IN3060/4060 – Semantic Technologies – Spring 2021

Lecture 11: OWL: Loose Ends

Jieying Chen

26th March 2021

DEPARTMENT OF
INFORMATICS

UNIVERSITY OF
OSLO

Mandatory exercises

- Oblig 6 published after lecture.
- First attempt by April 16th.
- Second attempt two weeks after feedback.

Outline

Make it simple!

- “Data level” with resources
- “Ontology level” with properties and “classes”
- Can have `rdf:type` relation between data objects and classes
- Allow a fixed vocabulary for relations between classes and properties
- Interpret:
  - Class as set of data objects
  - Property as relation between data objects

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: $\mathit{hasParent} \equiv \mathit{hasChild}^{-1}$
    - role inclusion $\mathit{hasBrother} \sqsubseteq \mathit{hasSibling}$
    - role chains $\mathit{hasParent} \circ \mathit{hasBrother} \sqsubseteq \mathit{hasUncle}$
  - Only certain combinations allowed

OWL 2 TBox and ABox

- 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`: $a \neq b$.

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?

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

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

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

Single and Married

- Try to model the relationship between the concepts *Person*, *Married* and *Single*:
- First try:

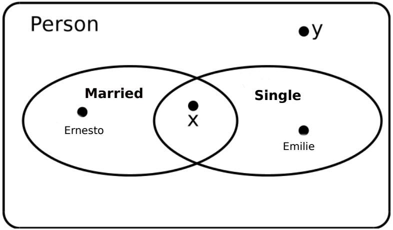
$$\begin{aligned} \mathit{Single} &\sqsubseteq \mathit{Person} \\ \mathit{Married} &\sqsubseteq \mathit{Person} \end{aligned}$$

Single and Married

- Try to model the relationship between the concepts *Person*, *Married* and *Single*:
- First try:

$$\begin{aligned} \mathit{Single} &\sqsubseteq \mathit{Person} \\ \mathit{Married} &\sqsubseteq \mathit{Person} \end{aligned}$$

- General shape of a model:



- x is both *Single* and *Married*, y is neither but a *Person*.

Disjointness Axioms

- Nothing should be both a *Single* and a *Married*
- Add a *disjointness* axiom for *Single* and *Married*
- Equivalent possibilities:

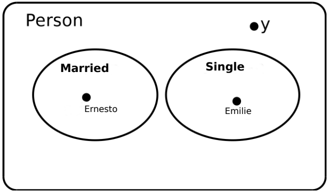
$$\begin{aligned} \mathit{Single} \sqcap \mathit{Married} &\equiv \bot \\ \mathit{Single} &\sqsubseteq \neg \mathit{Married} \\ \mathit{Married} &\sqsubseteq \neg \mathit{Single} \end{aligned}$$

Disjointness Axioms

- Nothing should be both a *Single* and a *Married*
- Add a *disjointness* axiom for *Single* and *Married*
- Equivalent possibilities:

$$\mathit{Single} \sqcap \mathit{Married} \equiv \bot$$
$$\mathit{Single} \sqsubseteq \neg \mathit{Married}$$
$$\mathit{Married} \sqsubseteq \neg \mathit{Single}$$

- General shape of a model:



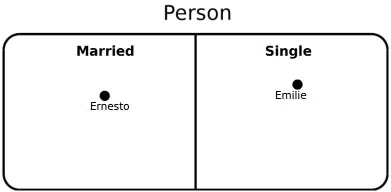
- Specific support in OWL (`owl:disjointWith`) and Protégé

Covering Axioms

- Any *Person* should be either *Single* or *Married*.
- Add a *covering axiom* $Person \sqsubseteq Married \sqcup Single$

Covering Axioms

- Any *Person* should be either *Single* or *Married*.
- Add a *covering axiom* $Person \sqsubseteq Married \sqcup Single$
- General shape of a model (with disjointness):



- Specific support in Protégé (Edit Menu: "Add Covering Axiom")

Meat and Veggies

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

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

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
- No disjointness axiom for *MeatEatingMammal* and *VeggieEatingMammal*

Cats and Dogs

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

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

Cats and Dogs

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

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

- Nothing is both a cat and a dog: $\mathit{Cat} \sqsubseteq \neg \mathit{Dog}$

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
- No covering axiom with subclasses *Cat* and *Dog* for *Mammal*

Teachers and Students

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

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

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 a student nor a teacher

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 a student nor a teacher
- 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 a student nor a teacher
- 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.

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

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

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 a student nor a teacher
- though *not* in this lecture hall
- No covering axiom for these subclasses of *Person*
- There are people who are both a student and a teacher
- 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 a student nor a teacher
- though *not* in this lecture hall
- No covering axiom for these subclasses of *Person*
- There are people who are both a student and a teacher
- E.g. most PhD students
- No disjointness axiom for *Student* and *Teacher*

Outline

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

Keys

- A Norwegian is uniquely identified by his/her “fødselsnummer”
  - Different Norwegians have different numbers

Keys

- A Norwegian is uniquely identified by his/her “fødselsnummer”
    - 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.

Keys

- A Norwegian is uniquely identified by his/her “fødselsnummer”
    - 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. $\langle$IN3060/4060, Spring, 2021$\rangle$

Keys

- A Norwegian is uniquely identified by his/her “fødselsnummer”
    - 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. ⟨IN3060/4060, Spring, 2021⟩
- hasKey: if two named instances of the class coincide on values for each of key properties, then these two individuals are the same.

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 a student nor a teacher
- though *not* in this lecture hall
- No covering axiom for these subclasses of *Person*
- There are people who are both a student and a teacher
- E.g. most PhD students
- No disjointness axiom for *Student* and *Teacher*

Outline

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

OWL 2 Keys

- 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.
  - It may not be supported by all OWL 2 reasoners

Reasoning with OWL Keys

- Given:
  - `:Norwegian hasKey {:personnr}`
  - `:david a :Norwegian`
  - `:david :personnr "12345698765"`
  - `:davidC a :Norwegian`
  - `:davidC   :personnr "12345698765"`

Reasoning with OWL Keys

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

Reasoning with OWL Keys

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

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

Reasoning with OWL Keys

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

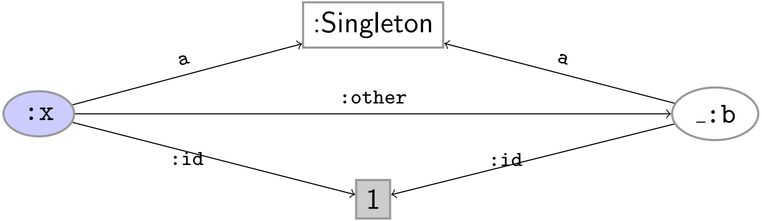
- Given:
  - `:Singleton hasKey {:id}`
  - :Singleton $\sqsubseteq$:id value 1
  - `:x a :Singleton`
  - `:y a :Singleton`
- Can infer:
  - `:x owl:sameAs :y`

What's with the "named instances"?

- Given:
  - `:Singleton hasKey {:id}`
  - :Singleton $\sqsubseteq$:id value 1
  - `:x a :Singleton`
  - :Singleton $\sqsubseteq$:other some :Singleton

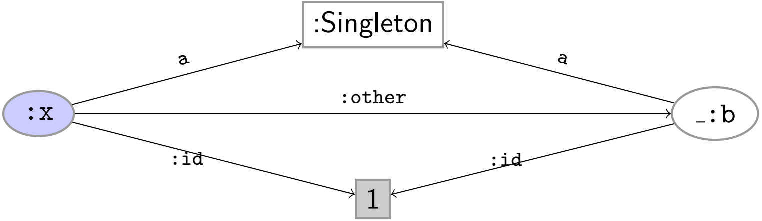
What's with the "named instances"?

- Given:
  - `:Singleton hasKey {:id}`
  - `:Singleton` $\sqsubseteq$ `:id value 1`
  - `:x a :Singleton`
  - `:Singleton` $\sqsubseteq$ `:other some :Singleton`



What's with the "named instances"?

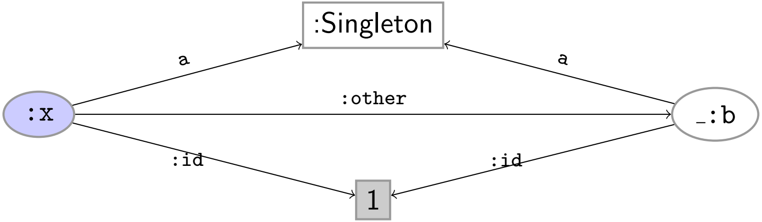
- Given:
  - `:Singleton hasKey {:id}`
  - `:Singleton ⊑ :id value 1`
  - `:x a :Singleton`
  - `:Singleton ⊑ :other some :Singleton`



- Since `_:b` is a blank node, and therefore not an explicitly named instance,
- the reasoner does not infer `:x owl:sameAs _:b`.

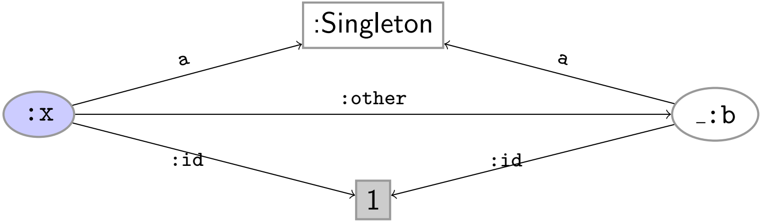
What's with the "named instances"?

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



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})`



- This is *not* inconsistent.
- Distinct keys only required for explicitly named individuals.

Outline

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

Punning

- Motivation Example:

```
(1):Service     rdf:type    owl:Class .
(2):Person      rdf:type    owl:Class .
(3)  s1         rdf:type    :Service .
(4)  s1            :input   :Person .
```

Punning

- Motivation Example:
```
(1):Service     rdf:type    owl:Class .
(2):Person      rdf:type    owl:Class .
(3)  s1         rdf:type    :Service .
(4)  s1            :input   :Person .
```
- Remember: In OWL strict separation of classes, properties and individuals. However, not entirely correct...

Punning

- Motivation Example:

```
(1):Service     rdf:type    owl:Class .
(2):Person      rdf:type    owl:Class .
(3)  s1         rdf:type    :Service .
(4)  s1            :input   :Person .
```

- Remember: In OWL strict separation of classes, properties and individuals. However, not entirely correct...
- OWL 2 introduces *punning*, allowing one URI to be used for, e.g., both a class and an individual, both an individual and property
- Restriction: not both a class and a datatype property, or for different property types.
- Example:

```
(1):Joe       rdf:type     :Eagle .
(2):Eagle     rdf:type     :Species .
```

`:Eagle` is both a class and an individual.

Punning

- Motivation Example:
  ```
  (1):Service      rdf:type    owl:Class .
  (2):Person       rdf:type    owl:Class .
  (3)  s1          rdf:type    :Service .
  (4)  s1             :input   :Person .
  ```
- Remember: In OWL strict separation of classes, properties and individuals. However, not entirely correct...
- OWL 2 introduces *punning*, allowing one URI to be used for, e.g., both a class and an individual, both an individual and property
- Restriction: not both a class and a datatype property, or for different property types.
- Example:
  ```
  (1):Joe      rdf:type    :Eagle .
  (2):Eagle    rdf:type    :Species .
  ```
  `:Eagle` is both a class and an individual.
- However, semantically, "punned" URI are treated as different terms. (under the hood)
  - Meaning, the class `:Eagle` is different from the individual `:Eagle`.
  - Axioms about the class is not transferred to the individual, or vice versa.

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

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?
  - 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.

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`

- $Scotch \sqsubseteq Whisky \sqcap \geq_3 casked.int$
- 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.

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
  - $Town \equiv \geq_{10000} inhabitant.Person$
  - $Metropolis \equiv \geq_{1000000} inhabitant.Person$
  - 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 built-in datatypes for literals
  - Numbers: real, rational, integer, positive integer, double, long,. . .
  - Strings
  - Booleans
  - Binary data
  - IRIs
  - Time Instants
  - XML Literals
- Varying tool support (e.g., depending on editor and reasoner)

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*
  - `xsd:integer[`$\geq$ `9]` – integers ≥ 9.
  - `xsd:integer[`$\geq$ `9,` $\leq$ `11]` – integers between 9 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]`

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 numberInhabitants 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 numberInhabitants some integer[>= 1000000]`

- Note: often makes best sense with functional properties
  Why?

Pattern Examples

- An integer or a string of digits
  - `xsd:integer or xsd:string[pattern "[0-9]+"]`
- ISBN numbers: 13 digits in 5 “–”-separated 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 "–"-separated 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:
  - `R` a functional datatype property
  - $A \equiv$ `R some string[pattern "(ab)*"]`
  - $B \equiv$ `R some string[pattern "a(ba)*b"]`
  - Reasoner can find out that $B \sqsubseteq A$.

Outline

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

Expressivity

- 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

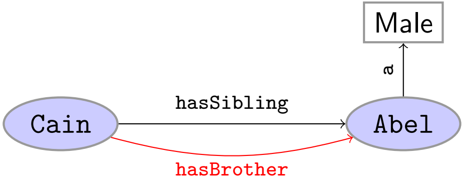
  hasSibling *Male*

Brothers

- Given terms

hasSibling *Male*

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

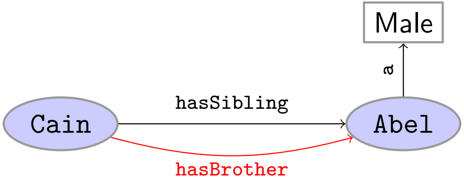


Brothers

- Given terms

$$hasSibling \qquad Male$$

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



- Best try:

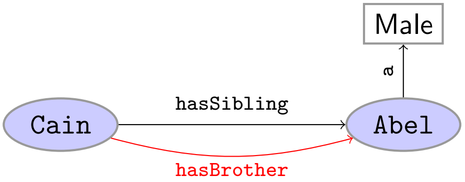
$$\begin{aligned} &hasBrother \sqsubseteq hasSibling \\ &\top \sqsubseteq \forall hasBrother.Male \qquad \text{or: } \mathrm{rg}(hasBrother, Male) \\ &\exists hasSibling.Male \sqsubseteq \exists hasBrother.\top \end{aligned}$$

Brothers

- Given terms

$$hasSibling \qquad Male$$

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



- Best try:

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

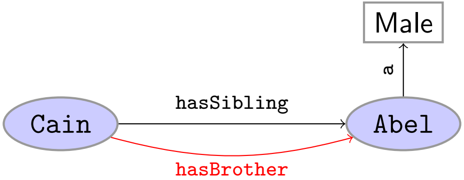
- Not enough to infer that one's male sibling is one's brother.

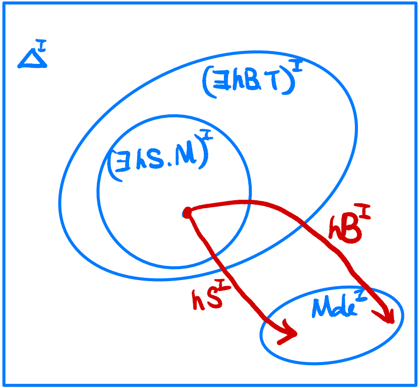
Brothers

- Given terms

$$\mathit{hasSibling} \qquad \mathit{Male}$$

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





- Best try:

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

- Not enough to infer that one's male sibling is one's brother.

Uncles

- Given terms

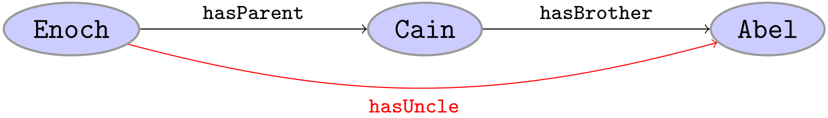
hasParent *hasBrother*

Uncles

- Given terms

hasParent *hasBrother*

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

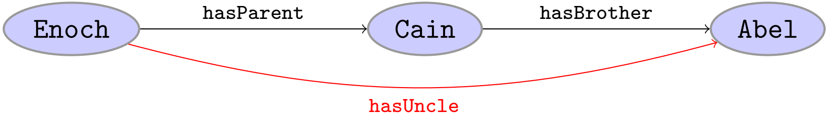


Uncles

- Given terms

$$hasParent \qquad hasBrother$$

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



- Best try:

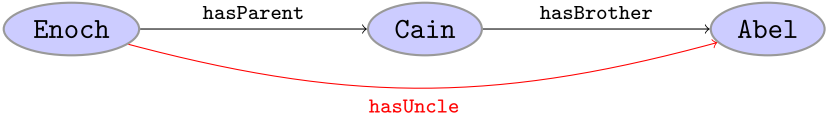
$$\begin{aligned} hasParent \circ hasBrother &\sqsubseteq hasUncle \\ hasUncle &\sqsubseteq hasParent \circ hasBrother \end{aligned}$$

Uncles

- Given terms

$$hasParent \qquad hasBrother$$

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



- Best try:

$$\begin{aligned} hasParent \circ hasBrother &\sqsubseteq hasUncle \\ hasUncle &\sqsubseteq hasParent \circ hasBrother \end{aligned}$$

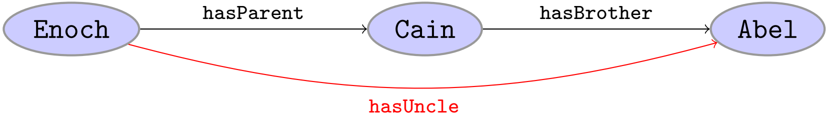
- properties cannot be declared sub-properties of property chains in OWL 2.

Uncles

- Given terms

$$\mathit{hasParent} \qquad \mathit{hasBrother}$$

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



- Best try:

$$\begin{aligned} \mathit{hasParent} \circ \mathit{hasBrother} &\sqsubseteq \mathit{hasUncle} \\ \mathit{hasUncle} &\sqsubseteq \mathit{hasParent} \circ \mathit{hasBrother} \end{aligned}$$

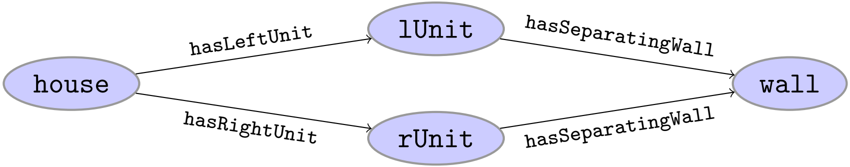
- properties cannot be declared sub-properties of property chains in OWL 2.
  - problematic for reasoning

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

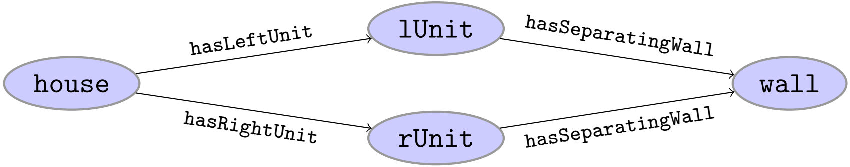
Diamond Properties

- 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"



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"



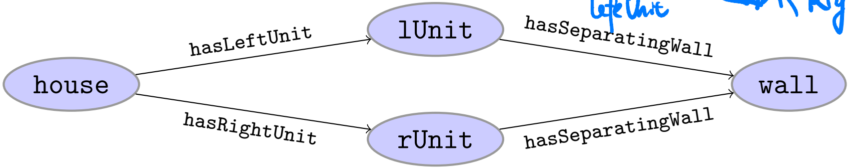
- Try. . .

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

- But this does not guarantee to use the same wall

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...

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

- But this does not guarantee to use the same wall

Connecting Datatype Properties

- Given terms

  Person *hasChild* *hasBirthday*

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

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...

$$\begin{aligned} TwinParent \equiv Person \quad & \sqcap \; \exists hasChild.\exists hasBirthday[\ldots] \\ & \sqcap \; \exists hasChild.\exists hasBirthday[\ldots] \end{aligned}$$

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...

$$\begin{aligned} TwinParent \equiv Person \quad & \sqcap \ \exists hasChild.\exists hasBirthday[\ldots] \\ & \sqcap \ \exists hasChild.\exists hasBirthday[\ldots] \end{aligned}$$

- 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)

Connecting Datatype Properties

- Given terms

$$\mathit{Person} \qquad \mathit{hasChild} \qquad \mathit{hasBirthday}$$

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

$$\begin{aligned}\mathit{TwinParent} \equiv \mathit{Person} \quad & \sqcap\ \exists \mathit{hasChild}.\exists \mathit{hasBirthday}[\ldots] \\ & \sqcap\ \exists \mathit{hasChild}.\exists \mathit{hasBirthday}[\ldots]\end{aligned}$$

- 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...

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

- 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
- 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

Combining OWL 2 and Rules

Some limitation may be addressed

- SWRL: Semantic Web Rule Language
- Uses XML syntax based on RuleML
- OWL 2 + unrestricted SWRL leads to undecidability
- Restricted SWRL + OWL is decidable and very powerful
- A bit more in the next SPARQL lesson

Outline

1. Reminder: OWL
2. Disjointness and Covering Axioms
3. Keys
4. Punning
5. More about Datatypes
6. What can’t be expressed in OWL 2
7. **OWL 2 profiles**

OWL 2 profiles

- OWL 2 has various *profiles* that correspond to different DLs.
- OWL 2 DL is the “normal” OWL 2 (sublanguage): “maximum” expressiveness while keeping reasoning problems decidable—but still very expensive.

OWL 2 profiles

- OWL 2 has various *profiles* that correspond to different DLs.
- OWL 2 DL is the “normal” OWL 2 (sublanguage): “maximum” expressiveness while keeping reasoning problems decidable—but still very expensive.
- (Other) profiles are tailored for specific ends, e.g.,
  - OWL 2 QL:
    - Specifically designed for efficient database integration.
  - OWL 2 EL:
    - A lightweight language with polynomial time reasoning.
  - OWL 2 RL:
    - Designed for compatibility with rule-based inference tools.

OWL 2 profiles

- OWL 2 has various *profiles* that correspond to different DLs.
- OWL 2 DL is the “normal” OWL 2 (sublanguage): “maximum” expressiveness while keeping reasoning problems decidable—but still very expensive.
- (Other) profiles are tailored for specific ends, e.g.,
  - OWL 2 QL:
    - Specifically designed for efficient database integration.
  - OWL 2 EL:
    - A lightweight language with polynomial time reasoning.
  - OWL 2 RL:
    - Designed for compatibility with rule-based inference tools.
- OWL Full: Anything goes: classes, relations, individuals, ... like in RDFS, are not kept apart. Highly expressive, not decidable. But we want OWL’s reasoning capabilities, so stay away if you can—and you almost always can.

OWL 2 Validator: `http://mowl-power.cs.man.ac.uk:8080/validator/`

OWL EL

Based on DL $\mathcal{EL}^{++}$.

$\mathcal{EL}^{++}$ concept descriptions, simplified

$$
\begin{array}{rlcl}
C, D \rightarrow & A & \mid & \text{(atomic concept)} \\
& \top & \mid & \text{(universal concept)} \\
& \bot & \mid & \text{(bottom concept)} \\
& \{a\} & \mid & \text{(}\textit{singular}\text{ enumeration)} \\
& C \sqcap D & \mid & \text{(intersection)} \\
& \exists R.C & \mid & \text{(existential restriction)}
\end{array}
$$

Axioms

- $C \sqsubseteq D$ and $C \equiv D$ for concept descriptions D and C.
- $R \sqsubseteq S$, $R \equiv S$, $R \circ S \sqsubseteq R$, $dom(R) \sqsubseteq C$ and $ran(R) \sqsubseteq D$ for concept descriptions D, C and roles R, S.
- $C(a)$ and $R(a, b)$ for concept C, role R and individuals a, b.

OWL EL contd.

Not supported, simplified:

- negation, (NB, disjointness of classes: $C \sqcap D \sqsubseteq \bot$ possible),
- disjunction,
- universal quantification,
- cardinalities,
- inverse roles,
- plus some role characteristics.
- reduced list of datatypes (e.g., not supported "boolean" nor "double")

Complete list: `http://www.w3.org/TR/owl2-profiles/#Feature_Overview`.

OWL EL contd.

Not supported, simplified:

- negation, (NB, disjointness of classes: $C \sqcap D \sqsubseteq \bot$ possible),
- disjunction,
- universal quantification,
- cardinalities,
- inverse roles,
- plus some role characteristics.
- reduced list of datatypes (e.g., not supported “boolean” nor “double”)

Complete list: `http://www.w3.org/TR/owl2-profiles/#Feature_Overview`.

- Checking ontology consistency, class expression subsumption, and instance checking is in **P**.
- “Good for large ontologies.”
- Used in many biomedical ontologies (e.g. SNOMED CT).

OWL QL

Based on DL-Lite$_R$.

DL-Lite$_R$ concept descriptions

$$\begin{array}{lcl} B & := & A \mid \exists R.\top \\ C & := & B \mid \neg B \end{array}$$

DL-Lite$_R$ role descriptions

$$\begin{array}{lcl} Q & := & R \mid R^{-} \\ S & := & Q \mid \neg Q \end{array}$$

DL-Lite$_R$ Axioms

- Concept inclusions $B \sqsubseteq C$ for concept descriptions B and C.
- Role inclusions $Q \sqsubseteq S$ for roles Q, S.
- $A(a)$ and $R(a, b)$ for atomic concept A, role R and individuals a, b.

OWL QL contd.

Not supported, simplified:

- disjunction,
- universal quantification,
- cardinalities,
- functional roles, keys,
- $=$ (`SameIndividual`)
- enumerations (closed classes),
- subproperties of chains, transitivity
- reduced list of datatypes (e.g., not supported "boolean" nor "double")

Complete list: `http://www.w3.org/TR/owl2-profiles/#Feature_Overview_2`.

OWL QL contd.

Not supported, simplified:

- disjunction,
- universal quantification,
- cardinalities,
- functional roles, keys,
- $=$ (`SameIndividual`)
- enumerations (closed classes),
- subproperties of chains, transitivity
- reduced list of datatypes (e.g., not supported "boolean" nor "double")

Complete list: `http://www.w3.org/TR/owl2-profiles/#Feature_Overview_2`.

- Captures language for which queries can be translated to SQL.
- "Good for large datasets."

OWL RL

- Puts constraints in the way in which constructs are used (i.e., syntactic subset of OWL 2).
- So that OWL 2 RL axioms can be directly translated into datalog rules.
- Enables desirable computational properties using rule-based reasoning engines.

OWL RL

- Puts constraints in the way in which constructs are used (i.e., syntactic subset of OWL 2).
- So that OWL 2 RL axioms can be directly translated into datalog rules.
- Enables desirable computational properties using rule-based reasoning engines.
- It is closely related to Description Logic Programs (DLP).
- Syntactic restriction on Class Expressions in OWL 2 RL can be found: `http://www.w3.org/TR/owl2-profiles/#Feature_Overview_3`.
- Supports all axioms of OWL 2 apart from disjoint unions of classes (DisjointUnion) and reflexive object property axioms (ReflexiveObjectProperty).

OWL RL

- Puts constraints in the way in which constructs are used (i.e., syntactic subset of OWL 2).
- So that OWL 2 RL axioms can be directly translated into datalog rules.
- Enables desirable computational properties using rule-based reasoning engines.
- It is closely related to Description Logic Programs (DLP).
- Syntactic restriction on Class Expressions in OWL 2 RL can be found: `http://www.w3.org/TR/owl2-profiles/#Feature_Overview_3`.
- Supports all axioms of OWL 2 apart from disjoint unions of classes (DisjointUnion) and reflexive object property axioms (ReflexiveObjectProperty).
- Reasoning in RL is possible in polynomial time, but in other cases results may be incomplete.

Next

- 9 April: SPARQL 1.1
- 16 April: RDF Validation
- 23 April: Application in Norway (Aibel, DNV)
- 30 April, 7 May: OTTR Templates (T.B.A.)
- 14 May: Open RDF Data