INF3580 – Semantic Technologies – Spring 2011

Lecture 12: OWL: Loose Ends

Martin Giese

12th April 2011

Department of Informatics
University of Oslo

Today's Plan

1. Reminder: OWL
2. Disjointness and Covering Axioms
3. Keys
4. More about Datatypes
5. What can't be expressed in OWL 2

Outline

1. Reminder: OWL
2. Disjointness and Covering Axioms
3. Keys
4. More about Datatypes
5. What can't be expressed in OWL 2

$\mathcal{ALCQ}$ 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

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

OWL 2 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 $\equiv$
    - roles symmetric, asymmetric, reflexive, irreflexive, transitive,...
    - roles functional, inverse functional
    - inverse roles: $hasParent = hasChild^{-1}$
    - role inclusion $hasBrother \sqsubseteq hasSibling$
    - role chains $hasParent \circ hasBrother \sqsubseteq hasUncle$
  - Only certain combinations allowed!
- The ABox
  - is for *assertional knowledge*
  - contains facts about concrete instances $a, b, c, \ldots$
  - A set of (negative) concept assertions $C(a)$, $\neg D(b)$ ...
  - and (negative) role assertions $R(b, c)$, $\neg S(a, b)$
  - also `owl:sameAs`: $a = b$
  - and `owl:differentFrom`: $a \neq b$

Nominals, Self-restrictions

- Sometimes, all elements of a class are known, and can be given in a list.
- Allow concept expressions $\{a, b, c\}$
- Does not imply that a, b, c are different!
- $Weekdays \equiv \{mon, tue, wed, thu, fri, sat, sun\}$
- r `value` x shorthand for $\exists R.\{x\}$

- The class of things related to themselves by R:
- $\exists R.Self$
- All people who know themselves: $Person \sqcap \exists knows.Self$
- Manchester Syntax: `Person and knows Self`

A Strange Catalogue

- We have seen many nice things that can be said in OWL
- Why the strange restrictions, e.g. on role axioms?
- Why not use 1st-order logic, could say much more?

- Because of the reasoning!
  - Class satisfiability ($C \not\equiv \bot$)
  - Classification ($C \sqsubseteq D$)
  - Instance Check ($C(a)$)
  - ...
- All *decidable*
- Algorithm gives a correct answer after finite time

- Add a little more to OWL, and this is lost!

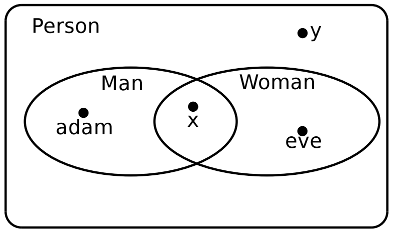
Outline

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:



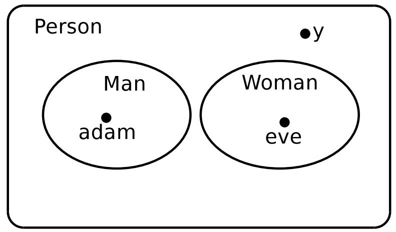
- x is both *Man* and *Woman*, y is neither but a *Person*.

Disjointness Axioms

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



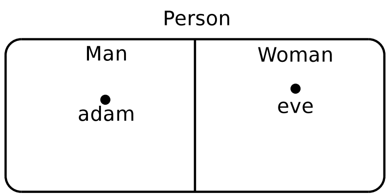
- Specific support in OWL (`owl:disjointWith`) and Protégé

Covering Axioms

- Any *Person* should be either a *Man* or a *Woman*.
- 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!
- Subclasses can be covering but not disjoint.
- E.g.

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

- All mammals eat either meat or vegetables...

  $Mammal \sqsubseteq MeatEatingMammal \sqcup VeggieEatingMammal$
- 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.
- E.g.

$$\begin{array}{rcl} Cat & \sqsubseteq & Mammal \\ Dog & \sqsubseteq & Mammal \end{array}$$

- 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.
- E.g.

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

- 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*!

Outline

Keys

- A Norwegian is uniquely identified by his/her "personnummer"
  - Different Norwegians have different numbers
- Each customer in the DB is uniquely identified by the customer ID
  - No two customers with the same customer ID
  - Referred to as a *key* for a database table.
- A course is uniquely determined by code, semester, year.
  - E.g. ⟨INF3580, Spring, 2011⟩
- R is a key for some set A if for all $x, y \in A$

$$xRk \quad \text{and} \quad yRk \quad \text{imply} \quad x = y$$

- That's the same as R^{-1} being functional:

$$kR^{-1}x \quad \text{and} \quad kR^{-1}y \quad \text{imply} \quad x = y$$

- So R is a key if it is "inverse functional"
  - There is a function giving exactly one object for every key value

OWL Keys

- Keys in applications are usually (tuples of) literals
- In OWL: inverse functional datatype properties
- Reasoning about these is problematic!
- Therefore, datatype properties cannot be declared inverse functional in OWL 2

- OWL 2 includes special “hasKey” axioms
- Example: `Course hasKey {hasCode, hasSemester, hasYear}`
- Works for object properties and datatype properties.
- OWL Keys apply only to explicitly *named instances*
  - Makes reasoning tractable.

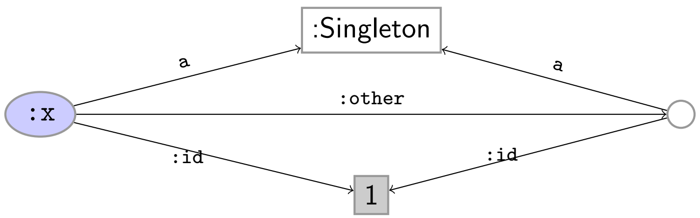
Reasoning with OWL Keys

- Given:
  - `:Norwegian hasKey {:personnr}`
  - `:drillo a :Norwegian`
  - `:drillo :personnr "12345698765"`
  - `:egil a :Norwegian`
  - `:egil :personnr "12345698765"`
- Can infer:
  - `:drillo owl:sameAs :egil`

- Given:
  - `:Singleton hasKey {:id}`
  - `:Singleton` $\sqsubseteq$ `:id value 1`
  - `:x a :Singleton`
  - `:y a :Singleton`
- Can infer:
  - `:x owl:sameAs :y`

What’s with the “named instances”?

- Given:
  - `:Singleton hasKey {:id}`
  - `:Singleton` $\sqsubseteq$ `:id value 1`
  - `:x a :Singleton`
  - `:Singleton` $\sqsubseteq$ `:other some (:Singleton and not {:x})`



- *not* inconsistent, since the blank node is not “named”!
- Distinct keys only required for explicitly named individuals.

Outline

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

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

A possible solution

- Idea: don't use age.
- 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

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
- 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 ≤ 5.
  - `xsd:string[minLength 5]` – strings of length ≥ 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]`
- 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
  - `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

Expressivity

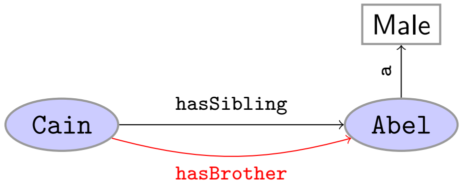
- Any concept or property can be described in OWL
- Maybe not *totally*, with all its aspects
- Might not be needed or meaningful
- Remember: working with *abstractions*
- Certain *relationships* between concepts and properties can't be expressed in OWL
- E.g.
  - Given that property *hasSibling* and class *Male* are defined. . .
  - . . . cannot say that $hasBrother(x, y)$ iff $hasSibling(x, y)$ and $Male(y)$.
- Usually, adding such missing relationships would lead to undecidability
- *Not* easy to show that something is not expressible
  - We look at some examples, not proofs

Brothers

- Given terms

$$hasSibling \qquad Male$$

- . . . a brother is *defined* to be a sibling who is male



- Best try:

$$hasBrother \sqsubseteq hasSibling$$
$$\forall hasBrother.Male \qquad \text{or: rg}(hasBrother, Male)$$
$$\exists hasSibling.Male \sqsubseteq \exists hasBrother.\top$$

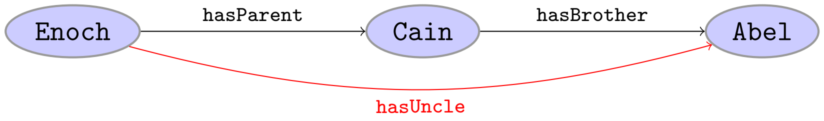
- Not enough to infer that *all* male siblings are brothers!
  - (probably mostly an "accident" in the OWL 2 specification)

Uncles

- Given terms

$$hasParent \qquad hasBrother$$

- . . . an uncle is *defined* to be a brother of a parent.



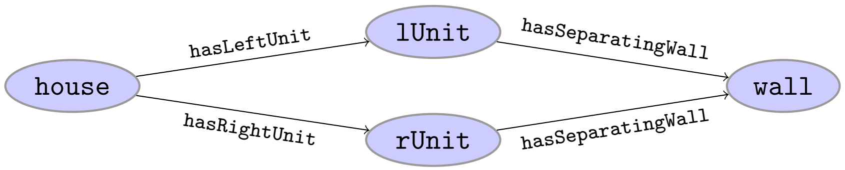
- Best try:

$$hasParent \circ hasBrother \sqsubseteq hasUncle$$
$$hasUncle \sqsubseteq hasParent \circ hasBrother$$

- properties cannot be declared sub-properties of property chains.
  - (can become problematic for reasoning in some constellations)

Diamond Properties

- A semi-detached house has a left and a right unit
- Each unit has a separating wall
- The separating walls of the left and right units are the same
- "diamond property"



- Try. . .

$$SemiDetached \sqsubseteq \exists hasLeftUnit.Unit \sqcap \exists hasRightUnit.Unit$$
$$Unit \sqsubseteq \exists hasSeparatingWall.Wall$$

- And now what?

Connecting Datatype Properties

- Given terms

$$Person \qquad hasChild \qquad hasBirthday$$

- A twin parent is defined to be a person who has two children with the same birthday.
- Try. . .

$$TwinParent \equiv Person \sqcap \exists hasChild.\exists hasBirthday[\ldots] \sqcap \exists hasChild.\exists hasBirthday[\ldots]$$

- No way to connect the two birthdays to say that they're the same.
  - (and no way to say that the children are *not* the same)
- Try. . .

$$TwinParent \equiv Person \sqcap \geq_2 hasChild.\exists hasBirthday[\ldots]$$

- Still no way of connecting the birthdays!

Reasoning about Numbers

- Reasoning about natural numbers is undecidable in general.
- DL Reasoning is decidable
- Therefore, general reasoning about numbers can't be "encoded" in DL
- For instance

$$\forall n.\exists p.(p > n \land \forall k, l.p = k \cdot l \rightarrow (k = 1 \lor l = 1))$$

- (There is no largest prime number)
- Could try...

$$Number(zero)$$
$$Number \sqsubseteq \exists hasSuccessor.Number$$

- Cannot encode addition, multiplication, etc.
- Note: a lot can be done with other logics, but not with DLs
  - Outside the intended scope of Description Logics

After the Easter Holidays

- More (practical) details about SPARQL
- RDF on the Web: Linked Open Data and RDFa
- Exporting relational databases as RDF with D2R
- Guest lecture: commercial projects with RDF