owl:onProperty :hasChild; owl:minCardinality 1].
:Grandparent owl:equivalentClass
 [a owl:Restriction; owl:onProperty :hasChild; owl:someValuesFrom :Parent].
```

[Filename: RDF/grandparent.n3]

```
prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>
prefix : <foo://bla/meta#>
select ?G ?P ?GP
from <file:grandparent.n3>
where {{?G a :Parent} UNION
       {?GP a :Grandparent} UNION
       {:Grandparent rdfs:subClassOf :Parent}}
```

[Filename: RDF/grandparent.sparql]

Non-Existence of Property Fillers (Possible Syntaxes)

```
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix : <foo://bla/meta#>.    @prefix p: <foo://bla/persons/>.
:ChildlessA owl:intersectionOf (:Person
  [ a owl:Restriction; owl:onProperty :hasChild; owl:maxCardinality 0]).
:ChildlessB owl:intersectionOf (:Person
  [ a owl:Restriction; owl:onProperty :hasChild; owl:allValuesFrom owl:Nothing]).
:ParentA owl:intersectionOf (:Person [owl:complementOf :ChildlessA]).          ### (*)
:ParentB owl:intersectionOf (:Person
  [ a owl:Restriction; owl:onProperty :hasChild; owl:minCardinality 1]).
:name a owl:FunctionalProperty.
p:john a :Person; :name "John"; :hasChild p:alice, p:bob.
p:sue a :ParentA; :name "Sue".
p:george a :Person; a :ChildlessA; :name "George".
```

[Filename: RDF/parents-childless.n3]

- export class tree: ChildlessA and ChildlessB are equivalent,
- ParentA and ParentB are also equivalent
- note: due to the Open World Assumption, only George is definitely known to be childless.
- Persons where parenthood is not known (Alice, Bob) are neither in Childless nor in Parent!
  Note: (*) states “Parent” vs. “Childless” as a disjoint, total partition of “Person”, but it is not *known* to which partition Alice and Bob belong. Both would be possible.

Non-Existence of Property Fillers – Open World vs. Closed World

- basically the same, Parent and Childless as classes, more persons,
- the focus is now on the different explicit and implicit knowledge about them:

```
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix : <foo://bla/meta#>.    @prefix p: <foo://bla/persons/>.
:Childless owl:intersectionOf (:Person
  [ a owl:Restriction; owl:onProperty :hasChild; owl:maxCardinality 0]).
:Parent owl:intersectionOf (:Person
  [ a owl:Restriction; owl:onProperty :hasChild; owl:minCardinality 1]).
:name a owl:FunctionalProperty.
p:kate a :Person; :name "Kate"; :hasChild p:john, p:sue.
p:john a :Person; :name "John"; :hasChild p:alice, p:bob.
p:alice a :Person; :name "Alice".
p:bob a :Person; :name "Bob".
p:sue a :Parent; :name "Sue".
p:george a :Person; a :Childless; :name "George".
```

[Filename: RDF/childless.n3]

```
prefix : <foo://bla/meta#>
select ?CL ?NCL ?P ?NP  ?X ?Y ?NHC from <file:childless.n3>
where {      {?CL a :Childless}
       union {?NCL a :Person FILTER NOT EXISTS { ?NCL a :Childless}}
       union {?P a :Parent}
       union {?NP a :Person FILTER NOT EXISTS { ?NP a :Parent}}
       union {?X :hasChild ?Y}
       union {?NHC a :Person FILTER NOT EXISTS {?NHC :hasChild ?Z}}}
```

[Filename: RDF/childless.sparql]

DL (and OWL) – everything that is done *inside the reasoner*: open world – monotonic, SPARQL: closed-world – non-monotonic:

- ?CL: only George is known to be Childless.
- ?NCL: Closed-World-Complement of ?C – all persons where it cannot be proven that they are childless – "definitely not childless or maybe not childless" – "where it is consistent to assume that they are not childless" – non-monotonic (all except George).
- Parents ?P: Sue, Kate, John;
- ?NP: Closed-World-Complement of ?P – ("consistent to be non-parents" – George, Alice, Bob)
- ?X, ?Y: only explicitly known parents/children (Sue's children not mentioned).
- ?NHC: George, Alice, Bob and Sue(!) – no children of them are *explicitly known*.

Inverse Properties

- *owl:ObjectProperty* owl:inverseOf *owl:ObjectProperty*
- owl:DatatypeProperties cannot have an inverse
  (this would define properties of objects, cf. next slide)

```
@prefix : <foo://bla/meta#>.
@prefix p: <foo://bla/persons/>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix owl: <http://www.w3.org/2002/07/owl#>.
:descendant rdf:type owl:TransitiveProperty.
:hasChild rdfs:subPropertyOf :descendant.
:hasChild owl:inverseOf :hasParent.
p:john :hasChild p:alice, p:bob.
p:john :hasParent p:kate .
```

[Filename: RDF/inverse.n3]

```
prefix : <foo://bla/meta#>
select ?X ?Y
from <file:inverse.n3>
where {?X :descendant ?Y}
```

[Filename: RDF/inverse.sparql]

No Inverses of owl:DatatypeProperties!

- an owl:DatatypeProperty must not have an inverse:
- ":john :age 35" would imply "35 :ageOf :john" which would mean that a literal has a property, which is not allowed.

```
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix : <foo://bla/meta#> .  @prefix p: <foo://bla/persons/> .
# p:john :name "John"; :age 35;
#        :hasChild [:name "Alice"], [:name "Bob"; :age 8].
:age a owl:DatatypeProperty.
:hasChild a owl:ObjectProperty.
:parent owl:inverseOf :hasChild.
:ageOf owl:inverseOf :age.
```

[Filename: RDF/inverseDTProp.n3]

```
jena -e -pellet -if inverseDTProp.n3
WARN [main] (OWLLoader.java:352) - Unsupported axiom:
Ignoring inverseOf axiom between foo://bla/meta#ageOf (ObjectProperty)
  and foo://bla/meta#age (DatatypeProperty)
```


SPECIFICATION OF INVERSE FUNCTIONAL PROPERTIES

- Mathematics: a mapping m is inverse-functional if the inverse of m is functional: $x\ p\ y$ is inverse-functional, if for every y, there is at most one x such that xpy holds.
- Example:
  - hasCarCode is functional: every country has one car code,
  - hasCarCode is also inverse functional: every car code uniquely identifies a country.
- OWL:

```
:m-inverse owl:inverseOf :m .
:m-inverse a owl:FunctionalProperty .
```

  not allowed for e.g. `mon:carCode a owl:DatatypeProperty`:

```
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix : <foo://bla#>.
:carCode a owl:DatatypeProperty; rdfs:domain :Country;
  owl:inverseOf :isCarCodeOf.
# :Germany :carCode "D".
```

[Filename: RDF/noinverse.n3]

- the statement is rejected.

OWL:INVERSEFUNCTIONALPROPERTY

- such cases are described with owl:InverseFunctionalProperty
- a property P is an owl:InverseFunctionalProperty if $\forall x, y_1, y_2 : P(y_1, x) \wedge P(y_2, x) \rightarrow y_1 = y_2$ holds

```
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix : <foo:bla#>.
:carCode rdfs:domain :Country; a owl:DatatypeProperty;
  a owl:FunctionalProperty; a owl:InverseFunctionalProperty.
:name a owl:DatatypeProperty; a owl:FunctionalProperty.
:Germany :carCode "D"; :name "Germany".
:DominicanRepublic :carCode "D"; :name "Dominican Republic".
```

[Filename: RDF/invfunctional.n3]

- the fragment is detected to be inconsistent.

OWL:hasKey (OWL 2)

Declaration of key attributes $(k_1, \ldots, k_n)$ is a relevant issue in data modeling.

- a key allows for unambiguously identifying a resource amongst a certain subset of the domain,
- in OWL, keys are not restricted to functional properties (i.e., SQL's UNIQUE is not required),
- values of key properties may be unknown for some instances; they might even be forbidden for some elements of the domain (e.g. using owl:maxCardinality 0 or owl:allValuesFrom owl:Nothing).
- note: InverseFunctionalProperty covers the simple case that $n = 1$ and the key is global.

```
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix : <foo:bla#>.
:name a owl:DatatypeProperty; a owl:FunctionalProperty.
:Country owl:hasKey (:carCode).
:DominicanRepublic a :Country; :carCode "D"; :name "Dominican Republic".
:Germany a :Country; :carCode "D"; :name "Germany".
```

[Filename: RDF/haskey.n3]

- the fragment is inconsistent.

OWL:hasKey (OWL 2) for Non-Functional Properties

- keys are not restricted to functional properties:

```
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix : <foo:bla#>.
:District owl:hasKey (:code).
:Country owl:hasKey (:code).
:goettingen a :District; :name "Goettingen"; :code "GOE", "DUD", "HMÃIJ".
:leipzig a :District; :name "Leipzig"; :code "L".
:lahndillkreis a :District; :name "Lahn-Dill-Kreis"; :code "LDK", "DIL", "WZ", "L".
:luxembourg a :Country; :name "Luxembourg"; :code "L".
```

[Filename: RDF/key-mvd.n3]

```
prefix : <foo:bla#>
select ?D ?N ?C
from <file:key-mvd.n3>
where { ?X a ?D ; :name ?N; :code ?C }
```

[Filename: RDF/key-mvd.sparql]

- Lahn-Dill-Kreis and Leipzig are identified (LDK had "L" from 1977-1990).
- Luxembourg is not identified with them since the key definitions are local to districts vs. countries.

OWL:hasKey (OWL 2) for Multi-Property-Keys

- consider triples about persons found in different Web sources.
- ABSOLUTELY BUGGY (27.7.2017) – it equates all four persons below:

```
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix : <foo:bla#>.
:Person owl:hasKey (:givenName :familyName).
 _:b1 a :Person; :givenName "John"; :familyName "Doe";  :age 35 .
 _:b2 a :Person; :givenName "John"; :familyName "Doe";  :address "Main Street 1" .
 _:b3 a :Person; :givenName "Mary"; :familyName "Doe";  :age 32; :address "Main Street 1" .
 _:b4 a :Person; :givenName "Donald"; :familyName "Trump";  :age 70; :address "White House" .
#:age a owl:FunctionalProperty.
```

[Filename: RDF/haskey2.n3]

```
prefix : <foo:bla#>
select ?X ?P ?Y
from <file:haskey2.n3>
where {?X a :Person ; ?P ?Y}
```

[Filename: RDF/haskey2.sparql]

NAMED AND UNNAMED RESOURCES

(from the DL reasoner's perspective)

Named Resources

- resources with explicit global URIs
  <http://www.semwebtech.org/mondial/10/country/D>
  <foo://bla/bob>
- resources with local IDs/named blank nodes
- unnamed blank nodes

Unnamed (implicit) Resources

- things that exist only implicitly:
  John's child in

```
:Parent a owl:Class; owl:equivalentClass
   [ a owl:Restriction; owl:onProperty :hasChild; owl:minCardinality 1].
:john a Parent.
```

- such implicit resources can even have properties (see next slides).

Implicit Resources

- "every person has a father who is a person" and "john is a person".
- the *standard model* is *infinite*:
  john, john's father, john's father's father, ...
- pure RDF graphs are always finite,
- only with OWL axioms, one can specify such infinite models,

⇒ they have only finitely many *locally to path length* n different nodes,

- the reasoner can detect the necessary n ("blocking", cf. Slides 508 ff) and create "typical" different structures.

Aside: "standard model" vs "nonstandard model"

- the term "standard model" is not only "what we understand (in this case)", but is a notion of mathematical theory which –roughly– means "the simplest model of a specification"
- nonstandard models of the above are those where there is a cycle in the ancestors relation.
  (as the length of the cycle is arbitrary, this would not make it easier for the reasoner - there is only the possibility to have an owl:sameAs somewhere)

Implicit Resources

```
@prefix : <foo://bla#>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix owl: <http://www.w3.org/2002/07/owl#>.
:Person owl:equivalentClass [a owl:Restriction;
  owl:onProperty :father; owl:someValuesFrom :Person].
:bob :name "Bob";  a :Person; :father :john.
:john :name "John";  a :Person.
```

[Filename: RDF/fathers-and-forefathers.n3]

```
prefix : <foo://bla#>
prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
select ?X ?F ?C
from <file:fathers-and-forefathers.n3>
where {{ ?X :father ?F } UNION { ?C a :Person }}
```

[Filename: RDF/fathers-and-forefathers.sparql]

- Reasoner: works on the model, including blocking, i.e. *modulo equivalence up to paths of length* n.
- SPARQL (and SWRL) rules: works on the graph – without the unnamed/implicit resorces.

9.3 RDF Graph vs. OWL Model; SPARQL vs. Reasoning

- SPARQL is an RDF (graph) query language
- OWL talks about models
- the RDF graph contains triples
  - ABox statements
  - partially also TBox – DL concepts are represented by several triples *that do not necessarily belong to the graph*
- The reasoner works in terms of DL concepts
  - when parsing input (RDF) files (or when activating the reasoner the first time), the DL concepts and assertions must be extracted ans communicated to the reasoner
- some only existentially known skolem objects exist only inside the reasoner
- (see Slide 562 for a sketch of the Jena architecture)

Queries against Metadata - Consequences (Overview)

⇒ SPARQL queries are answered against the graph of triples

- Some OWL notions are directly represented by triples, such as c a owl:Class.
- Some others are directly supported by special handling in the reasoners, e.g., c rdfs:subClassOf d and c owl:equivalentClass d.
- some others are only "answered" when given explicitly in the RDF input! The results then do not incorporate further results that could be found by reasoning!
- OWL notions in the input are often not contained as triples, but are only translated into DL atoms for the reasoner. (e.g. owl:Restriction definitions)
- Most OWL notions in queries are not "understood" as OWL, but only matched.

Queries against Resources

- SPARQL answers are only concerned with the RDF graph, not with existential things that are only known in the reasoner's model.

METADATA/ONTOLOGY LEVEL QUERYING

- SPARQL is defined by *matching* the underlying RDF graph.
- OWL triples are not always part of the RDF graph (they are intended to be translated into DL definitions in the reasoner)
- for traditional DL notions like

  ```
  ?C a owl:Class
  ?C a rdfs:subClassOf ?D
  ?C owl:equivalentClass ?D
  ?C owl:disjointWith ?D
  ```

  SPARQL implementations support to translate these internally into DL queries against the reasoner.
- SPARQL-DL (Sirin, Parsia OWLED 2007 [members of the Pellet team]) is a proposal that allows certain further OWL built-ins to be queried.

Test: Querying Metadata

Example: querying metadata (i) as triples, and (ii) by reasoning.

- the triples describing owl:Restrictions from the input are also in the graph and can be queried:

(intersect-restrictions.n3: Slide 389, cats-and-dogs.n3: Slide 451)

... see next slide


```
prefix : <foo://bla#>
prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>
prefix owl: <http://www.w3.org/2002/07/owl#>
select ?NC ?NC2 ?X ?P ?S ?A ?M ?C ?MQ ?Y ?Y2
from <file:intersect-restrictions.n3>
from <file:cats-and-dogs.n3>
where { ?X a owl:Restriction
        optional {?X owl:onProperty ?P}
        optional {?NC owl:equivalentClass ?X . filter(isURI(?NC))
                  optional {?NC rdfs:subClassOf ?Y . ?Y a owl:Restriction}}
        optional {?X rdfs:subClassOf ?Y2 . ?Y2 a owl:Restriction}
        optional {?X owl:equivalentClass ?NC2 . filter(isURI(?NC2))}
        optional {?X owl:onClass ?C}
        optional {?X owl:someValuesFrom ?S}
        optional {?X owl:allValuesFrom ?A}
        optional {?X owl:minCardinality ?M}
        optional {?X owl:minQualifiedCardinality ?MQ}}
```

[Filename: RDF/metadata-query.sparql]

- NC2 is never bound(!)
- NC=HasTwoDogs subClassOf Y=b1 is found
- X subClassOf Y2 is never bound(!)

Ontology Level Querying - a practical example

Consider again the “Childless” ontology from Slide 392.

Check that Childless $\sqcap$ Parent $= \emptyset$ and Person $\equiv$ Childless $\sqcup$ Parent (Partitioning)

- Allowed: (single line empty bindings result means true)

```
prefix : <foo://bla#>
prefix owl: <http://www.w3.org/2002/07/owl#>
select ?X from <file:childless.n3>
where { :Childless owl:disjointWith :Parent }
```

[Filename: RDF/childless1.sparql]

- Not allowed: complex class expression in the query (empty result since it tries a plain match with the RDF data)

```
prefix : <foo://bla#>
prefix owl: <http://www.w3.org/2002/07/owl#>
select ?X from <file:childless.n3>
where { :Person owl:equivalentClass [ owl:unionOf (:Childless :Parent) ] }
```

[Filename: RDF/childless2.sparql]

NOT ALLOWED

- instead: add auxiliary class definition to the TBox and export class tree with `jena -e -if childless.n3 childless3.n3`:

```
@prefix : <foo://bla#>.
@prefix owl: <http://www.w3.org/2002/07/owl#>.
:UnionCLP owl:equivalentClass [ owl:unionOf (:Childless :Parent) ] .
```

[Filename: RDF/childless3.n3]

NOT REASONED: OWL:FUNCTIONALPROPERTY

```
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix : <foo:bla#>.
:q a owl:FunctionalProperty.
:p a owl:ObjectProperty; rdfs:domain :D.
:D owl:equivalentClass [ a owl:Restriction; owl:onProperty :p;
                                            owl:maxCardinality 1 ].
# :x :p :a, :b.    :a owl:differentFrom :b.
```

[Filename:RDF/functional.n3]

```
prefix owl: <http://www.w3.org/2002/07/owl#>
prefix : <foo:bla#>
select ?P
from <file:functional.n3>
where { ?P a owl:FunctionalProperty }
```

[Filename:RDF/functional.sparql]

- tries just to match plain { ?P a owl:FunctionalProperty } triples in the RDF graph. Returns only q.
- does not *answer* that property q is in fact also functional, although the reasoner knows it.

NOT ALLOWED: COMPLEX TERMS IN SPARQL QUERIES

- example: all cities that are a capital
- works well with pellet alone (June 2017); not allowed with Jena

```
pellet query -query-file countrycaps.sparql \
       mondial-europe.n3 mondial-meta.n3 countrycaps.n3
```

- note: if the answer is empty, check that the mondial-namespace in the used mondial-meta.n3 is correct.

```
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix : <http://www.semwebtech.org/mondial/10/meta#> .
:CountryCapital owl:intersectionOf
  (:City [a owl:Restriction; owl:onProperty :isCapitalOf;
         owl:someValuesFrom :Country]).
```

[Filename: RDF/countrycaps.n3]

```
prefix owl: <http://www.w3.org/2002/07/owl#>
prefix : <http://www.semwebtech.org/mondial/10/meta#>
select ?N1 ?N2
where {{?X a :CountryCapital; :name ?N1} union
       {?Y a [a owl:Restriction; owl:onProperty :isCapitalOf;
              owl:someValuesFrom :Country]; :name ?N2}}
```

[Filename:RDF/countrycaps.sparql]

Not Allowed: Complex Terms in SPARQL Queries (Cont'd)

- all organizations whose headquarter city is a capital:
- neither allowed by pellet nor by jena+pellet (June 2017; worked with pellet alone in 2013)

```
pellet query -query-file organizations-query2.sparql \
  mondial-europe.n3 mondial-meta.n3
```

```
prefix owl: <http://www.w3.org/2002/07/owl#>
prefix : <http://www.semwebtech.org/mondial/10/meta#>
select ?A ?H
where {?X a [ owl:intersectionOf
            (:Organization [a owl:Restriction; owl:onProperty :hasHeadq;
                           owl:someValuesFrom
                             [ a owl:Restriction; owl:onProperty :isCapitalOf;
                               owl:someValuesFrom :Country ] ] ) ];
       :abbrev ?A; :hasHeadq ?C . ?C :name ?H . }
```

[Filename:RDF/organizations-query2.sparql]

HOW TO DO IT: SETS OF ANSWERS TO QUERIES AS AD-HOC CONCEPTS

- The result concept (and maybe others) must be added to the ontology.
- Example: all organizations whose headquarter city is a capital:

```
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix : <http://www.semwebtech.org/mondial/10/meta#> .
:CountryCapital owl:equivalentClass
  [ owl:intersectionOf
    (:City [a owl:Restriction; owl:onProperty :isCapitalOf;
           owl:someValuesFrom :Country])].
<bla:Result>  owl:equivalentClass [ owl:intersectionOf
  (:Organization [a owl:Restriction; owl:onProperty :hasHeadq;
    owl:someValuesFrom :CountryCapital])] .
```

[Filename: RDF/organizations-query.n3]

```
prefix : <http://www.semwebtech.org/mondial/10/meta#>
select ?A ?N
from <file:organizations-query.n3>
from <file:mondial-europe.n3>
from <file:mondial-meta.n3>
where {?X a <bla:Result> . ?X :abbrev ?A . ?X :hasHeadq ?C . ?C :name ?N}
```

[Filename:RDF/organizations-query.sparql]

SPARQL ON THE GRAPH: IMPLICITLY KNOWN RESOURCES

- SPARQL does not return any answer related with nodes (=resources) that are only implicitly known (=non-named resources)

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix : <foo://bla#>.
:ParentOf12YOChild  owl:equivalentClass [a owl:Restriction;
  owl:onProperty :hasChild; owl:someValuesFrom :12YOPerson].
:12YOPerson owl:equivalentClass [a owl:Restriction;
  owl:onProperty :age; owl:hasValue 12].
[ :name "John"; :age 35;  a :ParentOf12YOChild;
   :hasChild [:name "Alice"; :age 10], [:name "Bob"; :age 8]].
:age rdf:type owl:FunctionalProperty.
# :12YOPerson owl:equivalentClass owl:Nothing.

:TwoChildrenParent  owl:equivalentClass [a owl:Restriction;
  owl:onProperty :hasChild; owl:cardinality 2].
:ThreeChildrenParent  owl:equivalentClass [a owl:Restriction;
  owl:onProperty :hasChild; owl:minCardinality 3].
```

[Filename: RDF/john-three-children-impl.n3]

SPARQL and Non-Named Resources (Cont'd)

- implicit resources exist only on the reasoning level,
- not considered by SPARQL queries:

```
prefix : <foo://bla#>
prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
select ?X ?C ?A ?T
from <file:john-three-children-impl.n3>
where {{ ?X :name "John" . ?X a ?C }
       UNION {?X :age ?A} UNION {?T a :12YOPerson}}
```

[Filename: RDF/john-three-children-impl.sparql]

- John is a ThreeChildrenParent,
- no person known who is 12 years old
- adding `:12YOPerson owl:equivalentClass owl:Nothing` makes it inconsistent.
- implicity known things are also not considered for the OWL construct owl:hasKey (cf. Slides 398 and 416) and for SWRL rules (cf. Slides 519 ff).

[Aside/Example] owl:hasKey and Non-Named Resources

Show that owl:hasKey ignores resources that are only implicitly known (OWL ontology see next slide):

- create an (infinite) sequence of implicitly known fathers ... all being persons and having the name "Adam",
- guarantee that the sequence consists of different objects by making it irreflexive.
  (note: Transitivity and Irreflexivity are not allowed together, thus actually only every person is required to be different from his/her father – the grandfather might be the person again)

```
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix : <foo:bla#>.
:Person owl:hasKey (:name) .
:name a owl:DatatypeProperty .
# :name a owl:InverseFunctionalProperty .  ## that would do it instead of hasKey
:father a owl:FunctionalProperty, owl:IrreflexiveProperty; rdfs:range :Person.
:bob a :Person; :father :john .
:john :name "John" .
:Adam owl:equivalentClass [ a owl:Restriction; owl:onProperty :name; owl:hasValue "Adam"] .
:Person rdfs:subClassOf
  [ a owl:Restriction; owl:onProperty :father; owl:someValuesFrom :Adam].
:JohnAdam owl:equivalentClass [ owl:intersectionOf ( :Adam
     [ a owl:Restriction; owl:onProperty :name; owl:hasValue "John"] ) ].
:hasFatherJohnAdam owl:equivalentClass [ a owl:Restriction;
     owl:onProperty :father; owl:someValuesFrom :JohnAdam ] .
:hasGrandpaAdam owl:equivalentClass [ a owl:Restriction; owl:onProperty :father;
    owl:someValuesFrom [ a owl:Restriction; owl:onProperty :father;
       owl:someValuesFrom :Adam ] ].
:AdamFatherAdam owl:equivalentClass [ owl:intersectionOf (:Adam
     [ a owl:Restriction; owl:onProperty :father;  owl:someValuesFrom :Adam] ) ] .
```
[Filename: RDF/forefathers-keys.n3]

[Aside/Example] owl:hasKey and Non-Named Resources

```
prefix owl: <http://www.w3.org/2002/07/owl#>
prefix : <foo:bla#>
SELECT ?N ?A ?FA ?AFA ?GPA
FROM <forefathers-keys.n3>
WHERE {{ :bob :father [ :name ?N ] }
        # UNION { ?A :name "Adam" }   ## error/bug complains about anon(1)
        UNION { ?FA a :hasFatherJohnAdam }
        UNION { ?AFA a :AdamFatherAdam }
        UNION { ?GPA a :hasGrandpaAdam }}
```
[Filename: RDF/forefathers-keys.sparql]

- implicit nodes are not considered in the answers.
- owl:hasKey is not violated by the fact that several only implicitly known people are named “Adam”.
  Note that John, being Bob’s father, also gets the name “Adam”.

[Aside/Example] owl:hasKey and Non-Named Resources

Another example using multi-attribute keys (which could not be replaced by owl:InverseFunctionalProperty):

- nodes in a (x,y)-coordinate system; consider (10,10)
- insert a pointer to an implicit node (10,10).

```
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix : <foo:bla#>.
:XYThing owl:hasKey (:x :y).
:xy10 a :XYThing; :x 10; :y 10; :text "free".
:XYTen owl:intersectionOf ([ a owl:Restriction; owl:onProperty :x; owl:hasValue 10]
                           [ a owl:Restriction; owl:onProperty :y; owl:hasValue 10]
               [ a owl:Restriction; owl:onProperty :text; owl:hasValue "pointedTo"]).
:pointTo a owl:FunctionalProperty; rdfs:range :XYThing.
:foo a [ a owl:Restriction;
         owl:onProperty :pointTo; owl:onClass :XYTen; owl:qualifiedCardinality 1].
# :foo :pointTo :xyxy.   ## functionality of pointTo: makes :xyxy=(10,10) explicit
```

[Filename: RDF/easykeys-impl.n3]

Aside/Example owl:hasKey and Non-Named Resources (Cont'd)

```
prefix owl: <http://www.w3.org/2002/07/owl#>
prefix : <foo:bla#>
SELECT ?CT ?Y ?T ?SameAsxyxy
FROM <easykeys-impl.n3>
WHERE {{ :foo :pointTo [ :text ?CT ] }
        UNION { ?Y :text ?T }
        UNION { [:text ?T] }
        UNION { :xyxy owl:sameAs ?SameAsxyxy }}
```

[Filename: RDF/easykeys-impl.sparql]

Implicit nodes are not considered in the answers.

- with last in line in source commented out: not much – the “pointTo” text is not answered, nothing is :sameAs.
- with last line commented in: the implicit node which is pointed to is equated with :xyxy, made explicit and then equated also with :xy10.

[ASIDE] OWL VS. RDF LISTS

- RDF provides structures for representing lists by triples (cf. Slide 242): rdf:List, rdf:first, rdf:rest.
  These are *distinguished* classes/properties.
- OWL/reasoners have a still unclear relationship with these:
  - use of lists for its internal representation of owl:unionOf, owl:oneOf etc. (that are actually based on collections),
  - do or do not allow the user to query this internal representation,
  - ignore user-defined lists over usual resources.

[ASIDE] UNIONOF (ETC) AS TRIPLES: LISTS

- owl:unionOf (x y z), owl:oneOf (x y z) is actually only syntactic sugar for RDF lists.
- The following are equivalent:

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

:Male a owl:Class.
:Female a owl:Class.

:Person a owl:Class; owl:unionOf (:Male :Female).
:EqToPerson a owl:Class;
  owl:unionOf
  [ a rdf:List; rdf:first :Male;
    rdf:rest [ a rdf:List; rdf:first :Female; rdf:rest rdf:nil]].
:x a :Person.
```

[Filename: RDF/union-list.n3]

- jena -t -if union-list.n3: both in usual Turtle notation as owl:unionOf (:Male :Female).

[ASIDE] UNIONOF (ETC) AS TRIPLES (CONT'D)

```
prefix owl: <http://www.w3.org/2002/07/owl#>
prefix : <foo://bla#>
select ?C
from <file:union-list.n3>
where {:Person owl:equivalentClass ?C}
```

[Filename: RDF/union-list.sparql]

- jena -q -pellet -qf union-list.sparql: both are equivalent.

```
prefix owl: <http://www.w3.org/2002/07/owl#>
prefix : <foo://bla#>
select ?P1 ?P2 ?X ?Q ?R ?S ?T
from <file:union-list.n3>
where {{:Person owl:equivalentClass :EqToPerson} UNION
  {:Person ?P1 ?X . ?X ?Q ?R . OPTIONAL {?R ?S ?T}} UNION
  {:EqToPerson ?P2 ?X . ?X ?Q ?R} . OPTIONAL {?R ?S ?T}}
```

[Filename: RDF/union-list2.sparql]

- both have actually the same list structure
  (pellet2/nov 2008: fails; pellet 2.3/sept 2009: fails)

[ASIDE] REASONING OVER LISTS (PITFALLS!)

- rdf:first and rdf:rest are (partially) ignored for reasoning (at least by pellet?); they cannot be used for deriving other properties from it.
- they can even not be used in queries (since pellet2/nov 2008; before it just showed weird behavior)

```
prefix rdf:
<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
prefix owl: <http://www.w3.org/2002/07/owl#>
prefix : <foo://bla#>
select ?X ?Y ?Z
from <file:union-list.n3>
where {?X a rdf:List; rdf:first ?Y .
       OPTIONAL {?X rdf:rest ?Z}}
```

[Filename: RDF/union-list3.sparql]

- jena-tool with pellet2.3: OK.
- pellet2.3: NullPointerException.

[Aside] Extension of a class defined by a list

Given an RDF list as below, define an owl:Class :Invited which contains exactly the elements in the list (i.e., in the above sample data, :alice, :bob, :carol, :dave).

```
@prefix : <foo:bla#>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
# Problem: when the real rdf namespace is used, rdf:first/rest are ignored
@prefix rdfL: <http://www.w3.org/1999/02/22-rdf-syntax-nsL#>.    # <<<<<<<<<<<<<
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix owl: <http://www.w3.org/2002/07/owl#>.

:Invited a owl:Class.
:InvitationList rdfs:subClassOf rdfL:List.
:list1 a :InvitationList; rdfL:first :alice;
  rdfL:rest [a rdfL:List; rdfL:first :bob;
    rdfL:rest [a rdfL:List; rdfL:first :carol;
      rdfL:rest [a rdfL:List; rdfL:first :dave; rdfL:rest rdf:nil]]].

# rest of an InvitationList is also an InvitationList
:InvitationList owl:equivalentClass
  [a owl:Restriction;
    owl:onProperty rdfL:rest; owl:allValuesFrom :InvitationList],
  [ a owl:Restriction;
    owl:onProperty rdfL:first; owl:allValuesFrom :Invited].
```

```
prefix : <foo:bla#>
select ?I
from <file:invitation-list.n3>
where {?I a :Invited}
```

[Filename: RDF/invitation-list.sparql]

[Filename: RDF/invitation-list.n3]

9.4 Nominals: The O in SHOIQ

TBox vs. ABox

Description Logics Terminology

Clean separation between TBox and ABox vocabulary:

- TBox: RDFS/OWL vocabulary for information about classes and properties (further partitioned into definitions and axioms),
- ABox: Domain vocabulary and rdf:type.

RDF/RDF/OWL Ontologies

- Syntactically: allow to mix everything in a single set of triples.
- OWL-DL restriction: clean usage of individuals vs. classes
  - individuals only in application property triples (ABox)
  - classes only in context of RDFS/OWL built-ins (like (X a :Person) or (:hasChild rdfs:range :Person), etc.) (TBox)

Recall: Reification

- Reification treats a class (e.g. :Penguin) or a property as an individual (:Penguin a :Species)
- reification assigns properties from an application domain to classes and properties.
- useful when talking about metadata notions,
- risk: allows for paradoxes.

NOMINALS

- use individuals (that usually occur only in the ABox) in *specific positions* in the TBox:
- as individuals (that are often implemented in the reasoner as unary classes) with [a owl:Restriction; owl:onProperty *property*; owl:hasValue *object*] (the class of all things such that {?x *property object*} holds).
- in enumerated classes *class* owl:oneOf ($o_1,\ldots,o_n$) (*class* is defined to be the set {$o_1,\ldots,o_n$}).

USING NOMINALS: ITALIAN CITIES

```
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix mon: <http://www.semwebtech.org/mondial/10/meta#>.
@prefix it: <foo://bla#>.
it:Italy owl:sameAs <http://www.semwebtech.org/mondial/10/countries/I/>.
it:ItalianCity a owl:Class; owl:intersectionOf
  (mon:City
   [a owl:Restriction; owl:onProperty mon:cityIn;
    owl:hasValue it:Italy]).   # Nominal: an individual in a TBox axiom
```

[Filename: RDF/italiancities.n3]

```
prefix it: <foo://bla#>
select ?X ?Y
from <file:mondial-meta.n3>
from <file:mondial-europe.n3>
from <file:italiancities.n3>
where {?X a it:ItalianCity}
```

[Filename: RDF/italiancities.sparql]

- the query {?X :cityIn <http://www.semwebtech.org/mondial/10/countries/I/>} would be shorter, but here a class should be defined for further use ...

An Ontology in OWL

Consider the Italian-English-Ontology from Slide 52.

```
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix f: <foo://bla#>.
f:Italian rdfs:subClassOf f:Person;
  owl:disjointWith f:English;
  owl:unionOf (f:Lazy f:LatinLover).
f:Lazy owl:disjointWith f:LatinLover.
f:English rdfs:subClassOf f:Person.
f:Gentleman rdfs:subClassOf f:English.
f:Hooligan rdfs:subClassOf f:English.
f:LatinLover rdfs:subClassOf f:Gentleman.
```

Class tree with jena -e:

```
owl:Thing
   bla:Person
      bla:English
         bla:Hooligan
         bla:Gentleman
      bla:Italian = bla:Lazy
owl:Nothing = bla:LatinLover
```

- LatinLover is empty, thus Italian ≡ Lazy.

[Filename: RDF/italian-english.n3]

Italians and Englishmen (Cont'd)

- the conclusions apply to the instance level:

```
@prefix : <foo://bla#>.
:mario a :Italian.
```

[Filename: RDF/mario.n3]

```
prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
prefix : <foo://bla#>
select ?C
from <file:italian-english.n3>
from <file:mario.n3>
where {:mario rdf:type ?C}
```

[Filename: RDF/italian-english.sparql]

AN ONTOLOGY IN OWL

Consider the Italian-Professors-Ontology from Slide 53.

```
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix it: <foo://bla#>.
it:Bolzano owl:sameAs
<http://www.semwebtech.org/mondial/10/countries/I/provinces/TrentinoAltoAdige/cities/Bolzano/>
it:Italian owl:intersectionOf
  (it:Person
   [a owl:Restriction; owl:onProperty it:livesIn;
    owl:someValuesFrom it:ItalianCity]);
  owl:unionOf (it:Lazy it:Mafioso it:LatinLover).
it:Professor rdfs:subClassOf it:Person.
it:Lazy owl:disjointWith it:ItalianProf;
   owl:disjointWith it:Mafioso;
   owl:disjointWith it:LatinLover.
it:Mafioso owl:disjointWith it:ItalianProf;
   owl:disjointWith it:LatinLover.
it:ItalianProf owl:intersectionOf (it:Italian it:Professor).
it:enrico a it:Professor; it:livesIn it:Bolzano.
```

[Filename: RDF/italian-prof.n3]

```
prefix : <foo://bla#>
select ?C
from <file:italian-prof.n3>
from <file:mondial-meta.n3>
from <file:mondial-europe.n3>
from <file:italiancities.n3>
where {:enrico a ?C}
```

[Filename: RDF/italian-prof.sparql]

ENUMERATED CLASSES: ONEOF

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix mon: <http://www.semwebtech.org/mondial/10/meta#>.
<bla:MontanunionMembers> owl:intersectionOf
 (mon:Country
  [owl:oneOf
   (<http://www.semwebtech.org/mondial/10/countries/NL/>
    <http://www.semwebtech.org/mondial/10/countries/B/>
    <http://www.semwebtech.org/mondial/10/countries/L/>
    <http://www.semwebtech.org/mondial/10/countries/F/>
    <http://www.semwebtech.org/mondial/10/countries/I/>
    <http://www.semwebtech.org/mondial/10/countries/D/>)]).
<bla:Result> owl:intersectionOf (mon:Organization
  [a owl:Restriction; owl:onProperty mon:hasMember;
   owl:someValuesFrom <bla:MontanunionMembers>]).
```

[Filename: RDF/montanunion.n3]

```
select ?X
from <file:montanunion.n3>
from <file:mondial-europe.n3>
from <file:mondial-meta.n3>
where {?X a <bla:Result>}
```

[RDF/montanunion.sparql]

- Query: all organizations that share a member with the Montanunion.

oneOf (Example Cont'd)

- previous example: "all organizations that share a member with the Montanunion."
  (DL: $x \in \exists$hasMember.MontanunionMembers)
- "all organizations where *all* members are also members of the Montanunion."
  (DL: $x \in \forall$hasMember.MontanunionMembers)
- The result is empty (although there is e.g. BeNeLux) due to open world: it is not known whether there may exist additional members of e.g. BeNeLux.
- Only if the membership of Benelux is "closed", results can be proven:

```
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix mon: <http://www.semwebtech.org/mondial/10/meta#>.
<http://www.semwebtech.org/mondial/10/organizations/Benelux/>
 a [a owl:Restriction;
     owl:onProperty mon:hasMember; owl:cardinality 3].
<bla:SubsetOfMU> owl:intersectionOf (mon:Organization
   [a owl:Restriction; owl:onProperty mon:hasMember;
     owl:allValuesFrom <bla:MontanunionMembers>]).
mon:name a owl:FunctionalProperty. # not yet given in th
```

[Filename: RDF/montanunion2.n3]

```
select ?X
from <file:montanunion.n3>
from <file:montanunion2.n3>
from <file:mondial-europe.n3>
from <file:mondial-meta.n3>
where {?X a <bla:SubsetOfMU>}
```

[RDF/montanunion2.sparql]

oneOf (Example Cont'd)

- "all organizations that cover *all* members of the Montanunion."

```
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix mon: <http://www.semwebtech.org/mondial/10/meta#>.
<bla:EUMembers> owl:equivalentClass  [a owl:Restriction;
    owl:onProperty mon:isMember; owl:hasValue
    <http://www.semwebtech.org/mondial/10/organizations/EU/>].
```

[Filename: RDF/montanunion3.n3]

```
prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>
select ?X # ?Y ?Z
from <file:montanunion.n3>
from <file:montanunion3.n3>
from <file:mondial-europe.n3>
from <file:mondial-meta.n3>
where {#{?Y a <bla:EUMembers>} UNION {?Z a <bla:MontanunionMembers>} UNION
       {<bla:MontanunionMembers> rdfs:subClassOf ?X}}
```

[Filename: RDF/montanunion3.sparql]

ONEOF (EXAMPLE CONT'D)

Previous example:

- only for one organization
- defined a class that contains all members of the organization
- not possible to define a *family of classes* – one class for each organization.
- this would require a *parameterized constructor*:
  "c_{org} is the set of all members of org"
  Second-Order Logic: each organization can be seen as a unary predicate (=set):
  $\forall Org : Org(c) \leftrightarrow$ hasMember(Org, c)
  or in F-Logic syntax: `C isa Org :- Org:organization[hasMember->C]`
  yields e.g.
  $I(eu) = \{germany, france, \ldots\}$,
  $I(nato) = \{usa, canada, germany, \ldots\}$
  Recall that "organization" itself is a predicate:
  $I(organization) = \{eu, nato, \ldots, \}$
  So we have again reification: organizations are both first-order-individuals and classes.

CONVENIENCE CONSTRUCT: OWL:ALLDIFFERENT

- owl:oneOf defines a class as a closed set;
- in owl:oneOf ($x_1, \ldots, x_n$), two items may be the same (open world),

owl:AllDifferent

- Triples of the form :a owl:differentFrom :b state that two individuals are different.
  For a database with n elements, one needs
  $(n-1) + (n-2) + \ldots + 2 + 1 = \sum_{i=1..n} i = n \cdot (n+1)/2 = O(n^2)$ such statements.
- The –purely syntactical– convenience construct

  [a owl:AllDifferent; owl:members ($r_1\ r_2 \ldots r_n$)]

  provides a shorthand notation.
  - it is *immediately* translated into the set of all statements
    $\{r_i$ owl:differentFrom $r_j \mid i \neq j \in 1..n\}$
  - [a owl:AllDifferent; owl:members (...)]
    is to be understood as a (blank node) that acts as a *specification* that the listed things are different that does not actually exist in the model.

[SYNTAX] OWL:ALLDIFFERENT IN RDF/XML

```
<?xml version="1.0"?>
<rdf:RDF xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:f="foo://bla#"  xml:base="foo://bla#">
 <owl:Class rdf:about="Foo">
  <owl:equivalentClass> <owl:Class>
    <owl:oneOf rdf:parseType="Collection">
     <owl:Thing rdf:about="a"/>  <owl:Thing rdf:about="b"/>
     <owl:Thing rdf:about="c"/>  <owl:Thing rdf:about="d"/>
    </owl:oneOf>
   </owl:Class> </owl:equivalentClass>
 </owl:Class>
 <owl:AllDifferent>  <!-- use like a class, but is only a shorthand -->
    <owl:members rdf:parseType="Collection">
     <owl:Thing rdf:about="a"/>  <owl:Thing rdf:about="b"/>
     <owl:Thing rdf:about="c"/>  <owl:Thing rdf:about="d"/>
    </owl:members>
 </owl:AllDifferent>
 <owl:Thing rdf:about="a"> <owl:sameAs rdf:resource="b"/> </owl:Thing>
</rdf:RDF>
```

[Filename: RDF/alldiff.rdf]

```
prefix : <foo://bla#>
prefix owl: <http://www.w3.org/2002/07/owl#>
select ?X ?P ?P2 ?V
from <file:alldiff.rdf>
where {?X a owl:AllDifferent ;
       ?P [?P2 ?V]}
```

[Filename: RDF/alldiffxml.sparql]

- AllDifferent is only intended as a kind of command to the application to add all pairwise “different-from” statements, it does not actually introduce itself as triples:
- querying {?X a owl:AllDifferent} is actually not intended.

[SYNTAX] OWL:ALLDIFFERENT IN TURTLE

Example:

```
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix : <foo://bla#>.
:Foo owl:equivalentClass [ owl:oneOf (:a :b :c :d) ].
# both the following syntaxes are equivalent and correct:
[ a owl:AllDifferent; owl:members (:a :b)].
[] a owl:AllDifferent; owl:members (:c :d).
:a owl:sameAs :b.
# :b owl:sameAs :d.
```

[Filename: RDF/alldiff.n3]

```
prefix : <foo://bla#>
prefix owl: <http://www.w3.org/2002/07/owl#>
select ?X ?Y
from <file:alldiff.n3>
where {?X a owl:AllDifferent ; ?P [?P2 ?V]}
```

[Filename: RDF/alldiff.sparql]

ONEOF: A TEST

- owl:oneOf defines a “closed set” (use with anonymous class; see below):
- note that in owl:oneOf ($x_1, \ldots, x_n$), two items may be the same (open world),
- optional owl:AllDifferent to guarantee that ($x_1, \ldots, x_n$) are pairwise distinct.

```
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix : <foo://bla#>.
:Person owl:equivalentClass [ owl:oneOf (:john :alice :bob) ].
# :john owl:sameAs :alice.  # to show that it is consistent that they are the same
[] a owl:AllDifferent; owl:members (:john :alice :bob).  # to guarantee distinctness
# :name a owl:FunctionalProperty.  # this also guarantees distinctness ;)
:john :name "John".
:alice :name "Alice".
:bob :name "Bob".
:d a :Person.
:d owl:differentFrom :john, :alice.
# :d owl:differentFrom :bob.   ### adding this makes the ontology inconsistent
```

[Filename: RDF/three.n3]

- Who is :d?

oneOf: a Test (cont’d)

Who is :d?

- check the class tree:
  bla:Person - (bla:bob, bla:alice, bla:d, bla:john)
  The class tree does not indicate which of the “four” identifiers are the same.
- and ask it:

```
prefix : <foo://bla#>
select ?N
from <file:three.n3>
where {:d :name ?N}
```

  [Filename: RDF/three.sparql]
  The answer is ?N/“Bob”.

A bug in Pellet

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.
@prefix : <foo:bla/meta#>.

:Person a owl:Class ; owl:equivalentClass [ owl:oneOf (:john :alice) ] .
[ a owl:AllDifferent ; owl:members (:john :alice) ] .
:john a :Person .
:alice a :Person .
:bob a :Person .
```

[Filename: RDF/one-of-bug.n3]

```
prefix :<foo:bla/meta#>
prefix owl: <http://www.w3.org/2002/07/owl#>
SELECT ?X ?Y ?Z
FROM <file:one-of-bug.n3>
WHERE {{?X owl:differentFrom ?Y} UNION {?X owl:sameAs ?Z}}
```

[Filename: RDF/one-of-bug.sparql]

john differentFrom alice, but not the other way round, and not john sameAs bob.

X	Y	Z
bob	alice	
john	alice	
alice	john	
bob		bob
john		john
alice		alice

9.5 Closing Parts of the Open World

- “forall items” is only applicable if additional items can be excluded (⇒ locally closed predicate/property),
- often, RDF data is generated from a database,
- certain predicates can be closed by defining restriction classes with maxCardinality.

Closing Parts of the Open World for owl:allValuesFrom

```
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix : <foo://bla#>.
[ a :Male, :ThreeChildrenParent; :name "John";
   :hasChild [a :Female; :name "Alice"], [a :Male; :name "Bob"],
             [a :Female; :name "Carol"]].
[ a :Female, :TwoChildrenParent; :name "Sue";
   :hasChild [a :Female; :name "Anne";], [a :Female; :name "Barbara"]].
:name a owl:FunctionalProperty.
:OneChildParent owl:equivalentClass [a owl:Res
  owl:onProperty :hasChild; owl:cardinality 1]
:TwoChildrenParent owl:equivalentClass [a owl:
  owl:onProperty :hasChild; owl:cardinality 2]
:ThreeChildrenParent owl:equivalentClass [a owl:Restriction;
  owl:onProperty :hasChild; owl:cardinality 3].
:OnlyFemaleChildrenParent owl:equivalentClass [a owl:Restriction;
  owl:onProperty :hasChild; owl:allValuesFrom :Female].
```

```
prefix : <foo://bla#>
select ?N
from <file:allvaluesfrom.n3>
where {?X :name ?N .
  ?X a :OnlyFemaleChildrenParent}
```

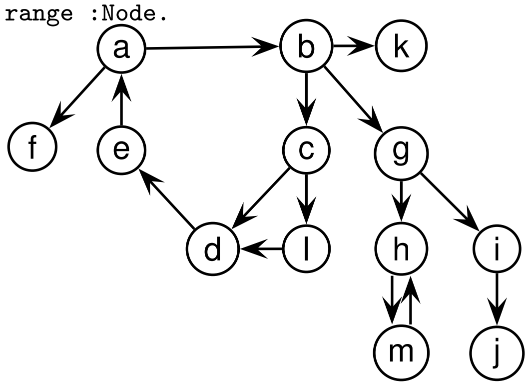
[Filename: RDF/allvaluesfrom.sparql]

[Filename: RDF/allvaluesfrom.n3]

EXAMPLE: WIN-MOVE-GAME IN OWL

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix : <foo://bla#>.

:Node a owl:Class; owl:equivalentClass
  [ a owl:Class; owl:oneOf (:a :b :c :d :e :f :g :h :i :j :k :l :m)].
:edge a owl:ObjectProperty; rdfs:domain :Node; rdfs:range :Node.
:out a owl:DatatypeProperty.
:a a :Node; :out 2; :edge :b, :f.
:b a :Node; :out 3; :edge :c, :g, :k.
:c a :Node; :out 2; :edge :d, :l.
:d a :Node; :out 1; :edge :e.
:e a :Node; :out 1; :edge :a.
:f a :Node; :out 0 .
:g a :Node; :out 2; :edge :i, :h.
:h a :Node; :out 1; :edge :m.
:i a :Node; :out 1; :edge :j.
:j a :Node; :out 0 .
:k a :Node; :out 0 .
:l a :Node; :out 1; :edge :d.
:m a :Node; :out 1; :edge :h.
```



[Filename: RDF/winmove-graph.n3]

Win-Move-Game in OWL – the Game Axioms

"If a player cannot move, he loses."

Which nodes are WinNodes, which one are LoseNodes (i.e., the player who has to move wins/loses)?

- if a player can move to some LoseNode (for the other), he will win.
- if a player can move only to WinNodes (for the other), he will lose.
- recall that there can be nodes that are neither WinNodes nor LoseNodes.

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix : <foo://bla#>.

:WinNode a owl:Class; owl:intersectionOf ( :Node
  [a owl:Restriction; owl:onProperty :edge; owl:someValuesFrom :LoseNode]).
:LoseNode a owl:Class; owl:intersectionOf ( :Node
  [a owl:Restriction; owl:onProperty :edge; owl:allValuesFrom :WinNode]).
```

[Filename: RDF/winmove-axioms.n3]

Win-Move-Game in OWL – Closure

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix : <foo://bla#>.
:DeadEndNode a owl:Class;  rdfs:subClassOf :Node;
  owl:equivalentClass [ a owl:Restriction; owl:onProperty :out; owl:hasValue 0],
                      [ a owl:Restriction; owl:onProperty :edge; owl:cardinality 0].
:OneExitNode a owl:Class; rdfs:subClassOf :Node;
  owl:equivalentClass [ a owl:Restriction; owl:onProperty :out; owl:hasValue 1],
                      [ a owl:Restriction; owl:onProperty :edge; owl:cardinality 1].
:TwoExitsNode a owl:Class; rdfs:subClassOf :Node;
  owl:equivalentClass [ a owl:Restriction; owl:onProperty :out; owl:hasValue 2],
                      [ a owl:Restriction; owl:onProperty :edge; owl:cardinality 2].
:ThreeExitsNode a owl:Class; rdfs:subClassOf :Node;
  owl:equivalentClass [ a owl:Restriction; owl:onProperty :out; owl:hasValue 3],
                      [ a owl:Restriction; owl:onProperty :edge; owl:cardinality 3].
```

[Filename: RDF/winmove-closure.n3]

Win-Move-Game in OWL: DeadEndNodes

Prove that DeadEndNodes are LoseNodes:

- obvious: Player cannot move from there
- exercise: give a formal (Tableau) proof
- The OWL Reasoner does it:

```
prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>
prefix : <foo://bla#>
select ?X
from <file:winmove-axioms.n3>
from <file:winmove-closure.n3>
where {:DeadEndNode rdfs:subClassOf :LoseNode}
```

[Filename: RDF/deadendnodes.sparql]

The answer contains an (empty) tuple which means “yes”.

Win-Move-Game instancs solving in OWL

```
prefix : <foo://bla#>
select ?W ?L ?DE
from <file:winmove-graph.n3>
from <file:winmove-axioms.n3>
from <file:winmove-closure.n3>
where {{?W a :WinNode} UNION {?L a :LoseNode} UNION
       {?DE a :DeadEndNode}}
```

[Filename: RDF/winmove.sparql]

lose: f, k, j, e, l
win: c, a, i, b, d
The nodes g, h, and m are not contained in any of these sets → they are drawn positions.

Aside: Comparison with the Win-Move-Game in the “Deductive Databases” lecture: Solving vs. Reasoning

- With well-founded or stable semantics, concrete example cases of the win-move-game could be *solved*.
- With well-founded or stable semantics cannot make general *proofs* like DeadEndeNode $\sqsubseteq$ LoseNode.
- an OWL/DL *Reasoner* can do such *proofs*.

Exercise

- Is it possible to characterize DrawNodes in OWL?
  - 2 alternative variants:
    * using the game axioms/rules,
    * consider the possible values: win/lost/drawn,
  - test with *typical* minimal examples,
  - explain the results [DB Theory: compare also with well-founded and stable models].
- Is it possible to use SPARQL to find the drawn positions?

9.6 OWL 2 (W3C Recommendation since October 2009)

- OWL2 notions belong to the OWL namespace
  (aside: development proposal owl11 used a separate namespace)
- Syntactic Sugar: owl:disjointUnionOf, owl:AllDifferent, owl:AllDisjointClasses, owl:AllDisjointProperties, and negative assertions: ObjectPropertyAssertion vs. NegativeObjectPropertyAssertion
- User-defined datatypes (like XML Schema simple types).
- $\mathcal{SROIQ}$: Qualified cardinality restrictions (only for non-complex properties), local reflexivity restrictions (individuals that are related to themselves via the given property), reflexive, irreflexive, symmetric, and anti-symmetric properties (only for non-complex properties), disjoint properties (only for non-complex properties), Property chain inclusion axioms (e.g., SubPropertyOf(PropertyChain(owns hasPart) owns) asserts that if x owns y and y has a part z, then x owns z.
- $\mathcal{SROIQ}(D)$ is decidable.
  The Even More Irresistible $\mathcal{SROIQ}$. Ian Horrocks, Oliver Kutz, and Ulrike Sattler. In Principles of Knowledge Representation and Reasoning (KR 2006). AAAI Press, 2006. Available at `www.cs.man.ac.uk/~sattler/publications/sroiq-tr.pdf`.

Qualified Role Restrictions

- extends owl:Restriction, owl:onProperty, owl:{min/max}QualifiedCardinality (int value) with owl:on{Class/DataRange} as result class/type.

```
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix : <foo://bla#> .
:Dog a owl:Class.   :Cat a owl:Class.    :Cat owl:disjointWith :Dog.
:alice :name "Alice"; :hasAnimal :pluto, :struppi.
:john :name "John"; :hasAnimal :garfield, :odie.
:pluto a :Dog; :name "Pluto".
:struppi a :Dog; :name "Struppi".
:garfield a :Cat; :name "Garfield".
:odie a :Dog; :name "Odie".
:name a owl:FunctionalProperty.
:HasTwoAnimals owl:equivalentClass
 [a owl:Restriction; owl:onProperty :hasAnimal; owl:minCardinality 2].
:HasTwoCats owl:equivalentClass  [a owl:Restriction;
   owl:onProperty :hasAnimal; owl:onClass :Cat; owl:minQualifiedCardinality 2].
:HasTwoDogs owl:equivalentClass  [a owl:Restriction;
   owl:onProperty :hasAnimal; owl:onClass :Dog; owl:minQualifiedCardinality 2].
```

[Filename: RDF/cats-and-dogs.n3]

[Filename: RDF/cats-and-dogs.sparql]

```
prefix : <foo://bla#>
prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>
select ?X ?Y ?Z ?C   from <file:cats-and-dogs.n3>
where {{?X a :HasTwoCats} UNION
       {?Y a :HasTwoDogs} UNION
       {?Z a :HasTwoAnimals} UNION
       {?C rdfs:subClassOf :HasTwoAnimals}}
```


Qualified Role Restrictions – Another Test

```
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix : <foo://bla#> .
:alice :name "Alice"; :hasAnimal :pluto, :struppi.
:john :name "John"; :hasAnimal :garfield, :nermal, :odie.
:sue :hasAnimal :grizabella.              :grizabella :name "Grizabella".
:pluto a :Dog; :name "Pluto".             :struppi a :Dog; :name "Struppi".
:garfield a :Cat; :name "Garfield".       :nermal a :Cat; :name "Nermal".
:odie a :Dog; :name "Odie".
:name a owl:FunctionalProperty.
:Dog a owl:Class.   :Cat a owl:Class.     :Cat owl:disjointWith :Dog.
:HasAnimal owl:equivalentClass
 [a owl:Restriction; owl:onProperty :hasAnimal; owl:minCardinality 1].
:HasCat owl:equivalentClass
 [a owl:Restriction; owl:onProperty :hasAnimal; owl:onClass :Cat; owl:minQualifiedCardinality 1].
:HasDog owl:equivalentClass
 [a owl:Restriction; owl:onProperty :hasAnimal; owl:someValuesFrom :Dog].
```

[Filename: RDF/hasanimals.n3]

- export class tree:
  HasCat and HasDog are (non-disjoint) subclasses of HasAnimal.
- "owl:onClass X & owl:minQualifiedCardinality 1" is equivalent to "owl:someValuesFrom X".
- "owl:minCardinality 1" alone is equivalent to "owl:someValuesFrom owl:Thing".

Qualified Role Restrictions – Another Test

```
@prefix : <foo://bla#>.
@prefix owl: <http://www.w3.org/2002/07/owl#>.
:TwoChildren owl:equivalentClass [a owl:Restriction;
  owl:onProperty :hasChild; owl:cardinality 2].
:ThreeMaleChildren owl:equivalentClass [a owl:Restriction;
  owl:onProperty :hasChild; owl:onClass :Male; owl:minQualifiedCardinality 3].
:TCTMC owl:equivalentClass
   [ owl:intersectionOf (:TwoChildren :ThreeMaleChildren) ].
```

[Filename: RDF/twochildren-threemale.n3]

- export class tree:
- note that the ontology is not inconsistent, but that simply TCTMC is derived to be equivalent to owl:Nothing.

OWL: Disjoint Union, AllDisjointClasses

... syntactic sugar for owl:unionOf and owl:disjointWith:

(only a simple test and syntax example for RDF/XML)

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:owl="http://www.w3.org/2002/07/owl#"
   xmlns:f="foo://bla#"
   xml:base="foo://bla#">
 <owl:Class rdf:about="Person">
  <owl:disjointUnionOf rdf:parseType="Collection">
   <owl:Class rdf:about="Male"/>
   <owl:Class rdf:about="Female"/>
  </owl:disjointUnionOf>
 </owl:Class>
 <f:Male rdf:about="John"/>
 <f:Female rdf:about="Mary"/>
 <!--<f:Female rdf:about="John"/>-->
</rdf:RDF>
```

[Filename: RDF/disjointunion.xml]

```
prefix f: <foo://bla#>
select ?X
from <file:disjointunion.xml>
where {?X a f:Person}
```

[Filename: RDF/disjointunion.sparql]

OWL: ALLDISJOINTCLASSES

- General Case without union (similar to owl:AllDifferent):
  [a owl:AllDisjointClasses; owl:members (...)]
- Typical usages:
  - typically used if subclasses are disjoint specializations, but not every element of the superclass is an element of one of the specializations.
  - for the Top classes of an ontology.

EXAMPLE: PARRICIDES IN GREEK MYTHOLOGY

(from ESWC'07 SPARQL tutorial by Marcelo Arenas et al)

A parricide is a person who killed his/her father.

```
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix : <foo:greek#>.
:Person owl:disjointUnionOf (:Parricide :Non-Parricide).
  :iokaste a :Person; :hasChild :oedipus.
  :oedipus a :Person, :Parricide; :married-to :iokaste; :hasChild :perineikes.
  :perineikes a :Person; :hasChild :thesandros.
  :thesandros a :Person; a :Non-Parricide.
:Parent-of-Parricide owl:equivalentClass [ a owl:Restriction;
  owl:onProperty :hasChild; owl:someValuesFrom :Parricide ].
:Parent-of-Non-Parricide owl:equivalentClass [ a owl:Restriction;
  owl:onProperty :hasChild; owl:someValuesFrom :Non-Parricide ].
:Parent-of-Parricide-Grandparent-of-Non-Parricide owl:intersectionOf
  ([a owl:Restriction; owl:onProperty :hasChild; owl:someValuesFrom :Parricide]
   [a owl:Restriction;
     owl:onProperty :hasChild; owl:someValuesFrom :Parent-of-Non-Parricide]).
```

[Filename: RDF/parricide.n3]

Example (Cont'd)

- have a short look on the results:

```
prefix : <foo:greek#>
prefix owl: <http://www.w3.org/2002/07/owl#>
select ?P ?NP ?PP ?PNP ?X
from <file:parricide.n3>
where {{?P a :Parricide} UNION
       {?NP a :Non-Parricide} UNION
       {?PP a :Parent-of-Parricide} UNION
       {?PNP a :Parent-of-Non-Parricide} UNION
       {?X a :Parent-of-Parricide-Grandparent-of-Non-Parricide}}
```

[Filename: RDF/parricide.sparql]

- No X reported.

Example (Cont'd)

- ask Zeus whether Parent-of-Parricide-Grandparent-of-Non-Parricide is really non-empty:

```
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix : <foo:greek#>.
:zeus :knows :iokaste, :oedipus, :perineikes, :thesandros.
:KnowsPoPGonP owl:equivalentClass [ a owl:Restriction; owl:onProperty :knows;
  owl:someValuesFrom :Parent-of-Parricide-Grandparent-of-Non-Parricide ].
```

[Filename: RDF/parricide2.n3]

```
prefix : <foo:greek#>
select ?K ?X
from <file:parricide.n3>
from <file:parricide2.n3>
where {{?K a :KnowsPoPGonP} UNION
       {?X a :Parent-of-Parricide-Grandparent-of-Non-Parricide}}
```

[Filename: RDF/parricide2.sparql]

- Zeus is in K, i.e., he knows such a person (explicitly: he knows a person who must be a P.o.p.G.o.N.P),
- but neither SPARQL, nor Zeus know who that person is,
- it can be either Iokaste or Oedipus (depending on whether Perineikes is a parricide, which nobody knows).

Example (Cont'd) – Exercise

Consider *absolutely strictly* the answer to `parricide2.sparql`.

- What has actually been logically proven by the answer?
- What *additional* human reasoning took place in the lecture when *interpreting* the answer to `parricide2.sparql` as "it can be either Iokaste or Oedipusa (depending on whether Perineikes is a parricide, which nobody knows)".
- complete the ontology and the SPARQL query in a way that the human resoning conclusions are mirrored in the setting.

NEGATIVE ASSERTIONS

- Assert that something is known *not* to hold: NegativeObjectPropertyAssertion and NegativeDataPropertyAssertion
- with owl:sourceIndividual, owl:assertionProperty, and owl:targetIndividual or owl:targetValue.

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix : <foo://bla#> .
:john a :Person.
[ rdf:type owl:NegativePropertyAssertion;
  owl:sourceIndividual :john;
  owl:assertionProperty :lives;
  owl:targetIndividual :germany].
:German owl:equivalentClass [ a owl:Restriction;
  owl:onProperty :lives; owl:hasValue :germany ].
:NonGerman owl:complementOf :German.
```

```
prefix : <foo://bla#>
select ?P
from <file:nongerman.n3>
where {?P a :NonGerman}
```

[Filename: RDF/nongerman.sparql]

[Filename: RDF/nongerman.n3]

- John is derived to be a Non-German.

Comment on Negative Assertions

... are just syntactic sugar for a construct using complement classes (and actually implemented in the reasoner by this):

Any owl:NegativeObjectPropertyAssertion $\neg(x\ r\ y)$ is encoded as

- a restriction $R(r, y)$ based on owl:hasValue:
  $R(r, y) = \{x | (x\ r\ y)\}$
  (above: R(lives,germany) = :German)
- its complement $CompR(r, y) := \top \setminus R(r, y)$
  (above: CompR(lives,germany) = :NonGerman)
- and the assertion that $x \in CompR(r, y)$.
  (above: assert (:john a :NonGerman))

DATATYPES: HASVALUE WITH LITERAL VALUE

Characterize a class as the set of all things where a given property has a given value:

- all things in Mondial that have the name “Berlin”:

```
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix mon: <http://www.semwebtech.org/mondial/10/meta#>.
@prefix : <foo:bla#>.
:Berlin owl:equivalentClass [ a owl:Restriction;
   owl:onProperty mon:name; owl:hasValue "Berlin"].
```

[Filename: RDF/has-literal-value.n3]

```
prefix : <foo:bla#>
select ?X
from <file:has-literal-value.n3>
from <file:mondial-europe.n3>
where {?X a :Berlin}
```

[Filename: RDF/has-literal-value.sparql]

- Often preferable: define an owl:DataRange (unary or enumeration), give it a url, and use some/allValuesFrom.

Enumerated Datatypes

```
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix uni: <foo://uni/>.
uni:graded a owl:FunctionalProperty;
 a owl:DatatypeProperty; rdfs:range uni:Grades.
uni:Grades a rdfs:Datatype;
 owl:equivalentClass [ a rdfs:Datatype;
  owl:oneOf ("1.0" "1.3" "1.7" "2.0" "2.3" "2.7" "3.0" "3.3" "3.7" "4.0") ] .
[ a uni:Thesis; uni:author <foo://bla/john>;
  uni:graded "2.5"].
```

[Filename: RDF/grades-one-of-namedset.n3]

- inconsistent: “2.5” does not belong to the allowed grades,
- note: “3” is also not allowed since “3” and “3.0” are different strings,
- see alternative next slide.

Enumerated Datatypes

```
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix uni: <foo://uni/>.
uni:graded a owl:FunctionalProperty;
 a owl:DatatypeProperty; rdfs:range [ a rdfs:Datatype;
  owl:oneOf (1 1.3 1.7 2.0 2.3 2.7 3 3.3 3.7 4) ] .
[ a uni:Thesis; uni:author <foo://bla/john>;
  uni:graded 2].
```

[Filename: RDF/grades-one-of-anonymous.n3]

```
prefix : <foo://uni/>
select ?X ?G
from <file:grades-one-of-anonymous.n3>
where {?X :graded ?G}
```

[Filename: RDF/grades-one-of-anonymous.sparql]

- grade 2.5 results in an inconsistency,
- internally (in case of an error message e.g.), the values are represented/handled as “2.3”^^xsd:decimal,
- parsing and output uses the default representation,
- both representations 2 and 2.0 are allowed.

ONEOF ON DATARANGE

```
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.
@prefix : <foo://bla/meta#>.
:Male a owl:Class.     :Female a owl:Class.
:Person owl:disjointUnionOf (:Male :Female).
:MaleNames a rdfs:Datatype; owl:equivalentClass [ a rdfs:Datatype;
     owl:oneOf ("John"^^xsd:string "Bob"^^xsd:string) ] .
:FemaleNames a rdfs:Datatype; owl:equivalentClass [ a rdfs:Datatype;
    owl:oneOf ("Alice"^^xsd:string "Carol"^^xsd:string) ].
:Male a owl:Class; owl:equivalentClass [owl:intersectionOf ( :Person
  [a owl:Restriction; owl:onProperty :name; owl:someValuesFrom :MaleNames])].
:Female a owl:Class; owl:equivalentClass [owl:intersectionOf ( :Person
  [a owl:Restriction; owl:onProperty :name; owl:someValuesFrom :FemaleNames])].
:name a owl:FunctionalProperty; a owl:DatatypeProperty.
:john a :Person; :name "John"^^xsd:string.
:alice a :Person; :name "Alice"^^xsd:string.
```

[Filename: RDF/names.n3]

```
prefix : <foo://bla/meta#>
select ?C ?N
from <file:names.n3>
where {:john a ?C ; :name ?N}
```

[Filename: RDF/names.sparql]

Exercise

Consider again the ontology from the previous slide

- The name “Maria” is a female first name, but (mainly by catholics) also used as an additional first name for males, e.g. Rainer Maria Rilke (German poet, 1875-1926), José Maria Aznar (*1956, Spanish Prime Minister 1996-2004). Discuss the consequences on the ontology.
- Check what happens with names like “Kim” that can be both male and Female names.

REIFICATION

Reification: treat a class (or a property or a statement) as a thing:

- Male and Female are both classes and instances of class Sex

```
@prefix rdfs: <http://www.w3.org/2000/01/rdf-
@prefix owl: <http://www.w3.org/2002/07/owl#>
@prefix : <foo://bla/meta#>.
:Person owl:disjointUnionOf (:Male :Female).
:Male a :Sex.
:Female a :Sex.
:MaleNames owl:equivalentClass [ a rdfs:Datatype; owl:oneOf ("John" "Bob") ] .
:FemaleNames owl:equivalentClass [ a rdfs:Datatype; owl:oneOf ("Mary" "Alice") ].
:Male a owl:Class; owl:equivalentClass [owl:intersectionOf ( :Person
  [a owl:Restriction; owl:onProperty :name; owl:someValuesFrom :MaleNames])].
:Female a owl:Class; owl:equivalentClass [owl:intersectionOf ( :Person
  [a owl:Restriction; owl:onProperty :name; owl:someValuesFrom :FemaleNames])].
:name a owl:FunctionalProperty; a owl:DatatypeProperty.
:john a :Person; :name "John".
:mary a :Person; :name "Mary".
```

[Filename: RDF/reification-class.n3]

```
prefix : <foo://bla/meta#>
select ?P ?N ?S
from <file:reification-class.n3>
where {{?S a :Sex .
        ?P a :Person ; a ?S ; :name ?N}}
```

[Filename: RDF/reification-class.sparql]

DATATYPES

- common built-ins from XML Schema: int, decimal, ..., date, time, datetime.
- “2”^^xsd:decimal is different from “2”^^xsd:int

```
prefix : <foo:bla#>
select ?X ?Y
from <file:decimal.n3>
where {?X :value ?Y}
```

[Filename: RDF/decimal.sparql]

```
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.
@prefix : <foo:bla#>.
:value a owl:DatatypeProperty; rdfs:range xsd:decimal.
:foo :value "2"^^xsd:decimal; :value "1.0"^^xsd:decimal.
:foo :value "2.0"^^xsd:decimal; :value "2.3"^^xsd:decimal.
:foo :value "2"^^xsd:integer; :value "1"^^xsd:integer.
```

[Filename: RDF/decimal.n3]

- jena: returns 6 results: “2”^^xsd:decimal, 1.0, 2.0, 2.3, 1, 2
- pellet: returns 5 results: 1, 2, 2.3, 2.0, 1.0

DEFINING OWN DATATYPES

Two possibilities:

- use XML Schema xsd:simpleType definitions on the Web:
  - OWL reasoners parse+understand XML Schema simpleType declarations
  - adopt the DAML+OIL solution: datatype URI is constructed from the URI of the XML schema document and the local name of the simple type.
- OWL vocabulary to do the same as in XML Schema simpleTypes.

DATATYPES IN OWL

- use the XML Schema built-in types as resources (int and string must be supported; Pellet does also support decimal)
- rdfs:Datatype: cf. simple Types in XML schema; derived from the basic ones (e.g. xsd:int is an rdfs:Datatype)
- specified by
  - owl:onDatatype: from what datatype they are derived,
  - owl:withRestrictions is a list of restricting facets
  - facets as in XML Schema:
    xsd:{max/min}{In/Ex}clusive etc.
- similar to owl:Restrictions: define by
  myDatatypeName owl:equivalentClass [*datatypeSpec*].

Data Ranges: Adults

```
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xsd:  <http://www.w3.org/2001/XMLSchema#>.
@prefix : <foo://bla#> .
:kate :name "Kate"; :age 62; :hasChild :john.
:john :name "John"; :age 35; :hasChild [:name "Alice"], [:name "Bob"; :age 8].
:hasChild rdfs:domain :Person; rdfs:range :Person.
:age a owl:FunctionalProperty; a owl:DatatypeProperty; rdfs:range xsd:int.
:name a owl:FunctionalProperty; a owl:DatatypeProperty; rdfs:range xsd:string.
:atLeast18T owl:equivalentClass
  [a rdfs:Datatype; owl:onDatatype xsd:int; owl:withRestrictions ( _:x1 )].
_:x1 xsd:minInclusive 18 .
:Adult owl:intersectionOf (:Person
  [ a owl:Restriction;
    owl:onProperty :age;
    owl:someValuesFrom :atLeast18T]).
:Child owl:intersectionOf (:Person
  [ owl:complementOf :Adult ]).
```

[Filename: RDF/adult.n3]

```
prefix : <foo://bla#>
select ?AN ?CN ?X ?Y
from <file:adult.n3>
where {{?A a :Adult; :name ?AN} UNION
       {?C a :Child; :name ?CN} UNION
       {?X :age ?Y}}
```

[Filename: RDF/adult.sparql]

An Example with Two QRRs

```
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix xsd:  <http://www.w3.org/2001/XMLSchema#>.
@prefix : <foo://bla#> .
:kate :name "Kate"; :age 62; :hasChild :john, :sue.
:sue :name "Sue";  :age 32; :hasChild [:name "Barbara"].
:john :name "John";  :age 35;
      :hasChild :alice, [:name "Bob"; :age 8], [:name "Alice"; :age 10].
:frank :name "Frank"; :age 40; :hasChild [:age 18], [:age 13].
:hasChild rdfs:domain :Person; rdfs:range :Person.
:age a owl:FunctionalProperty; a owl:DatatypeProperty; rdfs:range xsd:int.
:name a owl:FunctionalProperty; a owl:DatatypeProperty; rdfs:range xsd:string.
:atLeast18T owl:equivalentClass [a rdfs:Datatype;
  owl:onDatatype xsd:int; owl:withRestrictions ( [ xsd:minInclusive 18 ] ) ].
:Adult owl:intersectionOf (:Person
  [ a owl:Restriction; owl:onProperty :age; owl:someValuesFrom :atLeast18T]).
:HasTwoAdultChildren  owl:equivalentClass [ a owl:Restriction;
    owl:onProperty :hasChild; owl:onClass :Adult; owl:minCardinality 2 ].
```

[Filename: RDF/adultchildren.n3]

```
prefix : <foo://bla#>
select ?AN ?N
from <file:adultchildren.n3>
where {{?A a :Adult; :name ?AN} UNION
      {?X a :HasTwoAdultChildren; :name ?N}}
```

[Filename: RDF/adultchildren.sparql]

DataRange Restriction for Geographical Coordinates

```
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix mon: <http://www.semwebtech.org/mondial/10/meta#>
@prefix xsd:  <http://www.w3.org/2001/XMLSchema#>.
@prefix : <foo://bla/>.
:LongitudeT owl:equivalentClass [ a rdfs:Datatype; owl:onDatatype xsd:decimal;
   owl:withRestrictions ( [xsd:minExclusive -180] [xsd:maxInclusive 180] ) ] .
:LatitudeT owl:equivalentClass [ a rdfs:Datatype; owl:onDatatype xsd:decimal;
  owl:withRestrictions ( [ xsd:minInclusive -90] [xsd:maxInclusive 90] ) ] .
:EasternLongitudeT  owl:equivalentClass [a rdfs:Datatype;
  owl:onDatatype :LongitudeT; owl:withRestrictions ( [xsd:minInclusive 0] ) ] .
:EasternHemispherePlace owl:equivalentClass [a owl:Restriction;
  owl:onProperty mon:longitude; owl:someValuesFrom :EasternLongitudeT].
mon:longitude rdfs:range :LongitudeT.
mon:latitude rdfs:range :LatitudeT.
:Berlin a mon:City; :name "Berlin"; mon:longitude 13.3; mon:latitude 52.45 .
#:Atlantis a mon:City; :name "Atlantis"; mon:longitude -200; mon:latitude 100 .
:Lisbon a mon:City; :name "Lisbon"; mon:longitude -9.1; mon:latitude 38.7 .
```

[Filename: RDF/coordinates.n3]

```
prefix : <foo://bla/>
select ?N
from <file:coordinates.n3>
where {?X :name ?N .
  ?X a :EasternHemispherePlace}
```

[Filename: RDF/coordinates.sparql]

Example: Using XSD Datatypes

- [Does not work completely ...] Define simple datatypes in an XML Schema file:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  targetNamespace="file:coordinates2.xsd">
 <xs:simpleType name="longitudeT">
  <xs:restriction base="xs:decimal">
   <xs:minExclusive value="-180"/>
   <xs:maxInclusive value="180"/>
  </xs:restriction>
 </xs:simpleType>
 <xs:simpleType name="easternLongitude">
  <xs:restriction base="xs:decimal">
    <!-- note: base="longitudeT" would be nicer, but is not allowed when parsing from RDF -->
   <xs:minInclusive value="10"/>
   <xs:maxInclusive value="180"/>
  </xs:restriction>
 </xs:simpleType>
 <xs:simpleType name="latitudeT">
  <xs:restriction base="xs:decimal">
   <xs:minInclusive value="-90"/>
   <xs:maxInclusive value="90"/>
  </xs:restriction>
 </xs:simpleType>
</xs:schema>
```

[Filename: RDF/coordinates2.xsd]

... and now use the datatypes ...

```
<!DOCTYPE rdf:RDF [ <!ENTITY mon "http://www.semwebtech.org/mondial/10/meta#">
  <!ENTITY xsd "http://www.w3.org/2001/XMLSchema">
  <!ENTITY Coords "file:coordinates2.xsd"> ]>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
   xmlns:owl="http://www.w3.org/2002/07/owl#"
   xmlns:mon="http://www.semwebtech.org/mondial/10/meta#">

 <!-- ****** IMPORTANT: ALL DATATYPES MUST BE MENTIONED TO BE PARSED ****** -->
 <rdfs:Datatype rdf:about="&Coords;#longitudeT"/>
 <rdfs:Datatype rdf:about="&Coords;#easternLongitude"/>
 <rdfs:Datatype rdf:about="&Coords;#latitudeT"/>
 <owl:Class rdf:about="&mon;EasternHemispherePlace">
 <owl:equivalentClass> <!-- again: don't give a uri to an owl:Restriction! -->
  <owl:Restriction>
   <owl:onProperty rdf:resource="&mon;longitude"/>
   <owl:someValuesFrom rdf:resource="&Coords;#easternLongitude"/>
  </owl:Restriction>
 </owl:equivalentClass>
 </owl:Class>

 <mon:City mon:name="Berlin">
  <mon:longitude rdf:datatype="&Coords;#longitudeT">13.3</mon:longitude>
  <mon:latitude rdf:datatype="&Coords;#latitudeT">52.45</mon:latitude> </mon:City>
 <mon:City mon:name="Lisbon">
  <mon:longitude rdf:datatype="&Coords;#longitudeT">-9.1</mon:longitude>
  <mon:latitude rdf:datatype="&Coords;#latitudeT">38.7</mon:latitude> </mon:City>
</rdf:RDF>
```

[Filename: RDF/coordinates2.rdf]

... and now to the query:

```
prefix : <http://www.semwebtech.org/mondial/10/meta#>
select ?N
from <file:coordinates2.rdf>
where {?X :name ?N . ?X a :EasternHemispherePlace}
```

[Filename: RDF/coordinates2.sparql]

Comments

- the RDF file must “define” all used rdf:Datatypes to be parsed from the XML Schema file. (if <rdfs:Datatype rdf:about=“&Coords;#easternLongitude”/> is omitted, the result is empty)
- if a prohibited value, e.g. longitude=200 is given in the RDF file, it is rejected.
- the rdf:Datatype for mon:longitude and mon:latitude must be given, otherwise it is not recognized as a number (but it does not matter if xsd:int or coords:longitude is used).
- specifying rdfs:range for longitude and latitude *without* rdf:Datatype for mon:longitude and mon:latitude is even inconsistent!

Qualified Role Restrictions: Example

Example: Country with at least two cities with more than a million inhabitants.

- define “more than a million” as a rdfs:Datatype
- search for all BigCities (= more than 1000000 inhabitants)
- check -via Provinces- which countries have two such cities.

Example: Cont’d

```
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix mon: <http://www.semwebtech.org/mondial/10/meta#>.
@prefix xsd:  <http://www.w3.org/2001/XMLSchema#>.
@prefix : <foo://bla/>.

mon:population  rdfs:range xsd:int; a owl:FunctionalProperty. ## all cities are different.
_:Million a rdfs:Datatype; owl:onDatatype xsd:int; owl:withRestrictions (_:m1).
_:m1 xsd:minInclusive 1000000 .
:HasBigPopulation owl:equivalentClass [a owl:Restriction;
  owl:onProperty mon:population; owl:someValuesFrom _:Million].
:BigCity owl:intersectionOf (mon:City :HasBigPopulation).
:ProvinceWithBigCity owl:intersectionOf (mon:Province
  [a owl:Restriction; owl:onProperty mon:hasCity; owl:someValuesFrom :BigCity]).
:ProvinceWithTwoBigCities owl:intersectionOf (mon:Province ## europe: empty
  [a owl:Restriction; owl:onProperty mon:hasCity; owl:onClass :BigCity; owl:minCardinality 2]).
[owl:intersectionOf (mon:Country ## with 2 big cities, no provinces ## europe: empty
  [a owl:Restriction; owl:onProperty mon:hasCity; owl:onClass :BigCity; owl:minCardinality 2]);
  rdfs:subClassOf :CountryWithTwoBigCities].
[owl:intersectionOf (mon:Country ## with 2 provs with big cities ## TR,GB,E,R,UA,D,I,NL
  [a owl:Restriction; owl:onProperty mon:hasProvince; owl:onClass :ProvinceWithBigCity; owl:minCardinality 2]);
  rdfs:subClassOf :CountryWithTwoBigCities].
[owl:intersectionOf (mon:Country ## with a prov with 2 big cities ## europe: empty
  [a owl:Restriction; owl:onProperty mon:hasProvince; owl:someValuesFrom :ProvinceWithTwoBigCities]);
   rdfs:subClassOf :CountryWithTwoBigCities].
```

[Filename: RDF/bigcities.n3]

Example: Cont'd

```
@prefix mon: <http://www.semwebtech.org/mondial/10/meta#>.
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix : <foo://bla/>.
:grmny a mon:Country; mon:hasCity :bln, :mch .
:bln a :BigCity; mon:population 3500000 .
:mch a :BigCity; mon:population 1500000 .
:frc a mon:Country; mon:hasProvince :ile, :prov .
:ile owl:differentFrom :prov.
:prs a mon:City; mon:cityIn :ile; mon:population 2000000 .
:mrs a mon:City; mon:cityIn :prov; mon:population 1500000 .
```

[Filename: RDF/dummy-cities.n3]

```
prefix : <foo://bla/>
select ?BC ?P1 ?P2 ?X
from <file:bigcities.n3>
#from <file:dummy-cities.n3>
from <file:mondial-europe.n3>
from <file:mondial-meta.n3>
where {# {?BC a :BigCity} UNION
       # {?P1 a :ProvinceWithBigCity} UNION
       # {?P2 a :ProvinceWithTwoBigCities} UNION
       {?X a :CountryWithTwoBigCities}}
```

[Filename: RDF/bigcities.sparql]

9.7 OWL 2: More about Properties

- $\mathcal{SHIQ}$/OWL-DL concentrate on *concept* definitions ($\mathcal{SQ}$ portion),
  - The $\mathcal{H}$ allows for a hierarchy of *properties* as already provided by RDFS, the $\mathcal{I}$ allows for inverse.
- $\mathcal{SHOIQ}/\mathcal{SHOIQ}(\mathcal{D})$ add nominals and datatypes (i.e., provide database-oriented functionality for handling *instances*),
- $\mathcal{SROIQ}$ provides more expressiveness around *properties*.

Transitive and Symmetric Properties

- transitive: descendants (cf. Slide 237), train connections etc.
- symmetric: married

```
@prefix : <foo://bla#>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix owl: <http://www.w3.org/2002/07/owl#>.
  [ :name "John";  :married [ :name "Mary" ]] .
  :married rdf:type owl:SymmetricProperty.
```

[Filename: RDF/symmetric-married.n3]

```
prefix : <foo://bla#>
select ?X ?Y
from <file:symmetric-married.n3>
where { [ :name ?X ; :married [ :name ?Y]] }
```

[Filename: RDF/symmetric-married.sparql]

Symmetric Properties

```
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix : <foo://bla#> .
:germany :borders :austria, :switzerland.
:borders a owl:SymmetricProperty.
```

[Filename: RDF/symmetricborders.n3]

```
prefix : <foo://bla#>
select ?X ?Y
from <file:symmetricborders.n3>
where {?X :borders ?Y}
```

[Filename: RDF/symmetricborders.sparql]

Reflexive Properties (OWL 2)

```
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix : <foo://bla#> .
:john a :Person; :knows :mary; :hasChild :alice.
:knows a owl:ReflexiveProperty.
:germany a :Country.
```

[Filename: RDF/reflexive.n3]

```
prefix : <foo://bla#>
select ?X ?Y
from <file:reflexive.n3>
where {?X :knows ?Y}
```

[Filename: RDF/reflexive.sparql]

- only applied to individuals, but ... to all of them:
  John knows John, Alice knows Alice, and Germany knows Germany.

Irreflexive Properties

- irreflexive(rel): $\forall x : \neg rel(x, x)$.
- acts as constraint,
- but can also induce that two things must be different:
  $\forall x, y : rel(x, y) \rightarrow x \neq y$

```
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix : <foo://bla#> .
:john :hasAnimal :pluto, :garfield.
:pluto :bites :garfield.
# we exclude neurotic animals:
:bites a owl:IrreflexiveProperty.
:HasTwoAnimals  owl:equivalentClass
  [ a owl:Restriction;
    owl:onProperty :hasAnimal; owl:minCardinality 2 ].
```

[Filename: RDF/irreflexive.n3]

```
prefix : <foo://bla#>
select ?X ?Y ?Z
from <file:irreflexive.n3>
where {{?X :bites ?Y} UNION
       {?X :bites ?X} UNION
       {?Z a :HasTwoAnimals}}
```

[Filename: RDF/irreflexive.sparql]

- Pluto cannot be the same as Garfield.

Asymmetry

- asymmetric(rel): $\forall x, y : \neg rel(x, y) \vee \neg rel(y, x)$).
- acts as a constraint.

```
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix : <foo://bla#>.
:rel a owl:AsymmetricProperty.
:a a :Node; :rel :b.
:b a :Node; :rel :c.
:c a :Node.
# :a owl:sameAs :b.
```

[Filename: RDF/asymmetry.n3]

```
prefix owl: <http://www.w3.org/2002/07/owl#>
prefix : <foo://bla#>
select ?X ?Y
from <file:asymmetry.n3>
where {?X a :Node; owl:differentFrom ?Y . ?Y a :Node}
```

[Filename: RDF/asymmetry.sparql]

- a,b,c, are not identified to be different, but any owl:sameAs makes the ontology inconsistent.

Irreflexive and Asymmetric Properties

- Motivated by the “Ascending, Descending” graphics by M.C.Escher
  http://en.wikipedia.org/wiki/Ascending_and_Descending

```
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix : <foo://bla#>.
:Corner owl:oneOf (:a :b :c); rdfs:subClassOf
  [a owl:Restriction; owl:onProperty :higher; owl:cardinality 1].
:higher rdfs:domain :Corner; rdfs:range :Corner.
#:higher a owl:FunctionalProperty. ## redundant, note cardinality 1
:higher a owl:InverseFunctionalProperty. # necessary if there are more corners
:higher a owl:AsymmetricProperty; a owl:IrreflexiveProperty.
:a :higher :b.
```

[Filename: RDF/escherstairs.n3]

```
prefix : <foo://bla#>
select ?X ?Y
from <file:escherstairs.n3>
where {?X :higher ?Y}
```

[Filename: RDF/escherstairs.sparql]

- Solution: $a > b$, $b > c$, $c > a$ is the only model.

Exercise

- what happens when the above program is the extended to four corners (:a :b :c :d)? Analyze the result also from the logical point of view.

Disjoint Properties

- Syntax: ($prop_1$ owl:propertyDisjointWith $prop_2$)
- for more than 2 properties (similar to owl:AllDifferent):
  [a owl:AllDisjointProperties; owl:members (...)]

```
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix : <foo://bla#>.

:name a owl:FunctionalProperty.
:hasCat rdfs:subPropertyOf :hasAnimal; rdfs:range :Cat.
:hasDog rdfs:subPropertyOf :hasAnimal; rdfs:range :Dog.
:hasCat owl:propertyDisjointWith :hasDog.

:alice :name "Alice"; :hasDog :pluto, :struppi.
:john :name "John"; :hasCat :garfield, :nermal; :hasDog :odie.
:sue :hasCat :grizabella.
#:sue :hasDog :grizabella.   ### test ########
:pluto a :Dog; :name "Pluto".
:struppi a :Dog; :name "Struppi".
:garfield a :Cat; :name "Garfield".
:nermal a :Cat; :name "Nermal".
:odie a :Dog; :name "Odie".
:grizabella :name "Grizabella".
```

[Filename: RDF/disjoint-properties.n3]

```
prefix : <foo://bla#>
select ?A ?B ?C ?D ?E ?F
from <file:disjoint-properties.n3>
where {{?X :name ?A; :hasCat/:name ?B} UNION
       {?X :name ?C; :hasDog/:name ?D} UNION
       {?X :name ?E; :hasAnimal/:name ?F}}
```

[Filename: RDF/disjoint-properties.sparql]

AT THE DECIDABILITY BORDER

Some combinations of advanced constructs in DL that are part of OWL 2 are not even decidable:

- $\mathcal{ALC}_{reg}$ with transitivity, composition and union is EXPTIME-complete
- the same when inverse roles and even cardinalities for *atomic* roles ($\mathcal{ALCQI}_{reg}$) are added (recall that inverse and transitive closure are important concepts in ontologies).
- The combination of *non-atomic* roles with cardinalities is in general undecidable.
- The same holds for Role-Value-Maps. Decidability is obtained only for Role-Value-Maps over *functional* roles.

CARDINALITIES ON ATOMIC ROLES

- a city can be the capital of at most one country (but also of one or more provinces)

```
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix : <http://www.semwebtech.org/mondial/10/meta#>.
@prefix mon: <http://www.semwebtech.org/mondial/10/>.

:City a owl:Class; owl:equivalentClass
 [a owl:Restriction; owl:onProperty :isCapitalOf;
    owl:onClass :Country; owl:maxCardinality 1 ].

:name a owl:FunctionalProperty.
mon:C-Oslo a :City;
  :isCapitalOf mon:Norway, mon:P-Akershus, mon:P-Oslo.
mon:P-Akershus a :Province; :name "Akershus".
mon:P-Oslo a :Province; :name "Oslo".
mon:Norway a :Country; :name "Norway".
# mon:C-Oslo :isCapitalOf :foo.  :foo a :Country; :name "Foo".
```

[Filename: RDF/one-capital.n3]

- use jena -e to export class/instance tree

ACROSS THE DECIDABILITY BORDER

- Cardinality restrictions on complex (e.g. transitive) properties are not allowed (undecidable) ⇒ rejected by the reasoner

Every city can be located in several provinces, but these must belong to the same country.

```
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix : <http://www.semwebtech.org/mondial/10/meta#>.
@prefix mon: <http://www.semwebtech.org/mondial/10/>.

# Countries, Provinces, Cities:
:cityIn rdfs:subPropertyOf :belongsTo; rdfs:range :Province.
:isProvinceOf a owl:FunctionalProperty; rdfs:range :Country; rdfs:subPropertyOf :belongsTo.
:belongsTo a owl:TransitiveProperty; owl:inverseOf :hasProvOrCity.    # << trans.Prop <<<

:City a owl:Class; owl:equivalentClass
 [a owl:Restriction; owl:onProperty :belongsTo;                       # << cardinality <<
    owl:onClass :Country; owl:maxCardinality 1].

:name a owl:FunctionalProperty.
mon:C-Oslo a :City; :cityIn mon:P-Akershus, mon:P-Oslo.
mon:Norway a :Country; :name "Norway".
mon:P-Akershus a :Province; :isProvinceOf mon:Norway; :name "Akershus".
mon:P-Oslo a :Province; :isProvinceOf mon:Norway; :name "Oslo".
# mon:C-Oslo :isCapitalOf :foo.  :foo a :Country; :name "Foo".
```

[Filename: RDF/one-country.n3]

Detection of Potentially Undecidable Situations

Pellet does not accept combinations that can potentially be undecidable

The ontology is rejected by Pellet:

- Unsupported axiom: Ignoring transitivity axiom due to an existing cardinality restriction for property `http://www.semwebtech.org/mondial/10/meta#belongsTo`
- It is also rejected if

  ```
  :cityIn a owl:FunctionalProperty.
  :isProvinceOf a owl:FunctionalProperty.
  ```

  is added (which guarantees decidability).

Further Features of OWL 2

- Role Chains/Property Chains: SubPropertyOf(PropertyChain(owns hasPart) owns) asserts that if x owns y and y has a part z, then x owns z.

  SubPropertyOf(PropertyChain(parent brother) uncle)
  asserts that the relationship "uncle" is a superset of "parent ∘ brother", i.e., the brothers of my parents are my uncles.

- Cross-property restrictions/role-value maps:
  (cf. draft at `http://www.w3.org/Submission/owl11-overview/`)
  - ObjectAllValuesFrom(likes knows =) describes the class of individuals who like all people they know (in DL syntax: the concept defined by the role value map (X.knows ⊑ X.likes)).
  - DataSomeValuesFrom(shoeSize IQ greaterThan)
    describes the class of individuals whose shoeSize is greater than their IQ (in DL syntax: the concept defined by the role value map (X.shoeSize>X.IQ)).

Role Chains

- (brotherOf ∘ hasChild) ⊑ uncleOf

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix : <foo://bla#> .

:name a owl:FunctionalProperty.
[ owl:propertyChain (:brotherOf :hasChild)]
  rdfs:subPropertyOf :uncleOf.
:john a :Person; :brotherOf :sue.
:sue a :Person; :hasChild :anne, :barbara.
:anne :name "Anne".   :barbara :name "Barbara".
```

[Filename: RDF/uncle.n3]

```
prefix : <foo://bla#>
select ?U ?X
from <file:uncle.n3>
where {?U :uncleOf ?X}
```

[Filename: RDF/uncle.sparql]

Exercise

- Extend the above example: the husbands of sisters of parents of x are also x's uncles.

Syntax: Role Chains in RDF/XML

... as expected: a blank node that refers to an rdf:List which is an owl:subPropertyOf another property.

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
   xmlns:owl="http://www.w3.org/2002/07/owl#"
   xmlns="foo://bla#"
   xml:base="foo://bla#">
 <rdf:Description>
   <rdfs:subPropertyOf rdf:resource="#uncleOf"/>
   <owl:propertyChain>
     <rdf:List>
       <rdf:rest rdf:parseType="Collection">
         <owl:ObjectProperty rdf:about="#child"/>
       </rdf:rest>
       <rdf:first rdf:resource="#brotherOf"/>
     </rdf:List>
   </owl:propertyChain>
 </rdf:Description>
 <Person rdf:ID="sue">
  <child rdf:resource="#anne"/>
  <child rdf:resource="#barbara"/>
  <brother rdf:resource="#john"/>
 </Person>
 <Person rdf:ID="john">
  <brotherOf rdf:resource="#sue"/>
 </Person>
</rdf:RDF>
```

[Filename: RDF/uncle.rdf]

```
prefix : <foo://bla#>
select ?U ?X
from <file:uncle.rdf>
where {?U :uncleOf ?X}
```

[Filename: RDF/uncle2.sparql]

Role Chains

- propertyChains with 3 or more elements are allowed:

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix : <foo://bla#> .

[ owl:propertyChain (:brotherOf :hasChild)]
  rdfs:subPropertyOf :uncleOf.
[ owl:propertyChain (:parent :brotherOf :hasChild)]
  rdfs:subPropertyOf :cousinOf.
# [ owl:propertyChain (:father)] rdfs:subPropertyOf :parent.  ## complains
# [ :uncleOf rdfs:subPropertyOf owl:propertyChain (:brotherOf :hasChild)]
#      is also not allowed (nullpointer error from inside pellet!)
:name a owl:FunctionalProperty.
:john a :Person; :brotherOf :sue.
:bob :parent :john.
:sue a :Person; :hasChild :anne, :barbara.
:anne :name "Anne".   :barbara :name "Barbara".
```

[Filename: RDF/propchain3-family.n3]

```
prefix : <foo://bla#>
select ?U ?X ?C
from <file:propchain3-family.n3>
where {{?U :uncleOf ?X}
       union {?C :cousinOf ?X}}
```

[Filename: RDF/propchain3-family.sparql]

Undecidable: Role Chains and Cardinalities

```
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix : <foo://bla#> .

:uncleOf a owl:ObjectProperty.   ### required !!!
[ ] rdfs:subPropertyOf :uncleOf;
    owl:propertyChain (:brotherOf :hasChild).

:name a owl:FunctionalProperty.
:john a :Person; :brotherOf :sue.
:sue a :Person; :hasChild :anne, :barbara.
:anne :name "Anne".   :barbara :name "Barbara".

:UncleOfMore a owl:Class; owl:equivalentClass
 [a owl:Restriction; owl:onProperty :uncleOf; owl:minCardinality 2].
```

[Filename: RDF/uncleOfMore.n3]

```
prefix : <foo://bla#>
select ?U ?X
from <file:uncleOfMore.n3>
where {{?U :uncleOf ?X} UNION
       {?U a :uncleOfMore}}
```

[Filename: RDF/uncleOfMore.sparql]

- pellet: Definition of uncle is ignored; result empty.
  WARNING - Unsupported axiom: Ignoring transitivity and/or complex subproperty axioms for uncleOf

SELF RESTRICIONS: $\{x \mid x\ r\ x\}$

```
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.
@prefix : <foo://bla/>.
:Cyclic a owl:Class;
  owl:equivalentClass [ owl:intersectionOf
    (:Node [a owl:Restriction; owl:onProperty :to;
            owl:hasSelf "true"^^xsd:boolean ])].
:b a :Cyclic.
:a a :Node; :to :a, :b.
# :a a [ owl:complementOf :Cyclic ].
```

[Filename: RDF/cyclic.n3]

```
prefix : <foo://bla/>
select ?N ?N2
from <file:cyclic.n3>
where {{?N a :Cyclic} UNION
       {:a a :Cyclic} UNION
       {?N :to ?N2}}
```

[Filename: RDF/cyclic.sparql]

Self restrictions (Cont'd)

... just another example:

```
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.
@prefix : <foo://bla/>.
:NeuroticAnimal a owl:Class;
  owl:equivalentClass [ owl:intersectionOf
   ( :Animal
     [a owl:Restriction; owl:onProperty :bites; owl:hasSelf "true"^^xsd:boolean])].
:pluto a :Animal; :bites :pluto, :garfield.
:garfield a :NeuroticAnimal.
```

[Filename: RDF/neurotic.n3]

```
prefix : <foo://bla/>
select ?N ?N2
from <file:neurotic.n3>
where {{?N a :NeuroticAnimal} UNION
       {?N :bites ?N2}}
```

[Filename: RDF/neurotic.sparql]

Self restrictions (Cont'd)

... check for existence of cycles in a graph: Transitivity + SelfRestriction is not allowed:

WARNING: Unsupported axiom: Ignoring transitivity axiom due to an existing self restriction for property path

```
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.
@prefix : <foo://bla#>.
:edge rdfs:subPropertyOf :path.    ### use a win-move game as input
:path a owl:TransitiveProperty.
:Cyclic a owl:Class;
  owl:equivalentClass [ owl:intersectionOf  ( :Node
     [a owl:Restriction; owl:onProperty :path; owl:hasSelf "true"^^xsd:boolean])].
```

[Filename: RDF/cyclic-transitive.n3]

```
prefix : <foo://bla#>
select ?X ?Y ?N
from <file:cyclic-transitive.n3>
from <file:winmove-graph.n3>
where {{?X :path ?Y} UNION {?N a :Cyclic}}
```

[Filename: RDF/cyclic-transitive.sparql]

9.8 DL and OWL Proving and Query Answering

- Tableau provers use refutation techniques:
  Given an ontology formalization Φ,
  prove $\Phi \models \varphi$ by starting a tableau over $\Phi \wedge \neg\varphi$ and trying to close it.

For that, it is well-suited for *testing* if something holds:

- consistency of a concept definition:
  $KB \models C \equiv \bot \;\Leftrightarrow\; KB \cup \{C(a)\}$ for a new constant a is unsatisfiable.
- concept containment:
  $KB \models C \sqsubseteq D \;\Leftrightarrow\; KB \models (C \sqcap \neg D) \equiv \bot$.
- concept equivalence:
  $KB \models C \equiv D \;\Leftrightarrow\; KB \models C \sqsubseteq D$ and $KB \models D \sqsubseteq C$.
- concept membership (for a given individual a):
  $KB \models C(a) \;\Leftrightarrow\; KB \cup \{\neg C(a)\}$ is unsatisfiable.

Tableau Expansion Rules for DL

- DL: use tableau without free variables. Expansion of universally quantified formulas takes only place for constants that are actually introduced.
- makes it more similar to Model Checking
- actually, not the tableau is generated completely, but branches are investigated by backtracking.

$(C \sqcap D)(s)$	Add $C(s)$ and $D(s)$ to the branch.
$(C \sqcup D)(s)$	Add two branches, one with $C(s)$, the other with $D(s)$.
$\exists R.C(s)$	Add $R(s,x)$ and $C(x)$ where x is new.
$\forall R.C(s)$	Add $C(t)$ whenever $R(s,t)$ is on the tableau (requires bookkeeping).
$\geq nR.C(s)$	Add $R(s,x_1),\ldots,R(s,x_n),C(x_1),\ldots,C(x_n)$ and $x_i \neq x_j$ where x_i are new.
$\leq nR.C(s)$	Bookkeeping about $\{x \mid R(s,x)\}$. Whenever more than n, then add branches with all combinations $x_i = x_j$. Continue bookkeeping.
$C \sqsubseteq D$	For each s recursively add two branches with $\neg C(s)$ and $D(s)$.
Closure	Close a branch whenever $A(s)$ and $\neg A(s)$ occur.

Query Answering in DL and OWL

Query answering requires to find all answer bindings to variables.

- find all X such that $KB \models C(X)$.
- find all D such that $KB \models D \sqsubseteq C$.

Start a tableau and collect substitutions that close branches:

- start with $KB \cup \{\neg C(X)\}$.
- collect substitutions for X for which the tableau closes.
- without free variables: generate a new $\neg C(s)$ whenever any rule introduces a constant s. (= check if that s is an answer)
- harder to implement.
  Not always all answers are found by the current implementations.
- help the system by not only asking “{?X :age ?Y}”, but pruning the search space by “{?X a :Person; :age ?Y}”.

DL Tableaux: Examples

Who are John’s children?

hasChild(kate,john)
name(john,“John”)
hasChild(john,alice)
name(alice,“Alice”)
hasChild(john,bob)
name(bob,“Bob”)

Query: ?- hasChild(john,X).

¬ hasChild(john,X)
$\square\{X_1 \leftarrow$ alice}
$\square\{X_2 \leftarrow$ bob}

What are the names of John’s children?

hasChild(john,alice)
hasChild(john,bob)
name(john,“John”)
name(alice,“Alice”)
name(bob,“Bob”)

Query: ?- hasChild(john,_X), name(_X,N).

¬(hasChild(john,X) ∧ name(X,N))

¬(hasChild(john,X)) | ¬name(X,N)
Try $\square\{X_1 \leftarrow$ alice} | for X_1 and X_2:
Try $\square\{X_2 \leftarrow$ bob} | X_1 | X_2

¬ name(alice,N) | ¬ name(bob,N)
$N_1 \leftarrow$“Alice” | $N_2 \leftarrow$“Bob”

- Note: one could try close the right branch with $X_0 \leftarrow$john and $N_0 \leftarrow$“John”, but for that, the left branch will not close.
- Internal Strategy: don’t explicitly close with X_i.
  Instead prepare complete tableau and compute closing *relational algebra expression*:
  (π[$1=john](hasChild($1,X))) × name(X,N)

DL TABLEAUX: EXAMPLES

Consider the "Only female children" example from Slide 443.

TwoChildrenParent(sue)
hasChild(sue,ann)
Female(ann)
hasChild(sue,barbara)
Female(barbara)
ann $\neq$ barbara
TwoChildrenParent $\equiv \exists 2$ child.$\top$
OnlyFemaleChildrenParent $\equiv$ Person $\sqcap$ $\forall$child.Female

Query: ?- OnlyFemaleChildrenParent(X).

[$\neg$ OnlyFemaleChildrenParent(X)]

$\neg$(Person $\sqcap$ $\forall$child.Female(X))

$\neg$ Person(X) | $\neg$ ($\forall$child.Female)(sue)

try $\Box\{X \leftarrow$ sue$\}$ | ($\exists$ child).($\neg$ Female)(sue)

hasChild(sue,y)

$\neg$ Female(y)

count Sue's children=3: ann,barbara,y

ann=barbara | ann=y | barbara=y

$\Box$ | $\Box$ $\neg$ Female(ann) | $\Box$ $\neg$ Female(barbara)

- the negated query can be used for leading the expansion, but not for closing the tableau.
- Instead of X, all other persons are also tried to derive answers:
  John: tableau does not close
  (Alice)
  Kate: tableau does not close
  (Sue)

DL TABLEAUX: A MORE INVOLVED EXAMPLE

Consider again the Escher Stairs example (Slide 485).

(1) Corner = AllDifferent(a,b,c)
(2) cardinality: Corner $\sqsubseteq \exists 1$ higher.$\top$
(3) domain: Corner $\sqsupseteq \exists$ higher.$\top$
(4) range: $\top \sqsubseteq \forall$ higher.Corner
(5) AntiSymmetric(higher)
(6) Irreflexive(higher)
(7) higher(a,b)

Query: ?- higher(X,Y).

[$\neg$ higher(X,Y)]

First Answer Candidate:
with (7) X$\leftarrow$a, Y$\leftarrow$ b
Try further answers ...

- The negated query can be used for leading the expansion, but not for closing the tableau. The first answer candidate is higher(a,b) – which was given in the input.
- Show that the developing model is consistent,
- and try to find additional answer candidates.

(2) can be applied for any constant occurring in the branch.

- Choose "b" since it is already used in another fact and search for further answers in this model.

Escher stairs tableau: continue with (2) for b

(1) Corner = AllDifferent(a,b,c)
(2) cardinality: Corner $\sqsubseteq \exists 1$ higher.$\top$
(4) range: $\top \sqsubseteq \forall$ higher.Corner
(5) AntiSymmetric(higher)
(6) Irreflexive(higher)
(7) higher(a,b)

[$\neg$ higher(X,Y)]

apply (2) for b:

- $\neg$ Corner(b) — $\square$
- (8) $\exists 1$ higher.$\top$(b)
  - from (8): (9) higher(b,e)
  - next: range of "higher" derives that e is a corner:
  - (4) $\top \sqsubseteq \forall$ higher.Corner
    - $\neg\top$(b) — $\square$
    - (10) $\forall$ higher.Corner(b)
      - (11) Corner(e)
      - e must be one of a,b,c
        - e=a — higher(b,a) — $\square$ (Antisymm.)
        - e=b — higher(b,b) — $\square$ (Irrefl.)
        - e=c — (12) higher(b,c) — Answer candidate: (b,c)

- Expand the branch (=model) by investigating b.
- This yields another answer candidate.
- Still, it must be checked that the branch is not inconsistent.
- Further answers will be found.

Escher stairs tableau: continue with (2) for c

(1) Corner = AllDifferent(a,b,c)
(2) cardinality: Corner $\sqsubseteq \exists 1$ higher.$\top$
(4) range: $\top \sqsubseteq \forall$ higher.Corner
(5) AntiSymmetric(higher)
(6) Irreflexive(higher)
(7) higher(a,b)
(12) higher(b,c)

[$\neg$ higher(X,Y)]

continue with (2) for c

- $\neg$ Corner(c) — $\square$
- (13) $\exists 1$ higher.$\top$(c)
  - from (13): (14) higher(c,f)
  - next: range of "higher" derives that f is a corner:
  - (4) $\top \sqsubseteq \forall$ higher.Corner
    - $\neg\top$(c) — $\square$
    - (15) $\forall$ higher.Corner(c)
      - (16) Corner(f)
      - f must be one of a,b,c
        - f=a — higher(c,a) — Answer: (c,a) — [$\star$]
        - f=b — higher(c,b) — $\square$ (Antisymm.)
        - f=c — higher(c,c) — $\square$ (Irrefl.)

- The branch [$\star$] cannot be closed.
- The set of formulas on this branch is consistent and describes a model.
- The answers to ?- higher(X,Y) in this model are (a,b), (b,c), and (c,a).

Requirements on (not only DL) Tableau Strategies

- select most promising formula to be expanded next
  - based on coincident constants,
  - "selectivity" of conditions,
  - α-rules non-branching before β-rules (branching).
- non-closing branches: know when to stop and return answer matches
  - "saturated" branches: expansion does not add new formulas,
  - do not expand irrelevant formulas at all.

DL Tableaux: So far, so good ...

Consider the axiom

$$\text{Person} \sqsubseteq \exists \text{hasParent.Person}$$

The tableau generation does not terminate.

Blocking

- a constant s_2 is introduced as an existential filler from expanding a fact about constant s_1,
- the knowledge about s_1 and s_2 is *saturated* (i.e., nothing new about them can be derived),
- and the same facts are known about s_1 and s_2 except the above existential chain,
- then *block* s_2 from application of the existential formula (which would just create another same thing).
- Such blocking can be done for every existentially introduced thing, and it has only to be dropped if differences between it and its "predecessor" are derived.
- Such ontologies can be used. Queries only return instances in the "relevant" finite portion.

Blocking

```
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix : <foo://bla#>.
:kate a :Person; :name "Kate"; :hasChild :john.
:john a :Person; :name "John"; :hasChild :alice.
:alice a :Person; :name "Alice".
:hasChild rdfs:domain :Parent;
       owl:inverseOf :hasParent.
:Person rdfs:subClassOf
 [a owl:Restriction;  owl:onProperty :hasParent;  owl:cardinality 2].
:Parent owl:equivalentClass
 [a owl:Restriction; owl:onProperty :hasChild; owl:minCardinality 1].
:Grandparent owl:equivalentClass
 [a owl:Restriction; owl:onProperty :hasChild; owl:someValuesFrom :Parent].
:HasParent owl:equivalentClass
 [a owl:Restriction; owl:onProperty :hasParent; owl:someValuesFrom owl:Thing].
```

[Filename: RDF/infinite-parents.n3]

Blocking (cont'd)

```
prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>
prefix owl: <http://www.w3.org/2002/07/owl#>
prefix : <foo://bla#>
select ?A ?B ?C ?R ?X
from <file:infinite-parents.n3>
where {{?A a :Parent} UNION
       {?B a :Grandparent} UNION
       {?C a :HasParent} UNION  ## kate has a parent ...
       {:hasParent rdfs:range ?R} UNION
       {:kate :parent ?X}} # ... which is not output
```

[Filename: RDF/infinite-parents.sparql]

- The tableau strategy of pellet correctly blocks the generation of (useless) blank nodes.
- Note: when trying to count “how many persons must exist”, or “can we prove that at least 10 persons exist” would require to exclude that there is a cycle in the parents’ chain.

Exercise

Write RDF/OWL instances:

- John has two children in school, they are in the 3nd and 5th year. Children in the first year are 6 years old, those in the 2nd year are 7 years old, and so on. There are 12 years of school.
- Alice is a daugther of John. She is 10 years old.
- an "ideal family" consists of a father, a mother, and they have 2 children, a son and a daughter, and a dog.
- John's family is an "ideal family".
- Bob is John's son.

Feed them into the Jena tool, activate the reasoner.

- How old is Bob?
- which of the above information can be omitted without losing information how old Bob is?

9.9 Open World and Closed World: OWL/DL/Tableaux/Logic and SPARQL

- OWL/DL reasoning: OWA.
  Everything that can neither be proven nor disproven is unknown
- SPARQL queries/algebraic evaluation: CWA
  BGPs that do not match (not proven to be true, i.e. false or unknown) are considered as "no answer"

SPARQL CWA and OWL OWA: Possible – if not impossible

⇒ Use SPARQL to check what cannot be *proven* by using FILTER NOT EXISTS { *query* }:
If the negation of some formula φ cannot be proven – then φ is at least possible, i.e, there exists a model that makes φ true.

- Limited expressiveness $\neg\varphi$ must be OWL-DL-expressible.
  (means: wrt. stable models [Deductive Databases Lecture], where any possible solution can be described)
- Consider again Slide 392 for an earlier example.

SPARQL NOT EXISTS { $\neg\varphi$ } for checking possibility of φ

```
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix : <foo://bla#>.
:Childless owl:intersectionOf (:Person
  [ a owl:Restriction; owl:onProperty :hasChild; owl:maxCardinality 0]).
:Parent owl:intersectionOf (:Person
  [ a owl:Restriction; owl:onProperty :hasChild; owl:minCardinality 1]).
:john a :Person; :hasChild :alice, :bob.
:alice a :Person.  :bob a :Person.
```
[Filename: RDF/childless-small.n3]

```
prefix owl: <http://www.w3.org/2002/07/owl#>
prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>
prefix : <foo://bla#>
select ?X ?C
from <file:childless-small.n3>
where { ?X a :Person . ?C a owl:Class; rdfs:subClassOf :Person
        FILTER NOT EXISTS {?X a ?C}}
```
[Filename: RDF/childless-small.sparql]

- John: only possible that he is a parent;
  for alice and bob, it is possible to be a parent or to be childless.

SPARQL CWA and OWL OWA: Possible – if not impossible: A Scenario

- three rooms: bedroom, livingroom, guestroom
- some furniture: beds, a wardrobe, tables, chairs
- specification how many of these furniture can be placed in the rooms
- task: find out what can be placed where

Scenario

```
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix : <foo://rooms#>.
:in a owl:ObjectProperty, owl:FunctionalProperty; owl:inverseOf :has;
  rdfs:domain :Furniture; rdfs:range :Room.
:Room owl:oneOf (:bedroom :livingroom :guestroom).
:bedroom a :Room,
  [a owl:Restriction; owl:onProperty :has; owl:onClass :Bed; owl:qualifiedCardinality 1],
  [a owl:Restriction; owl:onProperty :has; owl:onClass :Wardrobe; owl:qualifiedCardinality 1],
  [a owl:Restriction; owl:onProperty :has; owl:onClass :Chair; owl:qualifiedCardinality 1],
  [a owl:Restriction; owl:onProperty :has; owl:onClass :Table; owl:maxQualifiedCardinality 0].
# :bedroom  :has :bed1 . # comment in or out ...
:guestroom a :Room,
  [a owl:Restriction; owl:onProperty :has; owl:onClass :Table; owl:maxQualifiedCardinality 0].
:livingroom a :Room,
  [a owl:Restriction; owl:onProperty :has; owl:onClass :Bed; owl:maxQualifiedCardinality 0],
  [a owl:Restriction; owl:onProperty :has; owl:onClass :Chair; owl:qualifiedCardinality 4].

:Furniture a owl:Class;
  owl:disjointUnionOf (:Bed :Wardrobe :Table :Chair);
  owl:equivalentClass [a owl:Restriction; owl:onProperty :in; owl:cardinality 1].
```