INF3580 – Semantic Technologies – Spring 2010

Lecture 10: OWL: Loose Ends

Martin Giese

13th April 2010

Department of Informatics

University of Oslo

Today's Plan

1. Reminder: OWL
2. Cardinality restrictions
3. More about Datatypes
4. owl:sameAs and owl:differentFrom
5. Disjointness and Covering Axioms

Outline

$\mathcal{ALC}$ Semantics

Interpretation

An interpretation $\mathcal{I}$ fixes a set $\Delta^{\mathcal{I}}$, the *domain*, $A^{\mathcal{I}} \subseteq \Delta$ for each atomic concept A, and $R^{\mathcal{I}} \subseteq \Delta \times \Delta$ for each role R

Interpretation of concept descriptions

$$
\begin{aligned}
\top^{\mathcal{I}} &= \Delta^{\mathcal{I}} \\
\bot^{\mathcal{I}} &= \emptyset \\
(\neg C)^{\mathcal{I}} &= \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}} \\
(C \sqcap D)^{\mathcal{I}} &= C^{\mathcal{I}} \cap D^{\mathcal{I}} \\
(C \sqcup D)^{\mathcal{I}} &= C^{\mathcal{I}} \cup D^{\mathcal{I}} \\
(\forall R.C)^{\mathcal{I}} &= \{a \in \Delta^{\mathcal{I}} \mid \forall b.(a, b) \in R^{\mathcal{I}} \to b \in C^{\mathcal{I}}\} \\
(\exists R.C)^{\mathcal{I}} &= \{a \in \Delta^{\mathcal{I}} \mid \exists b.(a, b) \in R^{\mathcal{I}} \land b \in C^{\mathcal{I}}\}
\end{aligned}
$$

$\mathcal{ALC}$ TBox and ABox

- The TBox

$\mathcal{ALC}$ TBox and ABox

- The TBox
  - is for *terminological knowledge*

$\mathcal{ALC}$ TBox and ABox

- The TBox
  - is for *terminological knowledge*
  - is independent of any actual instance data

$\mathcal{ALC}$ TBox and ABox

- The TBox
  - is for *terminological knowledge*
  - is independent of any actual instance data
  - is a set of axioms:

$\mathcal{ALC}$ TBox and ABox

- The TBox
  - is for *terminological knowledge*
  - is independent of any actual instance data
  - is a set of axioms:
    - Class inclusion $\sqsubseteq$, equivalence $\equiv$

$\mathcal{ALC}$ TBox and ABox

- The TBox
  - is for *terminological knowledge*
  - is independent of any actual instance data
  - is a set of axioms:
    - Class inclusion $\sqsubseteq$, equivalence $\equiv$
    - Role inclusion, functionality, transitivity, inverses,. . .

$\mathcal{ALC}$ TBox and ABox

- The TBox
  - is for *terminological knowledge*
  - is independent of any actual instance data
  - is a set of axioms:
    - Class inclusion $\sqsubseteq$, equivalence $\equiv$
    - Role inclusion, functionality, transitivity, inverses,...
- The ABox

$\mathcal{ALC}$ TBox and ABox

- The TBox
  - is for *terminological knowledge*
  - is independent of any actual instance data
  - is a set of axioms:
    - Class inclusion $\sqsubseteq$, equivalence $\equiv$
    - Role inclusion, functionality, transitivity, inverses,. . .
- The ABox
  - is for *assertional knowledge*

$\mathcal{ALC}$ TBox and ABox

- The TBox
  - is for *terminological knowledge*
  - is independent of any actual instance data
  - is a set of axioms:
    - Class inclusion $\sqsubseteq$, equivalence $\equiv$
    - Role inclusion, functionality, transitivity, inverses,. . .
- The ABox
  - is for *assertional knowledge*
  - contains facts about concrete instances $a, b, c, \ldots$

$\mathcal{ALC}$ TBox and ABox

- The TBox
  - is for *terminological knowledge*
  - is independent of any actual instance data
  - is a set of axioms:
    - Class inclusion $\sqsubseteq$, equivalence $\equiv$
    - Role inclusion, functionality, transitivity, inverses,...
- The ABox
  - is for *assertional knowledge*
  - contains facts about concrete instances $a, b, c, \ldots$
  - A set of concept assertions $C(a)$ ...

$\mathcal{ALC}$ TBox and ABox

- The TBox
  - is for *terminological knowledge*
  - is independent of any actual instance data
  - is a set of axioms:
    - Class inclusion $\sqsubseteq$, equivalence $\equiv$
    - Role inclusion, functionality, transitivity, inverses,...
- The ABox
  - is for *assertional knowledge*
  - contains facts about concrete instances $a, b, c, \ldots$
  - A set of concept assertions $C(a)$ ...
  - and role assertions $R(b, c)$

Recap of restrictions

- Existential restrictions

Recap of restrictions

- Existential restrictions
  - have the form $\exists R.C$

Recap of restrictions

- Existential restrictions
  - have the form $\exists R.C$
  - typically used to connect classes

Recap of restrictions

- Existential restrictions
  - have the form $\exists R.C$
  - typically used to connect classes
  - $A \sqsubseteq \exists R.C$: Every A-object is R-related to *some* C-object

Recap of restrictions

- Existential restrictions
  - have the form $\exists R.C$
  - typically used to connect classes
  - $A \sqsubseteq \exists R.C$: Every A-object is R-related to *some* C-object
- Universal restrictions

Recap of restrictions

- Existential restrictions
  - have the form $\exists R.C$
  - typically used to connect classes
  - $A \sqsubseteq \exists R.C$: Every A-object is R-related to *some* C-object
- Universal restrictions
  - have the form $\forall R.C$

Recap of restrictions

- Existential restrictions
  - have the form $\exists R.C$
  - typically used to connect classes
  - $A \sqsubseteq \exists R.C$: Every A-object is R-related to *some* C-object
- Universal restrictions
  - have the form $\forall R.C$
  - restrict the things a type of object can be connected to

Recap of restrictions

- Existential restrictions
  - have the form $\exists R.C$
  - typically used to connect classes
  - $A \sqsubseteq \exists R.C$: Every A-object is R-related to *some* C-object
- Universal restrictions
  - have the form $\forall R.C$
  - restrict the things a type of object can be connected to
  - $A \sqsubseteq \forall R.C$: Every A-object is R-related to C-objects *only*

Recap of restrictions

- Existential restrictions
  - have the form $\exists R.C$
  - typically used to connect classes
  - $A \sqsubseteq \exists R.C$: Every A-object is R-related to *some* C-object
- Universal restrictions
  - have the form $\forall R.C$
  - restrict the things a type of object can be connected to
  - $A \sqsubseteq \forall R.C$: Every A-object is R-related to C-objects *only*
  - A-objects may not be R-related to anything at all

Recap of restrictions

- Existential restrictions
  - have the form $\exists R.C$
  - typically used to connect classes
  - $A \sqsubseteq \exists R.C$: Every A-object is R-related to *some* C-object
- Universal restrictions
  - have the form $\forall R.C$
  - restrict the things a type of object can be connected to
  - $A \sqsubseteq \forall R.C$: Every A-object is R-related to C-objects *only*
  - A-objects may not be R-related to anything at all
- Example:

Recap of restrictions

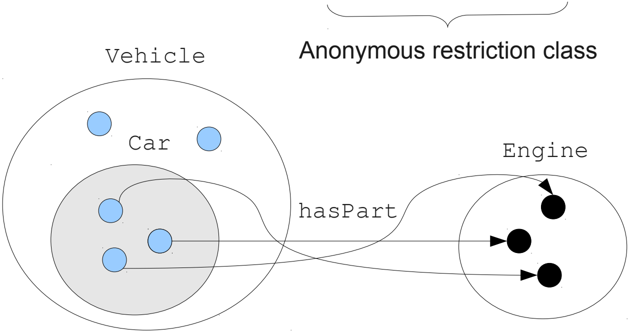
- Existential restrictions
  - have the form $\exists R.C$
  - typically used to connect classes
  - $A \sqsubseteq \exists R.C$: Every A-object is R-related to *some* C-object
- Universal restrictions
  - have the form $\forall R.C$
  - restrict the things a type of object can be connected to
  - $A \sqsubseteq \forall R.C$: Every A-object is R-related to C-objects *only*
  - A-objects may not be R-related to anything at all
- Example:
  - A car is a motorised vehicle

Recap of restrictions

- Existential restrictions
  - have the form $\exists R.C$
  - typically used to connect classes
  - $A \sqsubseteq \exists R.C$: Every A-object is R-related to *some* C-object
- Universal restrictions
  - have the form $\forall R.C$
  - restrict the things a type of object can be connected to
  - $A \sqsubseteq \forall R.C$: Every A-object is R-related to C-objects *only*
  - A-objects may not be R-related to anything at all
- Example:
  - A car is a motorised vehicle
  - $Car \sqsubseteq Vehicle \sqcap \exists hasPart.Engine$

Existential restrictions illustrated

$$Car \sqsubseteq Vehicle \sqcap \exists hasPart.Engine$$



A different perspective

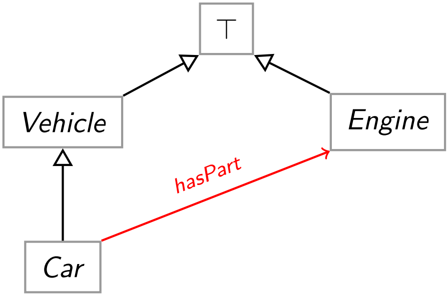
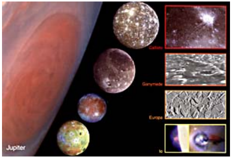


Figure: Connecting classes

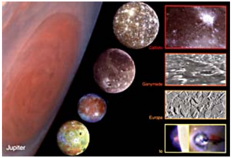
Outline

Cardinality restrictions

- Cardinality restrictions,

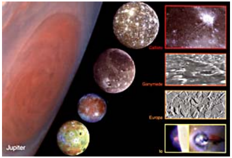


Cardinality restrictions



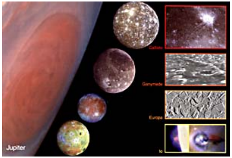
- Cardinality restrictions,
  - have the form $\geq_n R.C$ or $\leq_n R.C$

Cardinality restrictions



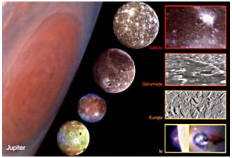
- Cardinality restrictions,
  - have the form $\geq_n R.C$ or $\leq_n R.C$
  - where n is a natural number

Cardinality restrictions



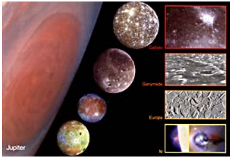
- Cardinality restrictions,
  - have the form $\geq_n R.C$ or $\leq_n R.C$
  - where n is a natural number
  - used to restrict *the number of connections*

Cardinality restrictions



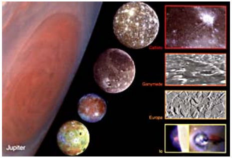
- Cardinality restrictions,
  - have the form $\geq_n R.C$ or $\leq_n R.C$
  - where n is a natural number
  - used to restrict *the number of connections*
  - $A \sqsubseteq \geq_3 R.C$: Every A-object is R-related to *at least* three C-objects.

Cardinality restrictions



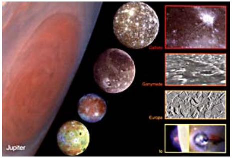
- Cardinality restrictions,
  - have the form $\geq_n R.C$ or $\leq_n R.C$
  - where n is a natural number
  - used to restrict *the number of connections*
  - $A \sqsubseteq \geq_3 R.C$: Every A-object is R-related to *at least* three C-objects.
  - $A \sqsubseteq \leq_3 R.C$: Every A-object is R-related to *at most* three C-objects.

Cardinality restrictions



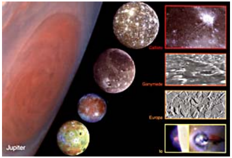
- Cardinality restrictions,
  - have the form $\geq_n R.C$ or $\leq_n R.C$
  - where n is a natural number
  - used to restrict *the number of connections*
  - $A \sqsubseteq \geq_3 R.C$: Every A-object is R-related to *at least* three C-objects.
  - $A \sqsubseteq \leq_3 R.C$: Every A-object is R-related to *at most* three C-objects.
- Example, combining restrictions:

Cardinality restrictions



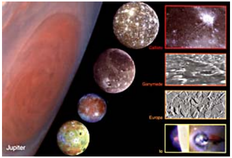
- Cardinality restrictions,
  - have the form $\geq_n R.C$ or $\leq_n R.C$
  - where n is a natural number
  - used to restrict *the number of connections*
  - $A \sqsubseteq \geq_3 R.C$: Every A-object is R-related to *at least* three C-objects.
  - $A \sqsubseteq \leq_3 R.C$: Every A-object is R-related to *at most* three C-objects.
- Example, combining restrictions:
  - Every planet orbits something: $Planet \sqsubseteq \exists orbits.\top$

Cardinality restrictions



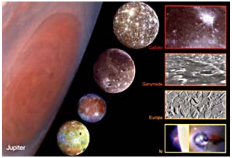
- Cardinality restrictions,
  - have the form $\geq_n R.C$ or $\leq_n R.C$
  - where n is a natural number
  - used to restrict *the number of connections*
  - $A \sqsubseteq \geq_3 R.C$: Every A-object is R-related to *at least* three C-objects.
  - $A \sqsubseteq \leq_3 R.C$: Every A-object is R-related to *at most* three C-objects.
- Example, combining restrictions:
  - Every planet orbits something: $Planet \sqsubseteq \exists orbits.\top$
  - Anything a planet orbits is a star: $Planet \sqsubseteq \forall orbits.Star$

Cardinality restrictions



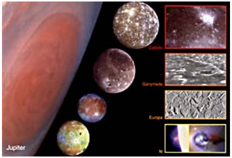
- Cardinality restrictions,
  - have the form $\geq_n R.C$ or $\leq_n R.C$
  - where n is a natural number
  - used to restrict *the number of connections*
  - $A \sqsubseteq \geq_3 R.C$: Every A-object is R-related to *at least* three C-objects.
  - $A \sqsubseteq \leq_3 R.C$: Every A-object is R-related to *at most* three C-objects.
- Example, combining restrictions:
  - Every planet orbits something: $Planet \sqsubseteq \exists orbits.\top$
  - Anything a planet orbits is a star: $Planet \sqsubseteq \forall orbits.Star$
  - Planets cannot orbit more than one star: $Planet \sqsubseteq \leq_1 orbits.Star$

Cardinality restrictions



- Cardinality restrictions,
  - have the form $\geq_n R.C$ or $\leq_n R.C$
  - where n is a natural number
  - used to restrict *the number of connections*
  - $A \sqsubseteq \geq_3 R.C$: Every A-object is R-related to *at least* three C-objects.
  - $A \sqsubseteq \leq_3 R.C$: Every A-object is R-related to *at most* three C-objects.
- Example, combining restrictions:
  - Every planet orbits something: $Planet \sqsubseteq \exists orbits.\top$
  - Anything a planet orbits is a star: $Planet \sqsubseteq \forall orbits.Star$
  - Planets cannot orbit more than one star: $Planet \sqsubseteq \leq_1 orbits.Star$
  - A solar system has at least one star and one planet:

Cardinality restrictions



- Cardinality restrictions,
  - have the form $\geq_n R.C$ or $\leq_n R.C$
  - where n is a natural number
  - used to restrict *the number of connections*
  - $A \sqsubseteq \geq_3 R.C$: Every A-object is R-related to *at least* three C-objects.
  - $A \sqsubseteq \leq_3 R.C$: Every A-object is R-related to *at most* three C-objects.
- Example, combining restrictions:
  - Every planet orbits something: $Planet \sqsubseteq \exists orbits.\top$
  - Anything a planet orbits is a star: $Planet \sqsubseteq \forall orbits.Star$
  - Planets cannot orbit more than one star: $Planet \sqsubseteq \leq_1 orbits.Star$
  - A solar system has at least one star and one planet:

$$SolarSystem \sqsubseteq \geq_1 hasPart.Star \sqcap \geq_1 hasPart.Planet$$

Some equivalences

- Existential restrictions vs. Cardinality restrictions:

$$\exists R.C \equiv \geq_1 R.C$$

Some equivalences

- Existential restrictions vs. Cardinality restrictions:

$$\exists R.C \equiv \geq_1 R.C$$

- Universal restrictions vs. Cardinality restrictions:

$$\forall R.C \equiv \leq_0 R.\neg C$$

Some equivalences

- Existential restrictions vs. Cardinality restrictions:

$$\exists R.C \equiv \geq_1 R.C$$

- Universal restrictions vs. Cardinality restrictions:

$$\forall R.C \equiv \leq_0 R.\neg C$$

- Minimum cardinality versus maximum cardinality

$$\leq_3 R.C \equiv \neg \geq_4 R.C$$

Some equivalences

- Existential restrictions vs. Cardinality restrictions:

$$\exists R.C \equiv \geq_1 R.C$$

- Universal restrictions vs. Cardinality restrictions:

$$\forall R.C \equiv \leq_0 R.\neg C$$

- Minimum cardinality versus maximum cardinality

$$\leq_3 R.C \equiv \neg \geq_4 R.C$$

$$\leq_n R.C \equiv \neg \geq_{n+1} R.C$$

Some equivalences

- Existential restrictions vs. Cardinality restrictions:

$$\exists R.C \equiv \geq_1 R.C$$

- Universal restrictions vs. Cardinality restrictions:

$$\forall R.C \equiv \leq_0 R.\neg C$$

- Minimum cardinality versus maximum cardinality

$$\leq_3 R.C \equiv \neg \geq_4 R.C$$

$$\leq_n R.C \equiv \neg \geq_{n+1} R.C$$

- The 0 case

$$\leq_0 R.C \equiv \neg\exists R.C$$

$$\geq_0 R.C \equiv \top$$

Some equivalences

- Existential restrictions vs. Cardinality restrictions:

$$\exists R.C \equiv \geq_1 R.C$$

- Universal restrictions vs. Cardinality restrictions:

$$\forall R.C \equiv \leq_0 R.\neg C$$

- Minimum cardinality versus maximum cardinality

$$\leq_3 R.C \equiv \neg \geq_4 R.C$$

$$\leq_n R.C \equiv \neg \geq_{n+1} R.C$$

- The 0 case

$$\leq_0 R.C \equiv \neg\exists R.C$$

$$\geq_0 R.C \equiv \top$$

- R is functional $\iff \leq_1 R.\top$

Manchester Syntax

- $\leq_1$ *orbits.Star*
  `orbits max 1 Star`
- $\geq_8$ *hasPart.Planet*
  `hasPart min 8 Planet`

The $\mathcal{ALCQ}$ Description Logic

$\mathcal{ALCQ}$ concept descriptions

$C, D \rightarrow$	A	$\mid$	(atomic concept)
	$\top$	$\mid$	(universal concept)
	$\bot$	$\mid$	(bottom concept)
	$\neg C$	$\mid$	(atomic negation)
	$C \sqcap D$	$\mid$	(intersection)
	$C \sqcup D$	$\mid$	(union)
	$\forall R.C$	$\mid$	(value restriction)
	$\exists R.C$	$\mid$	(existential restriction)
	$\leq_n R.C$	$\mid$	(max. cardinality restriction)
	$\geq_n R.C$	$\mid$	(min. cardinality restriction)

$\mathcal{ALCQ}$ Semantics

Interpretation of concept descriptions

$$
\begin{aligned}
\top^{\mathcal{I}} &= \Delta^{\mathcal{I}} \\
\bot^{\mathcal{I}} &= \emptyset \\
(\neg C)^{\mathcal{I}} &= \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}} \\
(C \sqcap D)^{\mathcal{I}} &= C^{\mathcal{I}} \cap D^{\mathcal{I}} \\
(C \sqcup D)^{\mathcal{I}} &= C^{\mathcal{I}} \cup D^{\mathcal{I}} \\
(\forall R.C)^{\mathcal{I}} &= \{a \in \Delta^{\mathcal{I}} \mid \forall b.(a, b) \in R^{\mathcal{I}} \rightarrow b \in C^{\mathcal{I}}\} \\
(\exists R.C)^{\mathcal{I}} &= \{a \in \Delta^{\mathcal{I}} \mid \exists b.(a, b) \in R^{\mathcal{I}} \wedge b \in C^{\mathcal{I}}\} \\
(\leq_n R.C)^{\mathcal{I}} &= \{a \in \Delta^{\mathcal{I}} \mid \#\{b \mid (a, b) \in R^{\mathcal{I}} \wedge b \in C^{\mathcal{I}}\} \leq n\} \\
(\geq_n R.C)^{\mathcal{I}} &= \{a \in \Delta^{\mathcal{I}} \mid \#\{b \mid (a, b) \in R^{\mathcal{I}} \wedge b \in C^{\mathcal{I}}\} \geq n\}
\end{aligned}
$$

Cardinalities, non-unique names and open worlds

Cardinalities + the OWA and the NUNA is tricky, consider:

TBox:

$$Ensemble \sqsubseteq ChamberEnsemble \sqcup Orchestra$$

$$ChamberEnsemble \sqsubseteq \leq_1 firstViolin.\top$$

Cardinalities, non-unique names and open worlds

Cardinalities + the OWA and the NUNA is tricky, consider:

TBox:

$$\mathit{Ensemble} \sqsubseteq \mathit{ChamberEnsemble} \sqcup \mathit{Orchestra}$$

$$\mathit{ChamberEnsemble} \sqsubseteq \leq_1 \mathit{firstViolin}.\top$$

That is;

- Ensembles are either orchestras or chamber ensembles
- Chamber ensembles have only one instrument on each voice...
- in particular, only one first violin.

Cardinalities, non-unique names and open worlds

Cardinalities + the OWA and the NUNA is tricky, consider:

TBox:

$$\mathit{Ensemble} \sqsubseteq \mathit{ChamberEnsemble} \sqcup \mathit{Orchestra}$$

$$\mathit{ChamberEnsemble} \sqsubseteq \leq_1 \mathit{firstViolin}.\top$$

That is;

- Ensembles are either orchestras or chamber ensembles
- Chamber ensembles have only one instrument on each voice...
- in particular, only one first violin.

ABox:

```
Ensemble(oslo)
firstViolin(oslo, båtnes)
firstViolin(oslo, tønnesen)
```

Musical taxons

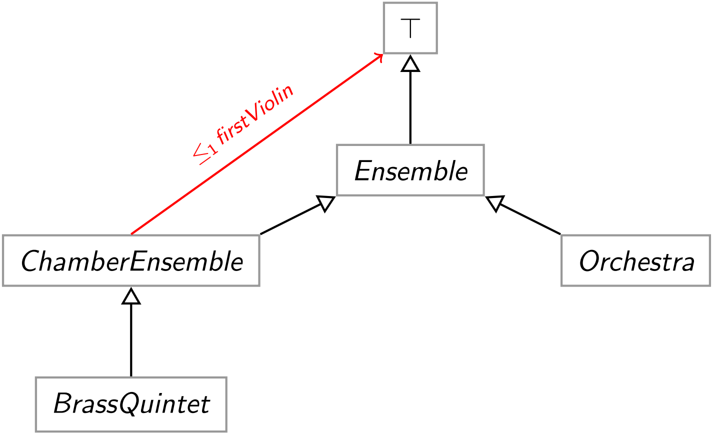


Figure: An ontology of ensembles

Unexpected (non-)results

- It does not follow from TBox + ABox that *oslo* is an *Orchestra*

Unexpected (non-)results

- It does not follow from TBox + ABox that *oslo* is an *Orchestra*
  - This is due to the NUNA

Unexpected (non-)results

- It does not follow from TBox + ABox that *oslo* is an *Orchestra*
  - This is due to the NUNA
  - We cannot assume that `båtnes` and `tønnesen` are distinct

Unexpected (non-)results

- It does not follow from TBox + ABox that *oslo* is an *Orchestra*
  - This is due to the NUNA
  - We cannot assume that `båtnes` and `tønnesen` are distinct
  - Hence, we must add statements to this effect to the ABox:

Unexpected (non-)results

- It does not follow from TBox + ABox that *oslo* is an *Orchestra*
    - This is due to the NUNA
    - We cannot assume that `båtnes` and `tønnesen` are distinct
    - Hence, we must add statements to this effect to the ABox:
        - `båtnes owl:differentFrom tønnesen`,

Unexpected (non-)results

- It does not follow from TBox + ABox that *oslo* is an *Orchestra*
  - This is due to the NUNA
  - We cannot assume that `båtnes` and `tønnesen` are distinct
  - Hence, we must add statements to this effect to the ABox:
    - `båtnes owl:differentFrom tønnesen`,
    - or in logic-notation: `båtnes`≠`tønnesen`,

Unexpected (non-)results

- It does not follow from TBox + ABox that *oslo* is an *Orchestra*
  - This is due to the NUNA
  - We cannot assume that `båtnes` and `tønnesen` are distinct
  - Hence, we must add statements to this effect to the ABox:
    - `båtnes owl:differentFrom tønnesen`,
    - or in logic-notation: `båtnes`≠`tønnesen`,
- Conversely, if we remove `firstViolin(oslo, tønnesen)`...

Unexpected (non-)results

- It does not follow from TBox + ABox that *oslo* is an *Orchestra*
  - This is due to the NUNA
  - We cannot assume that `båtnes` and `tønnesen` are distinct
  - Hence, we must add statements to this effect to the ABox:
    - `båtnes owl:differentFrom tønnesen`,
    - or in logic-notation: `båtnes`≠`tønnesen`,
- Conversely, if we remove `firstViolin(oslo, tønnesen)`...
  - it does not follow that `oslo` is a *ChamberEnsemble*

Unexpected (non-)results

- It does not follow from TBox + ABox that *oslo* is an *Orchestra*
  - This is due to the NUNA
  - We cannot assume that `båtnes` and `tønnesen` are distinct
  - Hence, we must add statements to this effect to the ABox:
    - `båtnes owl:differentFrom tønnesen`,
    - or in logic-notation: `båtnes`≠`tønnesen`,
- Conversely, if we remove `firstViolin(oslo, tønnesen)`...
  - it does not follow that `oslo` is a *ChamberEnsemble*
  - This is due to the OWA

Unexpected (non-)results

- It does not follow from TBox + ABox that *oslo* is an *Orchestra*
  - This is due to the NUNA
  - We cannot assume that `båtnes` and `tønnesen` are distinct
  - Hence, we must add statements to this effect to the ABox:
    - `båtnes owl:differentFrom tønnesen`,
    - or in logic-notation: `båtnes`$\neq$`tønnesen`,
- Conversely, if we remove `firstViolin(oslo, tønnesen)`...
  - it does not follow that `oslo` is a *ChamberEnsemble*
  - This is due to the OWA
  - According to which we may not know everything about `oslo`

Unexpected (non-)results

- It does not follow from TBox + ABox that *oslo* is an *Orchestra*
  - This is due to the NUNA
  - We cannot assume that `båtnes` and `tønnesen` are distinct
  - Hence, we must add statements to this effect to the ABox:
    - `båtnes owl:differentFrom tønnesen`,
    - or in logic-notation: `båtnes`$\neq$`tønnesen`,
- Conversely, if we remove `firstViolin(oslo, tønnesen)`...
  - it does not follow that `oslo` is a *ChamberEnsemble*
  - This is due to the OWA
  - According to which we may not know everything about `oslo`
  - in particular there may be other first violinists

A tempting mistake

- Cardinality restrictions are not suitable to express

A tempting mistake

- Cardinality restrictions are not suitable to express
  - durations

A tempting mistake

- Cardinality restrictions are not suitable to express
  - durations
  - intervals

A tempting mistake

- Cardinality restrictions are not suitable to express
  - durations
  - intervals
  - or any kind of sequence

A tempting mistake

- Cardinality restrictions are not suitable to express
  - durations
  - intervals
  - or any kind of sequence
  - and they cannot be used for arithmetic

A tempting mistake

- Cardinality restrictions are not suitable to express
  - durations
  - intervals
  - or any kind of sequence
  - and they cannot be used for arithmetic
- Anti-pattern:

A tempting mistake

- Cardinality restrictions are not suitable to express
  - durations
  - intervals
  - or any kind of sequence
  - and they cannot be used for arithmetic
- Anti-pattern:
  - Scotch whisky is aged at least 3 years:

A tempting mistake

- Cardinality restrictions are not suitable to express
  - durations
  - intervals
  - or any kind of sequence
  - and they cannot be used for arithmetic
- Anti-pattern:
  - Scotch whisky is aged at least 3 years:
  - Use a datatype property *age* with range *int*.

A tempting mistake

- Cardinality restrictions are not suitable to express
  - durations
  - intervals
  - or any kind of sequence
  - and they cannot be used for arithmetic
- Anti-pattern:
  - Scotch whisky is aged at least 3 years:
  - Use a datatype property *age* with range *int*.
  - $Scotch \sqsubseteq Whisky \sqcap \geq_3 age.int$

A tempting mistake

- Cardinality restrictions are not suitable to express
  - durations
  - intervals
  - or any kind of sequence
  - and they cannot be used for arithmetic
- Anti-pattern:
  - Scotch whisky is aged at least 3 years:
  - Use a datatype property *age* with range *int*.
  - $Scotch \sqsubseteq Whisky \sqcap \geq_3 age.int$
- Why?

A tempting mistake

- Cardinality restrictions are not suitable to express
  - durations
  - intervals
  - or any kind of sequence
  - and they cannot be used for arithmetic
- Anti-pattern:
  - Scotch whisky is aged at least 3 years:
  - Use a datatype property *age* with range *int*.
  - $Scotch \sqsubseteq Whisky \sqcap \geq_3 age.int$

- Why?
  - This says that Scotch has at least 3 *different ages*

A tempting mistake

- Cardinality restrictions are not suitable to express
  - durations
  - intervals
  - or any kind of sequence
  - and they cannot be used for arithmetic
- Anti-pattern:
  - Scotch whisky is aged at least 3 years:
  - Use a datatype property *age* with range *int*.
  - $Scotch \sqsubseteq Whisky \sqcap \geq_3 age.int$

- Why?
  - This says that Scotch has at least 3 *different ages*
  - For instance -1, 0, 15

A possible solution

- Idea: don't use age.

A possible solution

- Idea: don't use age.
- Use a property *casked*

A possible solution

- Idea: don't use age.
- Use a property *casked*
    - domain *Whisky*

A possible solution

- Idea: don't use age.
- Use a property *casked*
  - domain *Whisky*
  - range *int*

A possible solution

- Idea: don't use age.
- Use a property *casked*
  - domain *Whisky*
  - range *int*
  - relates the whisky to each year it is in the cask.

A possible solution

- Idea: don't use age.
- Use a property *casked*
  - domain *Whisky*
  - range *int*
  - relates the whisky to each year it is in the cask.

e.g. `:young :casked "2000"^^int, "2001"^^int, "2002"^^int`

A possible solution

- Idea: don't use age.
- Use a property *casked*
  - domain *Whisky*
  - range *int*
  - relates the whisky to each year it is in the cask.

e.g. `:young :casked "2000"^^int, "2001"^^int, "2002"^^int`

- $\mathit{Scotch} \sqsubseteq \mathit{Whisky} \sqcap \geq_3 \mathit{casked}.\mathit{int}$

A possible solution

- Idea: don't use age.
- Use a property *casked*
  - domain *Whisky*
  - range *int*
  - relates the whisky to each year it is in the cask.

e.g. `:young :casked "2000"^^int, "2001"^^int, "2002"^^int`

- $\mathit{Scotch} \sqsubseteq \mathit{Whisky} \sqcap \geq_3 \mathit{casked}.\mathit{int}$
- Works, but...

A possible solution

- Idea: don't use age.
- Use a property *casked*
  - domain *Whisky*
  - range *int*
  - relates the whisky to each year it is in the cask.

e.g. `:young :casked "2000"^^int, "2001"^^int, "2002"^^int`

- $\mathit{Scotch} \sqsubseteq \mathit{Whisky} \sqcap \geq_3 \mathit{casked}.\mathit{int}$
- Works, but. . .
- Can't express e.g. that the years are consecutive

A possible solution

- Idea: don't use age.
- Use a property *casked*
  - domain *Whisky*
  - range *int*
  - relates the whisky to each year it is in the cask.

e.g. `:young :casked "2000"^^int, "2001"^^int, "2002"^^int`

- $\mathit{Scotch} \sqsubseteq \mathit{Whisky} \sqcap \ \geq_3 \mathit{casked}.\mathit{int}$
- Works, but. . .
- Can't express e.g. that the years are consecutive
  - Knowing a whisky is casked in 2000 and 2009 doesn't imply it is casked for 10 years.

A possible solution

- Idea: don't use age.
- Use a property *casked*
  - domain *Whisky*
  - range *int*
  - relates the whisky to each year it is in the cask.

e.g. `:young :casked "2000"^^int, "2001"^^int, "2002"^^int`

- $\mathit{Scotch} \sqsubseteq \mathit{Whisky} \sqcap \geq_3 \mathit{casked}.\mathit{int}$
- Works, but...
- Can't express e.g. that the years are consecutive
  - Knowing a whisky is casked in 2000 and 2009 doesn't imply it is casked for 10 years.
- Reasoning about $\geq_n$ often works by generating n sample instances

A possible solution

- Idea: don't use age.
- Use a property *casked*
  - domain *Whisky*
  - range *int*
  - relates the whisky to each year it is in the cask.

e.g. `:young :casked "2000"^^int, "2001"^^int, "2002"^^int`

- $Scotch \sqsubseteq Whisky \sqcap \geq_3 casked.int$
- Works, but...
- Can't express e.g. that the years are consecutive
  - Knowing a whisky is casked in 2000 and 2009 doesn't imply it is casked for 10 years.
- Reasoning about $\geq_n$ often works by generating n sample instances
  - $Town \equiv \geq_{10000} inhabitant.Person$

A possible solution

- Idea: don't use age.
- Use a property *casked*
    - domain *Whisky*
    - range *int*
    - relates the whisky to each year it is in the cask.

e.g. `:young :casked "2000"^^int, "2001"^^int, "2002"^^int`

- $\mathit{Scotch} \sqsubseteq \mathit{Whisky} \sqcap \geq_3 \mathit{casked}.\mathit{int}$
- Works, but...
- Can't express e.g. that the years are consecutive
    - Knowing a whisky is casked in 2000 and 2009 doesn't imply it is casked for 10 years.
- Reasoning about $\geq_n$ often works by generating n sample instances
    - $\mathit{Town} \equiv \geq_{10000} \mathit{inhabitant}.\mathit{Person}$
    - $\mathit{Metropolis} \equiv \geq_{1000000} \mathit{inhabitant}.\mathit{Person}$

A possible solution

- Idea: don't use age.
- Use a property *casked*
  - domain *Whisky*
  - range *int*
  - relates the whisky to each year it is in the cask.

e.g. `:young :casked "2000"^^int, "2001"^^int, "2002"^^int`

- $\mathit{Scotch} \sqsubseteq \mathit{Whisky} \sqcap \ \geq_3 \mathit{casked}.\mathit{int}$
- Works, but. . .
- Can't express e.g. that the years are consecutive
  - Knowing a whisky is casked in 2000 and 2009 doesn't imply it is casked for 10 years.
- Reasoning about $\geq_n$ often works by generating n sample instances
  - $\mathit{Town} \equiv \ \geq_{10000} \mathit{inhabitant}.\mathit{Person}$
  - $\mathit{Metropolis} \equiv \ \geq_{1000000} \mathit{inhabitant}.\mathit{Person}$
  - Will kill almost any reasoner

Outline

Reminder: Datatype properties

- OWL distinguishes between

Reminder: Datatype properties

- OWL distinguishes between
  - object properties: go from resources to resources

Reminder: Datatype properties

- OWL distinguishes between
  - object properties: go from resources to resources
  - datatype properties: go from resources to literals

Reminder: Datatype properties

- OWL distinguishes between
  - object properties: go from resources to resources
  - datatype properties: go from resources to literals
- OWL (2) prescribes a list of available datatypes for literals

Reminder: Datatype properties

- OWL distinguishes between
  - object properties: go from resources to resources
  - datatype properties: go from resources to literals
- OWL (2) prescribes a list of available datatypes for literals
  - Numbers: real, rational, integer, positive integer, double, long,...

Reminder: Datatype properties

- OWL distinguishes between
  - object properties: go from resources to resources
  - datatype properties: go from resources to literals
- OWL (2) prescribes a list of available datatypes for literals
  - Numbers: real, rational, integer, positive integer, double, long,...
  - Strings

Reminder: Datatype properties

- OWL distinguishes between
  - object properties: go from resources to resources
  - datatype properties: go from resources to literals
- OWL (2) prescribes a list of available datatypes for literals
  - Numbers: real, rational, integer, positive integer, double, long,. . .
  - Strings
  - Booleans

Reminder: Datatype properties

- OWL distinguishes between
  - object properties: go from resources to resources
  - datatype properties: go from resources to literals
- OWL (2) prescribes a list of available datatypes for literals
  - Numbers: real, rational, integer, positive integer, double, long,...
  - Strings
  - Booleans
  - Binary data

Reminder: Datatype properties

- OWL distinguishes between
  - object properties: go from resources to resources
  - datatype properties: go from resources to literals
- OWL (2) prescribes a list of available datatypes for literals
  - Numbers: real, rational, integer, positive integer, double, long,...
  - Strings
  - Booleans
  - Binary data
  - IRIs

Reminder: Datatype properties

- OWL distinguishes between
  - object properties: go from resources to resources
  - datatype properties: go from resources to literals
- OWL (2) prescribes a list of available datatypes for literals
  - Numbers: real, rational, integer, positive integer, double, long,...
  - Strings
  - Booleans
  - Binary data
  - IRIs
  - Time Instants

Reminder: Datatype properties

- OWL distinguishes between
  - object properties: go from resources to resources
  - datatype properties: go from resources to literals
- OWL (2) prescribes a list of available datatypes for literals
  - Numbers: real, rational, integer, positive integer, double, long,...
  - Strings
  - Booleans
  - Binary data
  - IRIs
  - Time Instants
  - XML Literals

Reminder: Datatype properties

- OWL distinguishes between
  - object properties: go from resources to resources
  - datatype properties: go from resources to literals
- OWL (2) prescribes a list of available datatypes for literals
  - Numbers: real, rational, integer, positive integer, double, long,. . .
  - Strings
  - Booleans
  - Binary data
  - IRIs
  - Time Instants
  - XML Literals
- Varying tool support (Protégé 4.1 alpha for some of this)

Reminder: Datatype properties

- OWL distinguishes between
  - object properties: go from resources to resources
  - datatype properties: go from resources to literals
- OWL (2) prescribes a list of available datatypes for literals
  - Numbers: real, rational, integer, positive integer, double, long,...
  - Strings
  - Booleans
  - Binary data
  - IRIs
  - Time Instants
  - XML Literals
- Varying tool support (Protégé 4.1 alpha for some of this)
- Possible to define more (dates, date ranges, etc.)

Data Ranges

- Like concept descriptions, only for data types

Data Ranges

- Like concept descriptions, only for data types
- Boolean combinations allowed (Manchester syntax)

Data Ranges

- Like concept descriptions, only for data types
- Boolean combinations allowed (Manchester syntax)
  - `xsd:integer or xsd:string`

Data Ranges

- Like concept descriptions, only for data types
- Boolean combinations allowed (Manchester syntax)
  - `xsd:integer or xsd:string`
  - `xsd:integer and not xsd:byte`

Data Ranges

- Like concept descriptions, only for data types
- Boolean combinations allowed (Manchester syntax)
  - `xsd:integer or xsd:string`
  - `xsd:integer and not xsd:byte`
- Each basic datatype can be restricted by a number of *facets*

Data Ranges

- Like concept descriptions, only for data types
- Boolean combinations allowed (Manchester syntax)
  - `xsd:integer or xsd:string`
  - `xsd:integer and not xsd:byte`
- Each basic datatype can be restricted by a number of *facets*
  - `xsd:integer[>= 9]` – integers $>=$ 9.

Data Ranges

- Like concept descriptions, only for data types
- Boolean combinations allowed (Manchester syntax)
  - `xsd:integer or xsd:string`
  - `xsd:integer and not xsd:byte`
- Each basic datatype can be restricted by a number of *facets*
  - `xsd:integer[>= 9]` – integers $>=$ 9.
  - `xsd:integer[>= 9, <= 11]` – integers between 9, 10, and 11.

Data Ranges

- Like concept descriptions, only for data types
- Boolean combinations allowed (Manchester syntax)
  - `xsd:integer or xsd:string`
  - `xsd:integer and not xsd:byte`
- Each basic datatype can be restricted by a number of *facets*
  - `xsd:integer[>= 9]` – integers $>=$ 9.
  - `xsd:integer[>= 9, <= 11]` – integers between 9, 10, and 11.
  - `xsd:string[length 5]` – strings of length 5.

Data Ranges

- Like concept descriptions, only for data types
- Boolean combinations allowed (Manchester syntax)
  - `xsd:integer or xsd:string`
  - `xsd:integer and not xsd:byte`
- Each basic datatype can be restricted by a number of *facets*
  - `xsd:integer[>= 9]` – integers $>=$ 9.
  - `xsd:integer[>= 9, <= 11]` – integers between 9, 10, and 11.
  - `xsd:string[length 5]` – strings of length 5.
  - `xsd:string[maxLength 5]` – strings of length $\leq$ 5.

Data Ranges

- Like concept descriptions, only for data types
- Boolean combinations allowed (Manchester syntax)
  - `xsd:integer or xsd:string`
  - `xsd:integer and not xsd:byte`
- Each basic datatype can be restricted by a number of *facets*
  - `xsd:integer[>= 9]` – integers $>=$ 9.
  - `xsd:integer[>= 9, <= 11]` – integers between 9, 10, and 11.
  - `xsd:string[length 5]` – strings of length 5.
  - `xsd:string[maxLength 5]` – strings of length $\leq$ 5.
  - `xsd:string[minLength 5]` – strings of length $\geq$ 5.

Data Ranges

- Like concept descriptions, only for data types
- Boolean combinations allowed (Manchester syntax)
  - `xsd:integer or xsd:string`
  - `xsd:integer and not xsd:byte`
- Each basic datatype can be restricted by a number of *facets*
  - `xsd:integer[>= 9]` – integers $>=$ 9.
  - `xsd:integer[>= 9, <= 11]` – integers between 9, 10, and 11.
  - `xsd:string[length 5]` – strings of length 5.
  - `xsd:string[maxLength 5]` – strings of length $\leq$ 5.
  - `xsd:string[minLength 5]` – strings of length $\geq$ 5.
  - `xsd:string[pattern "[01]*"]` – strings consisting of 0 and 1.

Range Examples

- A whisky that is at least 12 years old:
  `Whisky and age some integer[>= 12]`

Range Examples

- A whisky that is at least 12 years old:
  `Whisky and age some integer[>= 12]`
- A teenager:
  `Person and age some integer[>= 13, <= 19]`

Range Examples

- A whisky that is at least 12 years old:
  `Whisky and age some integer[>= 12]`
- A teenager:
  `Person and age some integer[>= 13, <= 19]`
- A metropolis:
  `Place and nrInhabitants some integer[>= 1000000]`

Range Examples

- A whisky that is at least 12 years old:
  `Whisky and age some integer[>= 12]`
- A teenager:
  `Person and age some integer[>= 13, <= 19]`
- A metropolis:
  `Place and nrInhabitants some integer[>= 1000000]`

- Note: often makes best sense with functional properties

Pattern Examples

- An integer or a string of digits

Pattern Examples

- An integer or a string of digits
    - `xsd:integer or xsd:string[pattern "[0-9]+"]`

Pattern Examples

- An integer or a string of digits
  - `xsd:integer or xsd:string[pattern "[0-9]+"]`
- ISBN numbers: 13 digits in 5 −-separted groups, first 978 or 979, last a single digit.

Pattern Examples

- An integer or a string of digits
  - `xsd:integer or xsd:string[pattern "[0-9]+"]`
- ISBN numbers: 13 digits in 5 –-separted groups, first 978 or 979, last a single digit.
  - Book $\sqsubseteq$ `ISBN some string[length 17 ,`
    `pattern "97[89]-[0-9]+-[0-9]+-[0-9]+-[0-9]"]`

Pattern Examples

- An integer or a string of digits
  - `xsd:integer or xsd:string[pattern "[0-9]+"]`
- ISBN numbers: 13 digits in 5 −-separted groups, first 978 or 979, last a single digit.
  - Book $\sqsubseteq$ `ISBN some string[length 17 ,`
    `pattern "97[89]-[0-9]+-[0-9]+-[0-9]+-[0-9]"]`
- Reasoning about patterns:

Pattern Examples

- An integer or a string of digits
  - `xsd:integer or xsd:string[pattern "[0-9]+"]`
- ISBN numbers: 13 digits in 5 –-separted groups, first 978 or 979, last a single digit.
  - `Book` $\sqsubseteq$ `ISBN some string[length 17 ,`
    `pattern "97[89]-[0-9]+-[0-9]+-[0-9]+-[0-9]"]`
- Reasoning about patterns:
  - `str` a functional datatype property

Pattern Examples

- An integer or a string of digits
  - `xsd:integer or xsd:string[pattern "[0-9]+"]`
- ISBN numbers: 13 digits in 5 –-separted groups, first 978 or 979, last a single digit.
  - `Book` $\sqsubseteq$ `ISBN some string[length 17 ,`
    `pattern "97[89]-[0-9]+-[0-9]+-[0-9]+-[0-9]"]`
- Reasoning about patterns:
  - `str` a functional datatype property
  - $A \equiv$ `str some string[pattern "(ab)*"]`

Pattern Examples

- An integer or a string of digits
  - `xsd:integer or xsd:string[pattern "[0-9]+"]`
- ISBN numbers: 13 digits in 5 –-separted groups, first 978 or 979, last a single digit.
  - Book $\sqsubseteq$ `ISBN some string[length 17 ,`
    `pattern "97[89]-[0-9]+-[0-9]+-[0-9]+-[0-9]"]`
- Reasoning about patterns:
  - `str` a functional datatype property
  - $A \equiv$ `str some string[pattern "(ab)*"]`
  - $B \equiv$ `str some string[pattern "a(ba)*b"]`

Pattern Examples

- An integer or a string of digits
  - `xsd:integer or xsd:string[pattern "[0-9]+"]`
- ISBN numbers: 13 digits in 5 –-separted groups, first 978 or 979, last a single digit.
  - `Book` $\sqsubseteq$ `ISBN some string[length 17 ,`
    `pattern "97[89]-[0-9]+-[0-9]+-[0-9]+-[0-9]"]`
- Reasoning about patterns:
  - `str` a functional datatype property
  - $A \equiv$ `str some string[pattern "(ab)*"]`
  - $B \equiv$ `str some string[pattern "a(ba)*b"]`
  - Reasoner can find out that $B \sqsubseteq A$.

Outline

Orchestras again. . .

- **TBox:**

Orchestras again. . .

- **TBox:**

  $$Ensemble \sqsubseteq ChamberEnsemble \sqcup Orchestra$$

Orchestras again. . .

- **TBox:**

$$Ensemble \sqsubseteq ChamberEnsemble \sqcup Orchestra$$
$$ChamberEnsemble \sqsubseteq \leq_1 firstViolin.\top$$

Orchestras again. . .

- **TBox:**

$$\mathit{Ensemble} \sqsubseteq \mathit{ChamberEnsemble} \sqcup \mathit{Orchestra}$$
$$\mathit{ChamberEnsemble} \sqsubseteq \leq_1 \mathit{firstViolin}.\top$$

- **ABox:**

Orchestras again...

- **TBox:**

  $$Ensemble \sqsubseteq ChamberEnsemble \sqcup Orchestra$$
  $$ChamberEnsemble \sqsubseteq \leq_1 firstViolin.\top$$

- **ABox:**

  `Ensemble(oslo)`

Orchestras again...

- **TBox:**

  $$Ensemble \sqsubseteq ChamberEnsemble \sqcup Orchestra$$
  $$ChamberEnsemble \sqsubseteq \leq_1 firstViolin.\top$$

- **ABox:**

  ```
  Ensemble(oslo)
  firstViolin(oslo, båtnes)
  ```

Orchestras again...

- **TBox:**

  $$Ensemble \sqsubseteq ChamberEnsemble \sqcup Orchestra$$
  $$ChamberEnsemble \sqsubseteq \leq_1 firstViolin.\top$$

- **ABox:**

  ```
  Ensemble(oslo)
  firstViolin(oslo, båtnes)
  firstViolin(oslo, tønnesen)
  ```

Orchestras again...

- **TBox:**

  $$\mathit{Ensemble} \sqsubseteq \mathit{ChamberEnsemble} \sqcup \mathit{Orchestra}$$
  $$\mathit{ChamberEnsemble} \sqsubseteq \leq_1 \mathit{firstViolin}.\top$$

- **ABox:**

  ```
  Ensemble(oslo)
  firstViolin(oslo, båtnes)
  firstViolin(oslo, tønnesen)
  ```

- Want to infer: *Orchestra*(`oslo`)

Orchestras again...

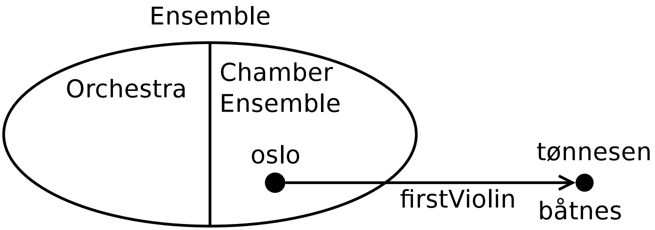
- **TBox:**

  $$Ensemble \sqsubseteq ChamberEnsemble \sqcup Orchestra$$
  $$ChamberEnsemble \sqsubseteq \leq_1 firstViolin.\top$$

- **ABox:**

  ```
  Ensemble(oslo)
  firstViolin(oslo, båtnes)
  firstViolin(oslo, tønnesen)
  ```

- Want to infer: *Orchestra*(oslo)
- But: $\texttt{tønnesen}^{\mathcal{I}} = \texttt{båtnes}^{\mathcal{I}}$



owl:differentFrom

- Need to say that båtnes and tønnesen are different

owl:differentFrom

- Need to say that `båtnes` and `tønnesen` are different
- This can be expressed with a triple
  `båtnes owl:differentFrom tønnesen`

owl:differentFrom

- Need to say that `båtnes` and `tønnesen` are different
- This can be expressed with a triple
  `båtnes owl:differentFrom tønnesen`
- **TBox:**

owl:differentFrom

- Need to say that båtnes and tønnesen are different
- This can be expressed with a triple
  båtnes owl:differentFrom tønnesen
- **TBox:**

$$\mathit{Ensemble} \sqsubseteq \mathit{ChamberEnsemble} \sqcup \mathit{Orchestra}$$

owl:differentFrom

- Need to say that båtnes and tønnesen are different
- This can be expressed with a triple
  båtnes owl:differentFrom tønnesen
- **TBox:**

$$\mathit{Ensemble} \sqsubseteq \mathit{ChamberEnsemble} \sqcup \mathit{Orchestra}$$
$$\mathit{ChamberEnsemble} \sqsubseteq \leq_1 \mathit{firstViolin}.\top$$

owl:differentFrom

- Need to say that båtnes and tønnesen are different
- This can be expressed with a triple
  båtnes owl:differentFrom tønnesen
- **TBox:**

$$Ensemble \sqsubseteq ChamberEnsemble \sqcup Orchestra$$
$$ChamberEnsemble \sqsubseteq \leq_1 firstViolin.\top$$

- **ABox:**

owl:differentFrom

- Need to say that `båtnes` and `tønnesen` are different
- This can be expressed with a triple
  `båtnes owl:differentFrom tønnesen`
- **TBox:**

$$Ensemble \sqsubseteq ChamberEnsemble \sqcup Orchestra$$
$$ChamberEnsemble \sqsubseteq \leq_1 firstViolin.\top$$

- **ABox:**

  `Ensemble(oslo)`

owl:differentFrom

- Need to say that `båtnes` and `tønnesen` are different
- This can be expressed with a triple
  `båtnes owl:differentFrom tønnesen`
- **TBox:**

$$\mathit{Ensemble} \sqsubseteq \mathit{ChamberEnsemble} \sqcup \mathit{Orchestra}$$
$$\mathit{ChamberEnsemble} \sqsubseteq \leq_1 \mathit{firstViolin}.\top$$

- **ABox:**

```
Ensemble(oslo)
firstViolin(oslo, båtnes)
```

owl:differentFrom

- Need to say that båtnes and tønnesen are different
- This can be expressed with a triple
  båtnes owl:differentFrom tønnesen
- **TBox:**

  $Ensemble \sqsubseteq ChamberEnsemble \sqcup Orchestra$
  $ChamberEnsemble \sqsubseteq \leq_1 firstViolin.\top$
- **ABox:**

  ```
  Ensemble(oslo)
  firstViolin(oslo, båtnes)
  firstViolin(oslo, tønnesen)
  ```

owl:differentFrom

- Need to say that `båtnes` and `tønnesen` are different
- This can be expressed with a triple
  `båtnes owl:differentFrom tønnesen`
- **TBox:**

$$\mathit{Ensemble} \sqsubseteq \mathit{ChamberEnsemble} \sqcup \mathit{Orchestra}$$
$$\mathit{ChamberEnsemble} \sqsubseteq \leq_1 \mathit{firstViolin}.\top$$

- **ABox:**

```
Ensemble(oslo)
firstViolin(oslo, båtnes)
firstViolin(oslo, tønnesen)
owl:differentFrom(tønnesen,båtnes)
```

owl:differentFrom

- Need to say that `båtnes` and `tønnesen` are different
- This can be expressed with a triple
  `båtnes owl:differentFrom tønnesen`
- **TBox:**

$$Ensemble \sqsubseteq ChamberEnsemble \sqcup Orchestra$$
$$ChamberEnsemble \sqsubseteq \leq_1 firstViolin.\top$$

- **ABox:**

```
Ensemble(oslo)
firstViolin(oslo, båtnes)
firstViolin(oslo, tønnesen)
owl:differentFrom(tønnesen,båtnes)
```

- ...together imply *Orchestra*(`oslo`).

owl:differentFrom

- Need to say that `båtnes` and `tønnesen` are different
- This can be expressed with a triple

  `båtnes owl:differentFrom tønnesen`
- **TBox:**

  $$Ensemble \sqsubseteq ChamberEnsemble \sqcup Orchestra$$
  $$ChamberEnsemble \sqsubseteq \leq_1 firstViolin.\top$$
- **ABox:**

  ```
  Ensemble(oslo)
  firstViolin(oslo, båtnes)
  firstViolin(oslo, tønnesen)
  owl:differentFrom(tønnesen,båtnes)
  ```
- ...together imply *Orchestra*(`oslo`).
- OWL also provides an "allDifferent" construct for whole sets

Information about Oslo

- DBpedia: `http://dbpedia.org/resource/Oslo`

Information about Oslo

- DBpedia: `http://dbpedia.org/resource/Oslo`
  - description in many languages

Information about Oslo

- DBpedia: `http://dbpedia.org/resource/Oslo`
  - description in many languages
  - `dbpprop:leaderName dbpedia:Fabian_Stang`

Information about Oslo

- DBpedia: `http://dbpedia.org/resource/Oslo`
  - description in many languages
  - `dbpprop:leaderName dbpedia:Fabian_Stang`
  - `dbpprop:aprSnowCm "3"^^xsd:double`

Information about Oslo

- DBpedia: `http://dbpedia.org/resource/Oslo`
  - description in many languages
  - `dbpprop:leaderName dbpedia:Fabian_Stang`
  - `dbpprop:aprSnowCm "3"^^xsd:double`
- Geonames: `http://sws.geonames.org/3143244/`

Information about Oslo

- DBpedia: `http://dbpedia.org/resource/Oslo`
  - description in many languages
  - `dbpprop:leaderName dbpedia:Fabian_Stang`
  - `dbpprop:aprSnowCm "3"^^xsd:double`
- Geonames: `http://sws.geonames.org/3143244/`
  - `:parentFeature http://sws.geonames.org/3143242/` (Oslo fylke)

Information about Oslo

- DBpedia: `http://dbpedia.org/resource/Oslo`
  - description in many languages
  - `dbpprop:leaderName dbpedia:Fabian_Stang`
  - `dbpprop:aprSnowCm "3"^^xsd:double`
- Geonames: `http://sws.geonames.org/3143244/`
  - `:parentFeature http://sws.geonames.org/3143242/` (Oslo fylke)
  - `:nearby http://sws.geonames.org/6697867/` (Oslo Sentrum)

Information about Oslo

- DBpedia: `http://dbpedia.org/resource/Oslo`
  - description in many languages
  - `dbpprop:leaderName dbpedia:Fabian_Stang`
  - `dbpprop:aprSnowCm "3"^^xsd:double`
- Geonames: `http://sws.geonames.org/3143244/`
  - `:parentFeature http://sws.geonames.org/3143242/` (Oslo fylke)
  - `:nearby http://sws.geonames.org/6697867/` (Oslo Sentrum)
- Freebase: `http://rdf.freebase.com/ns/guid.9202a8c...`

Information about Oslo

- DBpedia: `http://dbpedia.org/resource/Oslo`
  - description in many languages
  - `dbpprop:leaderName dbpedia:Fabian_Stang`
  - `dbpprop:aprSnowCm "3"^^xsd:double`
- Geonames: `http://sws.geonames.org/3143244/`
  - `:parentFeature http://sws.geonames.org/3143242/` (Oslo fylke)
  - `:nearby http://sws.geonames.org/6697867/` (Oslo Sentrum)
- Freebase: `http://rdf.freebase.com/ns/guid.9202a8c...`
  - `fb:local_transportation fb:en.oslo_t-bane`

Information about Oslo

- DBpedia: `http://dbpedia.org/resource/Oslo`
  - description in many languages
  - `dbpprop:leaderName dbpedia:Fabian_Stang`
  - `dbpprop:aprSnowCm "3"^^xsd:double`
- Geonames: `http://sws.geonames.org/3143244/`
  - `:parentFeature http://sws.geonames.org/3143242/` (Oslo fylke)
  - `:nearby http://sws.geonames.org/6697867/` (Oslo Sentrum)
- Freebase: `http://rdf.freebase.com/ns/guid.9202a8c...`
  - `fb:local_transportation fb:en.oslo_t-bane`
- And a couple more!

Information about Oslo

- DBpedia: `http://dbpedia.org/resource/Oslo`
    - description in many languages
    - `dbpprop:leaderName dbpedia:Fabian_Stang`
    - `dbpprop:aprSnowCm "3"^^xsd:double`
- Geonames: `http://sws.geonames.org/3143244/`
    - `:parentFeature http://sws.geonames.org/3143242/` (Oslo fylke)
    - `:nearby http://sws.geonames.org/6697867/` (Oslo Sentrum)
- Freebase: `http://rdf.freebase.com/ns/guid.9202a8c...`
    - `fb:local_transportation fb:en.oslo_t-bane`
- And a couple more!
- Many different URIs for the same resource!

Information about Oslo

- DBpedia: `http://dbpedia.org/resource/Oslo`
  - description in many languages
  - `dbpprop:leaderName dbpedia:Fabian_Stang`
  - `dbpprop:aprSnowCm "3"^^xsd:double`
- Geonames: `http://sws.geonames.org/3143244/`
  - `:parentFeature http://sws.geonames.org/3143242/` (Oslo fylke)
  - `:nearby http://sws.geonames.org/6697867/` (Oslo Sentrum)
- Freebase: `http://rdf.freebase.com/ns/guid.9202a8c...`
  - `fb:local_transportation fb:en.oslo_t-bane`
- And a couple more!
- Many different URIs for the same resource!
- How can a machine combine the information?

owl:sameAs

- Two resources can be made the same using `owl:sameAs`,e.g.
  `dbpedia:Oslo owl:sameAs geonames:3143244`

owl:sameAs

- Two resources can be made the same using `owl:sameAs`,e.g.
  `dbpedia:Oslo owl:sameAs geonames:3143244`
- Semantics: a `owl:sameAs` b is true in $\mathcal{I}$ iff $a^{\mathcal{I}} = b^{\mathcal{I}}$

owl:sameAs

- Two resources can be made the same using `owl:sameAs`,e.g.
  `dbpedia:Oslo owl:sameAs geonames:3143244`
- Semantics: a `owl:sameAs` b is true in $\mathcal{I}$ iff $a^{\mathcal{I}} = b^{\mathcal{I}}$
- Allows to infer the same, joint, information about several URIs:

$$\frac{a \texttt{ owl:sameAs } b \qquad a\ R\ c}{b\ R\ c}$$

owl:sameAs

- Two resources can be made the same using `owl:sameAs`,e.g.
  `dbpedia:Oslo owl:sameAs geonames:3143244`
- Semantics: a `owl:sameAs` b is true in $\mathcal{I}$ iff $a^{\mathcal{I}} = b^{\mathcal{I}}$
- Allows to infer the same, joint, information about several URIs:

$$\frac{a \; \texttt{owl:sameAs} \; b \qquad a\; R\; c}{b\; R\; c}$$

- Note: only for individuals. For classes, use class equivalence axioms:
  `en:Town owl:equivalentClass no:By .`

`owl:sameAs` in Practice

- Many Semantic Web sites are interlinked with `owl:sameAs`:

owl:sameAs in Practice

- Many Semantic Web sites are interlinked with owl:sameAs:
  - DBpedia

`owl:sameAs` in Practice

- Many Semantic Web sites are interlinked with `owl:sameAs`:
  - DBpedia
  - geonames

owl:sameAs in Practice

- Many Semantic Web sites are interlinked with `owl:sameAs`:
  - DBpedia
  - geonames
  - freebase

`owl:sameAs` in Practice

- Many Semantic Web sites are interlinked with `owl:sameAs`:
  - DBpedia
  - geonames
  - freebase
  - OpenCyc

`owl:sameAs` in Practice

- Many Semantic Web sites are interlinked with `owl:sameAs`:
  - DBpedia
  - geonames
  - freebase
  - OpenCyc
  - etc.

owl:sameAs in Practice

- Many Semantic Web sites are interlinked with `owl:sameAs`:
  - DBpedia
  - geonames
  - freebase
  - OpenCyc
  - etc.
- Not always both ways but often

`owl:sameAs` in Practice

- Many Semantic Web sites are interlinked with `owl:sameAs`:
  - DBpedia
  - geonames
  - freebase
  - OpenCyc
  - etc.
- Not always both ways but often
- Easy to misuse for things not quite the same

`owl:sameAs` in Practice

- Many Semantic Web sites are interlinked with `owl:sameAs`:
  - DBpedia
  - geonames
  - freebase
  - OpenCyc
  - etc.
- Not always both ways but often
- Easy to misuse for things not quite the same
  - E.g. two FOAF files at the current and a former employer

`owl:sameAs` in Practice

- Many Semantic Web sites are interlinked with `owl:sameAs`:
  - DBpedia
  - geonames
  - freebase
  - OpenCyc
  - etc.
- Not always both ways but often
- Easy to misuse for things not quite the same
  - E.g. two FOAF files at the current and a former employer
  - A `owl:sameAs` link between the two identities

`owl:sameAs` in Practice

- Many Semantic Web sites are interlinked with `owl:sameAs`:
  - DBpedia
  - geonames
  - freebase
  - OpenCyc
  - etc.
- Not always both ways but often
- Easy to misuse for things not quite the same
  - E.g. two FOAF files at the current and a former employer
  - A `owl:sameAs` link between the two identities
  - $\Rightarrow$ Two workplaces, two addresses, etc.

`owl:sameAs` in Practice

- Many Semantic Web sites are interlinked with `owl:sameAs`:
  - DBpedia
  - geonames
  - freebase
  - OpenCyc
  - etc.
- Not always both ways but often
- Easy to misuse for things not quite the same
  - E.g. two FOAF files at the current and a former employer
  - A `owl:sameAs` link between the two identities
  - $\Rightarrow$ Two workplaces, two addresses, etc.
  - OK to equate the old me and the new me?

`owl:sameAs` in Practice

- Many Semantic Web sites are interlinked with `owl:sameAs`:
  - DBpedia
  - geonames
  - freebase
  - OpenCyc
  - etc.
- Not always both ways but often
- Easy to misuse for things not quite the same
  - E.g. two FOAF files at the current and a former employer
  - A `owl:sameAs` link between the two identities
  - $\Rightarrow$ Two workplaces, two addresses, etc.
  - OK to equate the old me and the new me?
  - Temporal aspects are a weakness in sem. tek. standards!

`owl:sameAs` in Practice

- Many Semantic Web sites are interlinked with `owl:sameAs`:
  - DBpedia
  - geonames
  - freebase
  - OpenCyc
  - etc.
- Not always both ways but often
- Easy to misuse for things not quite the same
  - E.g. two FOAF files at the current and a former employer
  - A `owl:sameAs` link between the two identities
  - $\Rightarrow$ Two workplaces, two addresses, etc.
  - OK to equate the old me and the new me?
  - Temporal aspects are a weakness in sem. tek. standards!
- Can't trust `owl:sameAs` links blindly

`owl:sameAs` in Practice

- Many Semantic Web sites are interlinked with `owl:sameAs`:
  - DBpedia
  - geonames
  - freebase
  - OpenCyc
  - etc.
- Not always both ways but often
- Easy to misuse for things not quite the same
  - E.g. two FOAF files at the current and a former employer
  - A `owl:sameAs` link between the two identities
  - $\Rightarrow$ Two workplaces, two addresses, etc.
  - OK to equate the old me and the new me?
  - Temporal aspects are a weakness in sem. tek. standards!
- Can't trust `owl:sameAs` links blindly
- Linked Open Data browsers treat them like other predicates

Outline

1. Reminder: OWL
2. Cardinality restrictions
3. More about Datatypes
4. owl:sameAs and owl:differentFrom
5. Disjointness and Covering Axioms

Guys and Gals

- Try to model the relationship between the concepts

Guys and Gals

- Try to model the relationship between the concepts
  - Person

Guys and Gals

- Try to model the relationship between the concepts
  - Person
  - Man

Guys and Gals

- Try to model the relationship between the concepts
  - Person
  - Man
  - Woman

Guys and Gals

- Try to model the relationship between the concepts
  - Person
  - Man
  - Woman
- First try:

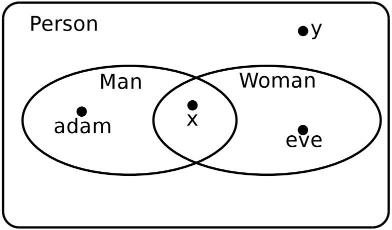
$$\begin{aligned} \mathit{Man} &\sqsubseteq \mathit{Person} \\ \mathit{Woman} &\sqsubseteq \mathit{Person} \end{aligned}$$

Guys and Gals

- Try to model the relationship between the concepts
  - Person
  - Man
  - Woman
- First try:

$$\begin{aligned} Man &\sqsubseteq Person \\ Woman &\sqsubseteq Person \end{aligned}$$

- General shape of a model:

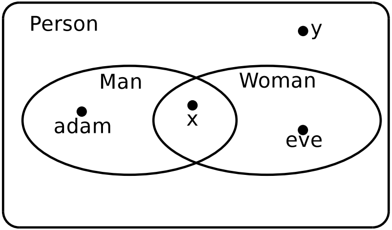


Guys and Gals

- Try to model the relationship between the concepts
  - Person
  - Man
  - Woman
- First try:

$$\begin{aligned} Man &\sqsubseteq Person \\ Woman &\sqsubseteq Person \end{aligned}$$

- General shape of a model:



- x is both *Man* and *Woman*, y is neither but a *Person*.

Disjointness Axioms

- Nothing should be both a *Man* and a *Woman*

Disjointness Axioms

- Nothing should be both a *Man* and a *Woman*
- Add a *disjointness* axiom for *Man* and *Woman*

Disjointness Axioms

- Nothing should be both a *Man* and a *Woman*
- Add a *disjointness* axiom for *Man* and *Woman*
- Equivalent possibilities:

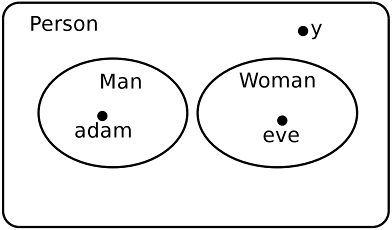
$$\mathit{Man} \sqcap \mathit{Woman} \equiv \bot$$
$$\mathit{Man} \sqsubseteq \neg \mathit{Woman}$$
$$\mathit{Woman} \sqsubseteq \neg \mathit{Man}$$

Disjointness Axioms

- Nothing should be both a *Man* and a *Woman*
- Add a *disjointness* axiom for *Man* and *Woman*
- Equivalent possibilities:

$$Man \sqcap Woman \equiv \bot$$
$$Man \sqsubseteq \neg Woman$$
$$Woman \sqsubseteq \neg Man$$

- General shape of a model:

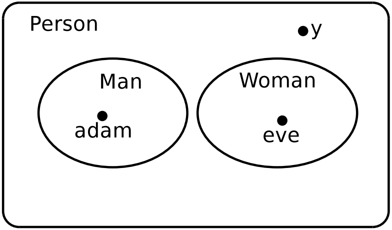


Disjointness Axioms

- Nothing should be both a *Man* and a *Woman*
- Add a *disjointness* axiom for *Man* and *Woman*
- Equivalent possibilities:

$$Man \sqcap Woman \equiv \bot$$
$$Man \sqsubseteq \neg Woman$$
$$Woman \sqsubseteq \neg Man$$

- General shape of a model:



- Specific support in OWL (`owl:disjointWith`) and Protégé

Covering Axioms

- Any *Person* should be either a *Man* or a *Woman*.

Covering Axioms

- Any *Person* should be either a *Man* or a *Woman*.
- Add a *covering axiom*

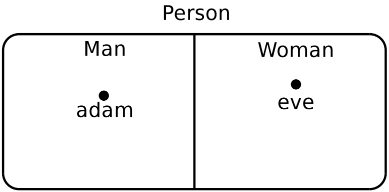
$$Person \sqsubseteq Man \sqcup Woman$$

Covering Axioms

- Any *Person* should be either a *Man* or a *Woman*.
- Add a *covering axiom*

$$Person \sqsubseteq Man \sqcup Woman$$

- General shape of a model (with disjointness!):

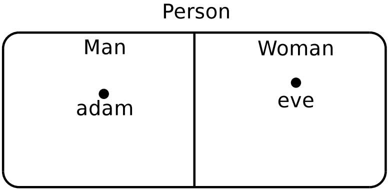


Covering Axioms

- Any *Person* should be either a *Man* or a *Woman*.
- Add a *covering axiom*

$$\mathit{Person} \sqsubseteq \mathit{Man} \sqcup \mathit{Woman}$$

- General shape of a model (with disjointness!):



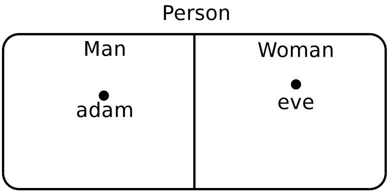
- Specific support in Protégé ("Add Covering Axiom")

Covering Axioms

- Any *Person* should be either a *Man* or a *Woman*.
- Add a *covering axiom*

$$Person \sqsubseteq Man \sqcup Woman$$

- General shape of a model (with disjointness!):



- Specific support in Protégé ("Add Covering Axiom")
- Compare to "abstract classes" in OO!

Meat and Veggies

- Careful: not all subclasses are disjoint and covering!

Meat and Veggies

- Careful: not all subclasses are disjoint and covering!
- Subclasses can be covering but not disjoint.

Meat and Veggies

- Careful: not all subclasses are disjoint and covering!
- Subclasses can be covering but not disjoint.
- E.g.

$$\begin{aligned} \mathit{MeatEatingMammal} &\sqsubseteq \mathit{Mammal} \\ \mathit{VeggieEatingMammal} &\sqsubseteq \mathit{Mammal} \end{aligned}$$

Meat and Veggies

- Careful: not all subclasses are disjoint and covering!
- Subclasses can be covering but not disjoint.
- E.g.

$$\begin{aligned} \mathit{MeatEatingMammal} &\sqsubseteq \mathit{Mammal} \\ \mathit{VeggieEatingMammal} &\sqsubseteq \mathit{Mammal} \end{aligned}$$

- All mammals eat either meat or vegetables...

Meat and Veggies

- Careful: not all subclasses are disjoint and covering!
- Subclasses can be covering but not disjoint.
- E.g.

$$\begin{aligned} \mathit{MeatEatingMammal} &\sqsubseteq \mathit{Mammal} \\ \mathit{VeggieEatingMammal} &\sqsubseteq \mathit{Mammal} \end{aligned}$$

- All mammals eat either meat or vegetables...

  $\mathit{Mammal} \sqsubseteq \mathit{MeatEatingMammal} \sqcup \mathit{VeggieEatingMammal}$

Meat and Veggies

- Careful: not all subclasses are disjoint and covering!
- Subclasses can be covering but not disjoint.
- E.g.

$$\begin{aligned} \mathit{MeatEatingMammal} &\sqsubseteq \mathit{Mammal} \\ \mathit{VeggieEatingMammal} &\sqsubseteq \mathit{Mammal} \end{aligned}$$

- All mammals eat either meat or vegetables...

  $\mathit{Mammal} \sqsubseteq \mathit{MeatEatingMammal} \sqcup \mathit{VeggieEatingMammal}$
- But there are mammals eating both...

Meat and Veggies

- Careful: not all subclasses are disjoint and covering!
- Subclasses can be covering but not disjoint.
- E.g.

$$\begin{array}{rcl} \mathit{MeatEatingMammal} & \sqsubseteq & \mathit{Mammal} \\ \mathit{VeggieEatingMammal} & \sqsubseteq & \mathit{Mammal} \end{array}$$

- All mammals eat either meat or vegetables…

  $\mathit{Mammal} \sqsubseteq \mathit{MeatEatingMammal} \sqcup \mathit{VeggieEatingMammal}$
- But there are mammals eating both…
- …in this lecture hall!

Meat and Veggies

- Careful: not all subclasses are disjoint and covering!
- Subclasses can be covering but not disjoint.
- E.g.

$$\begin{aligned} \mathit{MeatEatingMammal} &\sqsubseteq \mathit{Mammal} \\ \mathit{VeggieEatingMammal} &\sqsubseteq \mathit{Mammal} \end{aligned}$$

- All mammals eat either meat or vegetables. . .

  $\mathit{Mammal} \sqsubseteq \mathit{MeatEatingMammal} \sqcup \mathit{VeggieEatingMammal}$
- But there are mammals eating both. . .
- . . . in this lecture hall!
- No disjointness axiom for *MeatEatingMammal* and *VeggieEatingMammal*!

Cats and Dogs

- Subclasses can be disjoint but not covering.

Cats and Dogs

- Subclasses can be disjoint but not covering.
- E.g.

$$\begin{aligned} Cat &\sqsubseteq Mammal \\ Dog &\sqsubseteq Mammal \end{aligned}$$

Cats and Dogs

- Subclasses can be disjoint but not covering.
- E.g.

$$\begin{aligned} Cat &\sqsubseteq Mammal \\ Dog &\sqsubseteq Mammal \end{aligned}$$

- Nothing is both a cat and a dog...

Cats and Dogs

- Subclasses can be disjoint but not covering.
- E.g.

$$\begin{aligned} Cat &\sqsubseteq Mammal \\ Dog &\sqsubseteq Mammal \end{aligned}$$

- Nothing is both a cat and a dog...

$$Cat \sqsubseteq \neg Dog$$

Cats and Dogs

- Subclasses can be disjoint but not covering.
- E.g.

$$\begin{aligned} Cat &\sqsubseteq Mammal \\ Dog &\sqsubseteq Mammal \end{aligned}$$

- Nothing is both a cat and a dog...

$$Cat \sqsubseteq \neg Dog$$

- But there are mammals which are neither...

Cats and Dogs

- Subclasses can be disjoint but not covering.
- E.g.

$$\begin{aligned} Cat &\sqsubseteq Mammal \\ Dog &\sqsubseteq Mammal \end{aligned}$$

- Nothing is both a cat and a dog...

$$Cat \sqsubseteq \neg Dog$$

- But there are mammals which are neither...
- ...in this lecture hall!

Cats and Dogs

- Subclasses can be disjoint but not covering.
- E.g.

$$\begin{aligned} Cat &\sqsubseteq Mammal \\ Dog &\sqsubseteq Mammal \end{aligned}$$

- Nothing is both a cat and a dog...

$$Cat \sqsubseteq \neg Dog$$

- But there are mammals which are neither...
- ...in this lecture hall!
- No covering axiom for subclasses *Cat* and *Dog* of *Mammal*

Teachers and Students

- Subclasses can be neither disjoint nor covering.

Teachers and Students

- Subclasses can be neither disjoint nor covering.
- E.g.

$$\begin{aligned} \mathit{Teacher} &\sqsubseteq \mathit{Person} \\ \mathit{Student} &\sqsubseteq \mathit{Person} \end{aligned}$$

Teachers and Students

- Subclasses can be neither disjoint nor covering.
- E.g.

$$\begin{array}{rcl} Teacher & \sqsubseteq & Person \\ Student & \sqsubseteq & Person \end{array}$$

- There are people who are neither students nor teachers

Teachers and Students

- Subclasses can be neither disjoint nor covering.
- E.g.

$$\begin{aligned} \mathit{Teacher} &\sqsubseteq \mathit{Person} \\ \mathit{Student} &\sqsubseteq \mathit{Person} \end{aligned}$$

- There are people who are neither students nor teachers
- though *not* in this lecture hall!

Teachers and Students

- Subclasses can be neither disjoint nor covering.
- E.g.

$$\begin{aligned} \mathit{Teacher} &\sqsubseteq \mathit{Person} \\ \mathit{Student} &\sqsubseteq \mathit{Person} \end{aligned}$$

- There are people who are neither students nor teachers
- though *not* in this lecture hall!
- No covering axiom for these subclasses of *Person*

Teachers and Students

- Subclasses can be neither disjoint nor covering.
- E.g.

$$\begin{aligned} Teacher &\sqsubseteq Person \\ Student &\sqsubseteq Person \end{aligned}$$

- There are people who are neither students nor teachers
- though *not* in this lecture hall!
- No covering axiom for these subclasses of *Person*
- There are people who are both students and teachers

Teachers and Students

- Subclasses can be neither disjoint nor covering.
- E.g.

$$\begin{aligned} Teacher &\sqsubseteq Person \\ Student &\sqsubseteq Person \end{aligned}$$

- There are people who are neither students nor teachers
- though *not* in this lecture hall!
- No covering axiom for these subclasses of *Person*
- There are people who are both students and teachers
- E.g. most PhD students

Teachers and Students

- Subclasses can be neither disjoint nor covering.
- E.g.

$$\begin{aligned} Teacher &\sqsubseteq Person \\ Student &\sqsubseteq Person \end{aligned}$$

- There are people who are neither students nor teachers
- though *not* in this lecture hall!
- No covering axiom for these subclasses of *Person*
- There are people who are both students and teachers
- E.g. most PhD students
- No disjointness axiom for *Teacher* and *Student*!

Next Week

- Audun will take a recap:
- Some basic notions of sets and relations
- Repetition of logic, models, entailment, etc.