INF3580 – Semantic Technologies – Spring 2010

Lecture 9: More OWL, Role modeling

Audun Stolpe

23rd March 2010

Department of Informatics

University of Oslo

Om de obligatoriske oppgavene

- oblig 1 er rettet
- e-post skal være sendt ut til alle som har levert
- frist for ny levering 8. april
- kommentarer ligger ute på kursets hjemmeside
- sammen med enkelte hint til løsningen

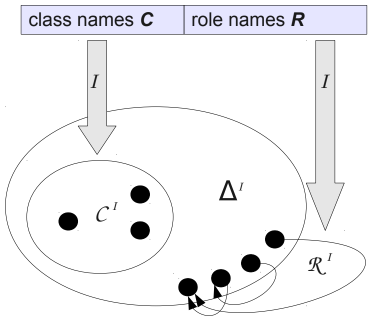
Today's Plan

1. Reminder: OWL
2. Cardinality restrictions
3. Role modeling
4. A worked example

Outline

1. Reminder: OWL
2. Cardinality restrictions
3. Role modeling
4. A worked example

Schematic representation of OWL/DL interpretations



- No reference/extension distinction
- That is, no function *IEXT*
- No properties in the domain
- Classes are sets
- Properties are relations
- Simple extensional semantics

$\mathcal{ALC}$ 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

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

$\mathcal{ALC}$ TBox and ABox

- The TBox
  - is for *terminological knowledge*
  - is independent of any actual instance data
  - is a set of $\sqsubseteq$ axioms
- 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 $R(b, c)$

Outline

We shall add

- Cardinality restrictions to the TBox
  - $\leq_n R.C$ and $\geq_n R.C$
- Equality and difference to the ABox, that is
  - `a owl:sameAs b` and `a owl:differentFrom b`, or
  - $a = b$ and $a \neq b$ in logic notation
- An 'RBox', that is
  - Role characteristics
  - Role relationships
- Note that
  - An ontology consists of classes *and roles*
  - Axioms in the TBox may affect roles
  - Role axioms may affect classes
  - Talk of boxes should not be taken too literally

The $\mathcal{ALCQ}$ Description Logic

$\mathcal{ALCQ}$ concept descriptions

$$\begin{array}{rlll} C, D \rightarrow & A & \mid & \text{(atomic concept)} \\ & \top & \mid & \text{(universal concept)} \\ & \bot & \mid & \text{(bottom concept)} \\ & \neg C & \mid & \text{(atomic negation)} \\ & C \sqcap D & \mid & \text{(intersection)} \\ & C \sqcup D & \mid & \text{(union)} \\ & \forall R.C & \mid & \text{(value restriction)} \\ & \exists R.C & \mid & \text{(existential restriction)} \\ & \leq_n R.C & \mid & \text{(cardinality restriction)} \end{array}$$

$\mathcal{ALCQ}$ Semantics

Interpretation of concept descriptions

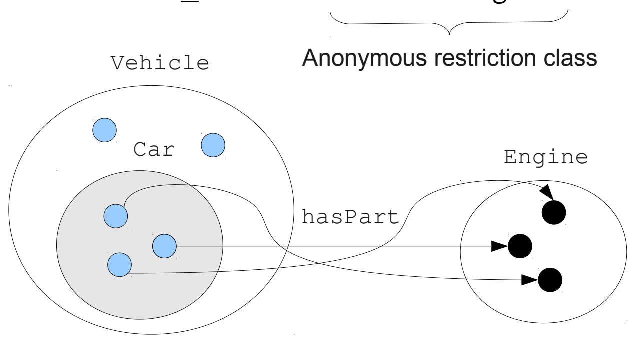
$$\begin{array}{rcl} \top^{\mathcal{I}} & = & \Delta^{\mathcal{I}} \\ \bot^{\mathcal{I}} & = & \emptyset \\ (\neg C)^{\mathcal{I}} & = & \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}} \\ (C \sqcap D)^{\mathcal{I}} & = & C^{\mathcal{I}} \cap D^{\mathcal{I}} \\ (C \sqcup D)^{\mathcal{I}} & = & C^{\mathcal{I}}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

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

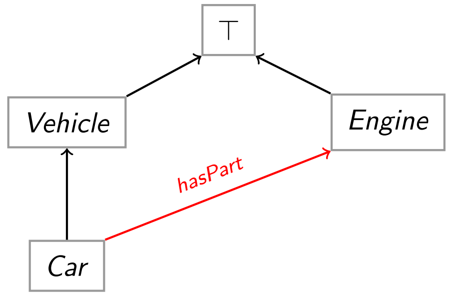
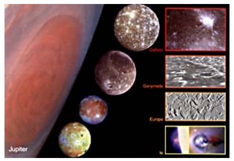


Figure: Connecting classes

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:

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

A tempting mistake

Cardinality restrictions cannot be used to reason with

- durations
- intervals
- or any kind of sequence
- and it cannot be used for arithmetic

Anti-pattern:

- Scotch whisky is casked for more than three years:
- $Scotch \sqsubseteq Whisky \sqcap \geq_3 casked.Years$

Why?

- The class *Years* is just a set of objects
- they are not necessarily related, except by type
- the axiom may be satisfied by any random collection of years
- $\geq_{12} casked.Years$ is not *longer* than $\geq_3 casked.Years$

Cardinalities, non-unique names and open worlds

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

TBox:

$Ensemble \sqsubseteq ChamberEnsemble \sqcup Orchestra$

$ChamberEnsemble \sqsubseteq \leq_1 firstViolin.\top$

ABox:

`firstViolin(oslo, båtnes)`

`firstViolin(oslo, tønnesen)`

That is;

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

Musical taxons

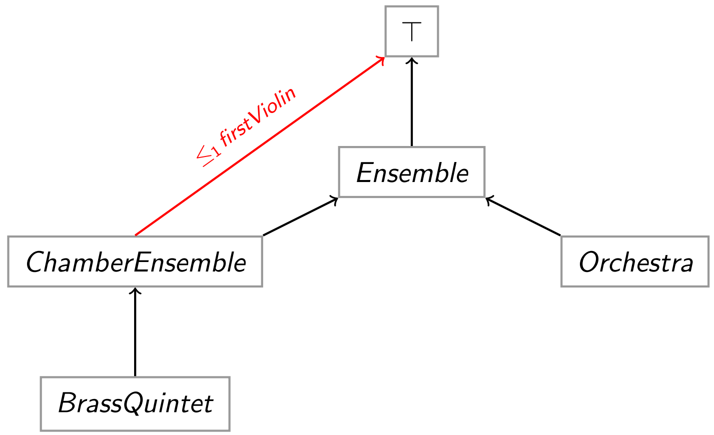


Figure: An ontology of ensembles

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
- According to which we may not know everything about `oslo`
- in particular there may be other first violinists

Outline

Role characteristics and relationships

Role characteristics are mathematical properties of roles.

- A role can be:
  - reflexive/irreflexive
  - symmetric/asymmetric
  - transitive
  - functional/inverse functional

Role relationships: Roles R and S can be

- declared *disjoint*, meaning that $R^{\mathcal{I}} \cap S^{\mathcal{I}} = \emptyset$
- related as *inverses*, meaning that $S^{\mathcal{I}} = (R^{-})^{\mathcal{I}}$
- subsumed under each other, meaning that $R^{\mathcal{I}} \subseteq S^{\mathcal{I}}$
- chained, e.g. $R^{\mathcal{I}} \circ S^{\mathcal{I}} \subseteq S^{\mathcal{I}}$

Corresponding mathematical properties and operations

A relation R over a set X is

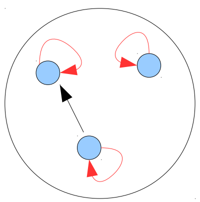
Reflexive:	if $(a, a) \in R$ for all $a \in X$
Irreflexive:	if $a \in X$ implies $(a, a) \notin R$
Symmetric:	if $(a, b) \in R$ implies $(b, a) \in R$
Asymmetric:	if $(a, b) \in R$ implies $(b, a) \notin R$
Transitive:	if $(a, b), (b, c) \in R$ implies $(a, c) \in R$
Functional:	if $(a, b), (a, c) \in R$ implies $b = c$
Inverse functional:	if $(a, b), (c, b) \in R$ implies $a = c$

If R and S are binary relations on X then

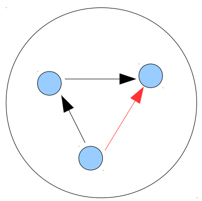
$(\mathbf{a}, \mathbf{c}) \in \mathbf{R} \circ \mathbf{S}$:	if $(a, b) \in R$ and $(b, c) \in S$ for some $b \in X$
$(\mathbf{b}, \mathbf{a}) \in \mathbf{R}^{-}$:	if $(a, b) \in R$.

Relation diagrams

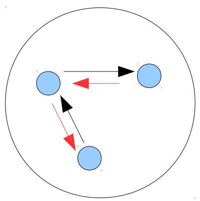
A reflexive relation:



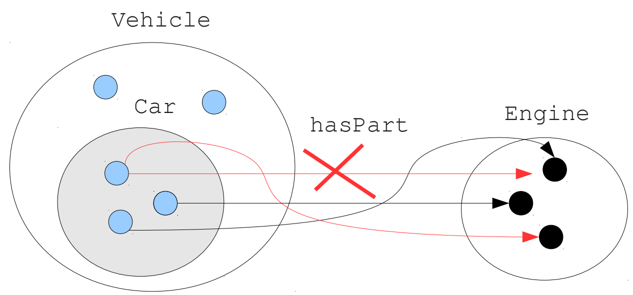
A transitive relation:



A symmetric relation:

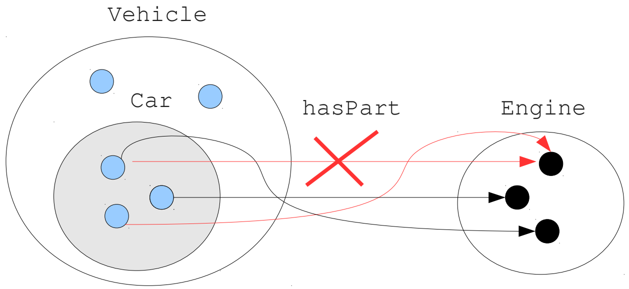


Functionality



A (normal) car doesn't have more than one engine

Inverse functionality

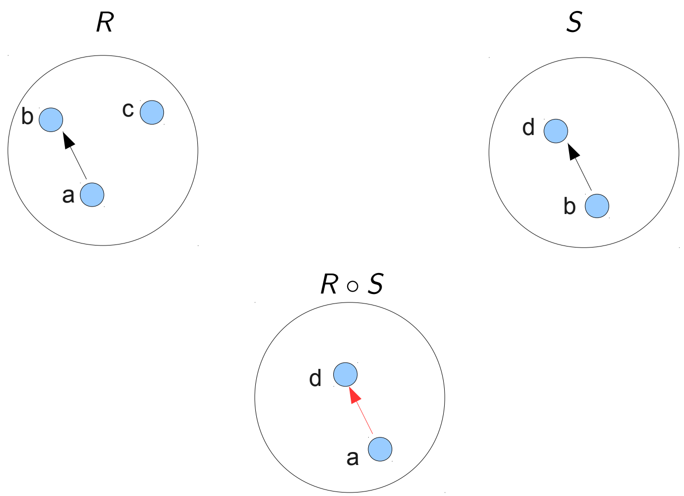


An engine doesn't sit in more than one car (simultaneously)

Some role relationships: Inverses

Inverse roles R and R^{-}.

Chaining of roles



Some relations from ordinary language

- Symmetric relations:
  - _ sibling of _
  - _ different from _
- *Non*-symmetric relations:
  - _ brother of _
  - _ likes _
- Asymmetric relations:
  - _ taller than _ (under a strict interpretation)
  - _ member of _
- Transitive relations:
  - _ taller than _
  - _ part of _ (under certain qualifications)
- Functional relations:
  - _ was born by _
- Inverse functional relations:
  - _ gave birth to _

Som inverses and chains

Some inverses:

- Uncle/nephew
- Gave birth to/was born by
- To the left of/to the right of
- Taller than/shorter than
- etc.

Some role chains:

- $\texttt{fatherOf} \circ \texttt{brotherOf} \sqsubseteq \texttt{uncleOf}$
- $\texttt{isLocatedIn} \circ \texttt{isPartOf} \sqsubseteq \texttt{isLocatedIn}$

Datatype properties and object properties

OWL enforces a separation between datatype- and object properties:

Object properties:

- Also known as *abstract roles*
- connect objects with objects
- Example in Turtle syntax:

```
foaf:knows a owl:ObjectProperty .
```

Datatype properties:

- Also known as *concrete roles*
- connect objects with literal values, i.e. with elements of datatypes.
- Example in Turtle-syntax:

```
ex:age a owl:DatatypeProperty .
ex:age rdfs:range xsd:positiveInteger .
```


Datatype properties and existential restrictions

Datatype properties:

- May be used in existential restrictions too ..
- to define membership conditions for other classes

Example–defining a class `Teenager`:

- Add a property age as on the previous slide.
- Add an existential restriction that sets the age range.
- In Manchester syntax:

```
Person and (age some positiveInteger[>= 13, <= 19])
```


Characteristics of datatype properties

Datatype properties cannot be

- reflexive, or they would not be datatype properties
- transitive, since literals cannot be subjects of triples
- symmetric, for the same reason
- inverse functional, for computational reasons

In fact, as of today datatype properties may only be functional

Quirks

Role modeling in OWL 2 can get excessively complicated

For instance:

- transitive roles cannot be irreflexive or asymmetric
- role inclusions are not allowed to cycle, i.e. not

  $\texttt{hasParent} \circ \texttt{hasHusband} \sqsubseteq \texttt{hasFather}$

  $\texttt{fasFather} \sqsubseteq \texttt{hasParent}$
- transitive roles *R* and *S* cannot be declared disjoint

Note

- these restrictions can be hard to keep track of
- the reason they exist are computational, not logical

Fortunately:

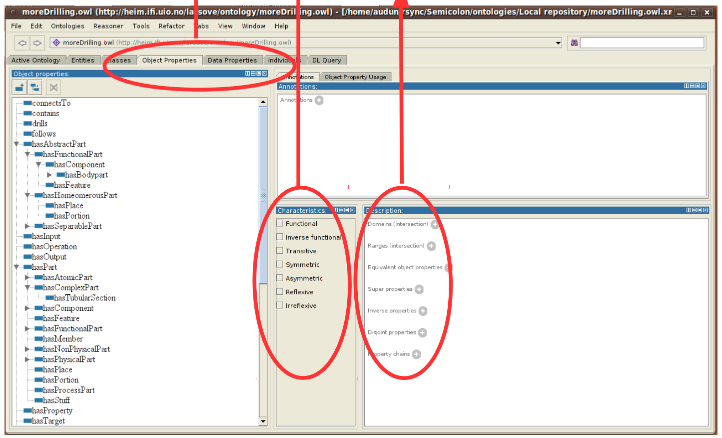
- There are also *simple* patterns ..
- that are extremely useful

Managing roles in Protege

Object/datatype property tabs

Role characteristics

Domain/range, role relationships



Outline

Merging data from databases

Information in a table can be encoded as RDF:

The recipe is:

1. Come up with a URI for the database as such, and in this namespace:
   - Make each row in the table a resource,
   - construct the resource name from the table name and the primary key
2. make each cell a triple where
   - the resource corresponding to the row is the subject of the triple
   - the predicate name is constructed from the table and column name
   - the cell value is the object of the triple

This is called *exposing RDBs as RDF* and can be done by several tools:

For instance:

- D2RQ
- SquirrelRDF
- OpenLink Virtuoso

Desirable features

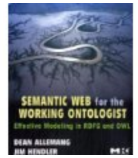
These tools have one or more of the following features

- the data is exposed as *virtual RDF*,
- that is, conversion is on-demand rather than up-front
- they offer general-purpose mapping from RDB to ontology
- that is, tables can be mapped to classes of one's own choosing
- and columns can be mapped to properties

D2RQ, for one, has all features.

Example: Merging product infromation

The example is an adaptation from Allemang and Hendler: "Semantic Web for the Working Ontologist":



Suppose we want to integrate product information, and that

- data is stored in two different tables
- in two different databases
- one contains information about the product per se
- and the other about the facilities needed to produce them

Table excerpts I

Product				
ID	**Model Number**	**Division**	**Manufacture Location**	**Available**
1	ZX-3	Manufacturing	Sacramento	23
2	ZX-3P	Manufacturing	Seoul	14
3	ZX-3S	Support	Hong Kong	100
4	B1430X	Engineering	Elizabeth	14
5	B1431	Control	Hong Kong	4
6	DBB-12	Accessories	Cleveland	87

Figure: Table of products

Table excerpts II

Product		
ID	**Model Number**	**Facility**
1	B1430X	Assembly Center
2	1180-M	Machine Shop
3	TC-43	Factory
4	ZX-3P	Factory
5	B1431	Assembly Center
6	SP-1234	Machine Shop

Figure: Parts and the facilities required to produce them

The RDF encoding

There are $5 \times 6 = 30$ triples for the first table, among others

Manufacture location triples

```
mf:Product1 mf:Product_Manufacture_location "Sacramento" .
mf:Product2 mf:Product_Manufacture_location "Seoul" .
mf:Product3 mf:Product_Manufacture_location "Hong Kong" .
mf:Product4 mf:Product_Manufacture_location "Elizabeth" .
mf:Product5 mf:Product_Manufacture_location "Hong Kong" .
mf:Product6 mf:Product_Manufacture_location "Cleveland" .
```

We assume that `mf:` abbreviates the namespace of the database.

.. contd

Similarly there are $3 \times 6 = 18$ triples for the second table, among others

Production facility triples

```
p:Product1 p:Product_Facility "Assembly Center" .
p:Product2 p:Product_Facility "Machine Shop" .
p:Product3 p:Product_Facility "Factory" .
p:Product4 p:Product_Facility "Factory" .
p:Product5 p:Product_Facility "Assembly Center" .
p:Product6 p:Product_Facility "Machine Shop" .
```

We assume that `p:` abbreviates the namespace of the database.

The challenge

We wish to integrate the two tables, so that e.g.

- places (i.e. manufacture locations) can be correlated with production facilities

However, we would like to do so in manner such that

- we do not have to go through the rows one-by-one
- in a manual editing process

Rather we would like to

- Specify a set of general relationships between the respective columns
- that enables a reasoner to *infer* the correlations whenever they exist

Solution

This can be solved by a two-step procedure:

1. Declare the respective **Model Number** columns equivalent properties:
   - if a product x has a `mf:Model_Number` value of "ZX-3P"
   - then x also has the same value for `p:Model_Number`

- This can be done manully, by adding the following triples:

```
mf:Product_Number rdfs:subPropertyOf p:Product_Number .
p:Product_Number rdfs:subPropertyOf mf:Product_Number .
```

- or it can be done in Protegé

solution contd.

2. Declare one property to be *inverse functional*

- The range of such a property can be considered a set of unique keys
- i.e. elements of the range provide unique identifiers for each element of the domain.

Thus,

- If, say, `mf:Model_Number` is declared to be inverse functional,
- then records with the same `mf:Model_Number` represent the same product,

Inverse functionality,

- can be declared manually by adding a triple such as

```
mf:Model_Number a owl:InverseFunctionalProperty .
```

- or one can simply check the appropriate box in the Protegé GUI

A sample trace

A SPARQL query

```
SELECT ?location ?facility WHERE{
        ?product mf:Manufacture_Location ?location .
        ?product p:Product_Facility ?facility.
       }
```

- SPARQL finds `mf:Product4`
- which has `mf:Manufacture_Location` "Hong Kong"
- and `mf:Product_Number` **B1431**

trace contd.

- **B1431** is also the `p:Product_Number` of `p:Product5`
- these properties are equivalent
- and `mf:Product_Number` is inverse funtional
- so it follows that `p:Product5` is the same product as `mf:Product4`
- now, `p:Product5` has `p:Product_Facility` "Assembly Center",
- so, `mf:Product4` also has `p:Product_Facility` "Assembly Center"
- So ("Hong Kong", "Assembly Center") is a solution for the query

Other common role modeling patterns

- Transitivity and reflexivity for ordering relations, e.g.
  - the mereological notion of part-whole
  - being a part of a part of is being a part of
  - everything is part of itself
- Inversely related ordering relations, e.g.
  - `hasPart` and `partOf`
  - if a has b as a part then b is a part of a
- Asymmetry for strict ordering relations, e.g.
  - the mereological `isProperPartOf`
  - if a is a proper part of b then b cannot be a proper part of a
- Functional properties where sameness should be inferred, e.g.
  - the `hasFather` relation,
  - where fathers may be known by different names