Knowledge Representation for the Semantic Web
Lecture 4: Description Logics III

Daria Stepanova

slides based on Reasoning Web 2011 tutorial “Foundations of Description Logics and OWL” by S. Rudolph

Max Planck Institute for Informatics
D5: Databases and Information Systems group

WS 2017/18
Unit Outline

Modeling

Description Logics and OWL
Modeling with DLs: Motivating Examples

- individual *angelina* belongs to the set of all actors:
Modeling with DLs: Motivating Examples

- individual *angelina* belongs to the set of all actors:

\[ \text{Actor(angelina)} \]
Modeling with DLs: Motivating Examples

- individual *angelina* belongs to the set of all actors:
  \[ \text{Actor}(\text{angelina}) \]

- individuals *angelina* and *brad* are in the relation of being married:
Modeling with DLs: Motivating Examples

- individual *angelina* belongs to the set of all actors:
  \[ \text{Actor}(\text{angelina}) \]
- individuals *angelina* and *brad* are in the relation of being married:
  \[ \text{married}(\text{angelina}, \text{brad}) \]
Modeling with DLs: Motivating Examples

- individual *angelina* belongs to the set of all actors:

  \[ \text{Actor}(\text{angelina}) \]

- individuals *angelina* and *brad* are in the relation of being married:

  \[ \text{married}(\text{angelina}, \text{brad}) \]

- every actor is an artist:
Modeling with DLs: Motivating Examples

- individual *angelina* belongs to the set of all actors:
  \[ \text{Actor}(\text{angelina}) \]

- individuals *angelina* and *brad* are in the relation of being married:
  \[ \text{married}(\text{angelina}, \text{brad}) \]

- every actor is an artist:
  \[ \text{Actor} \sqsubseteq \text{Artist} \]
  \[ \forall x. \text{Actor}(x) \rightarrow \text{Artist}(x) \]
Modeling with DLs: Motivating Examples

- individual *angelina* belongs to the set of all actors:
  \[ \text{Actor}(\text{angelina}) \]

- individuals *angelina* and *brad* are in the relation of being married:
  \[ \text{married}(\text{angelina}, \text{brad}) \]

- every actor is an artist:
  \[ \text{Actor} \sqsubseteq \text{Artist} \]
  \[ \forall x. \text{Actor}(x) \rightarrow \text{Artist}(x) \]

- every actor who is a US governor is also a bodybuilder or not Austrian:
Modeling with DLs: Motivating Examples

- individual *angelina* belongs to the set of all actors:
  \[ \text{Actor}(angelina) \]

- individuals *angelina* and *brad* are in the relation of being married:
  \[ \text{married}(angelina, brad) \]

- every actor is an artist:
  \[ \text{Actor} \sqsubseteq \text{Artist} \]
  \[ \forall x. \text{Actor}(x) \rightarrow \text{Artist}(x) \]

- every actor who is a US governor is also a bodybuilder or not Austrian:
  \[ \text{Actor} \sqcap \text{USGovernor} \sqsubseteq \text{Bodybuilder} \sqcup \neg \text{Austrian} \]
  \[ \forall x. (\text{Actor}(x) \land \text{USGovernor}(x)) \rightarrow (\text{BodyBuilder}(x) \lor \neg \text{Austrian}(x)) \]
Modeling with DLs: Motivating Examples, cont’d.

- everybody knowing some actor has only envious friends:
Modeling with DLs: Motivating Examples, cont’d.

- everybody knowing some actor has only envious friends:

\[
\exists \text{knows}.\text{Actor} \sqsubseteq \forall \text{hasfriend}.\text{Envious}
\]

\[
\forall x (\exists y (\text{knows}(x, y) \land \text{Actor}(y)) \rightarrow \\
\forall z (\text{hasfriend}(x, z) \rightarrow \text{Envious}(z)))
\]
Modeling with DLs: Motivating Examples, cont’d.

- everybody knowing some actor has only envious friends:
  \[ \exists \text{knows}. \text{Actor} \sqsubseteq \forall \text{hasfriend}. \text{Envious} \]

\[ \forall x (\exists y (\text{knows}(x, y) \land \text{Actor}(y)) \rightarrow \forall z (\text{hasfriend}(x, z) \rightarrow \text{Envious}(z))) \]

- everybody having a child is the child of only grandparents:
  \[ \exists \text{hasChild}. \top \sqsubseteq \forall \text{hasChild} \sqsubseteq \forall \text{Grandparent} \]

\[ \forall x (\exists y (\text{hasChild}(x, y)) \rightarrow \forall z (\text{hasChild}(z, x) \rightarrow \text{Grandparent}(x))) \]
Modeling with DLs: Motivating Examples, cont’d.

- everybody knowing some actor has only envious friends:

\[ \exists \text{knows}. \text{Actor} \sqsubseteq \forall \text{hasfriend}. \text{Envious} \]

\[ \forall x (\exists y (\text{knows}(x, y) \land \text{Actor}(y)) \rightarrow \forall z (\text{hasfriend}(x, z) \rightarrow \text{Envious}(z))) \]

- everybody having a child is the child of only grandparents:

\[ \exists \text{hasChild}. \top \sqsubseteq \forall \text{hasChild}.\neg \text{Grandparent} \]

\[ \forall x (\exists y (\text{hasChild}(x, y)) \rightarrow \forall z (\text{hasChild}(z, x) \rightarrow \text{Grandparent}(x))) \]
Modeling with DLs: Motivating Examples, cont’d.

- everybody knowing some actor has only envious friends:
  \[ \exists \text{knows}.\text{Actor} \sqsubseteq \forall \text{hasfriend}.\text{Envious} \]
  \[ \forall x (\exists y (\text{knows}(x, y) \land \text{Actor}(y)) \rightarrow \\
  \forall z (\text{hasfriend}(x, z) \rightarrow \text{Envious}(z))) \]

- everybody having a child is the child of only grandparents:
  \[ \exists \text{hasChild}.\top \sqsubseteq \forall \text{hasChild}^{-}.\text{Grandparent} \]
  \[ \forall x (\exists y (\text{hasChild}(x, y)) \rightarrow \\
  \forall z (\text{hasChild}(z, x) \rightarrow \text{Grandparent}(x))) \]

- a polygamist is married to at least two distinct individuals:
Modeling with DLs: Motivating Examples, cont’d.

- everybody knowing some actor has only envious friends:
  \[ \exists \text{knows} . \text{Actor} \sqsubseteq \forall \text{hasfriend} . \text{Envious} \]
  \[ \forall x (\exists y (\text{knows}(x, y) \land \text{Actor}(y)) \rightarrow \forall z (\text{hasfriend}(x, z) \rightarrow \text{Envious}(z))) \]

- everybody having a child is the child of only grandparents:
  \[ \exists \text{hasChild} . \top \sqsubseteq \forall \text{hasChild}^- . \text{Grandparent} \]
  \[ \forall x (\exists y (\text{hasChild}(x, y)) \rightarrow \forall z (\text{hasChild}(z, x) \rightarrow \text{Grandparent}(x))) \]

- a polygamist is married to at least two distinct individuals:
  \[ \text{Polygamist} \sqsubseteq \geq 2 \text{marriedTo}. \top \]
  \[ \forall x (\text{Polygamist}(x) \rightarrow \exists y \exists z (\text{marriedTo}(x, y) \land \text{marriedTo}(x, z) \land y \neq z)) \]
Modeling with DLs: Motivating Examples, cont’d.

- being married to Brad is a property only applying to Angelina:

\[ \exists \text{marriedTo} \{ \text{brad} \} \sqsubseteq \{ \text{angelina} \} \]

- being married to somebody implies loving them:

\[ \exists x (\text{marriedTo}(x, \text{brad}) \rightarrow x = \text{angelina}) \]

- the child of somebody I am a child of is my sibling:

\[ \forall x \forall y \forall z (\text{hasChild}(y, x) \land \text{hasChild}(y, z) \rightarrow \text{hasSibling}(x, z)) \]
Modeling with DLs: Motivating Examples, cont’d.

- being married to Brad is a property only applying to Angelina:

\[
\exists_{\text{marriedTo.}} \{ \text{brad} \} \subseteq \{ \text{angelina} \}
\]

\[
\exists x (\text{marriedTo}(x, \text{brad}) \rightarrow x = \text{angelina})
\]
Modeling with DLs: Motivating Examples, cont’d.

- being married to Brad is a property only applying to Angelina:

\[ \exists \text{marriedTo.} \{ \text{brad} \} \subseteq \{ \text{angelina} \} \]

\[ \exists x (\text{marriedTo}(x, \text{brad}) \rightarrow x = \text{angelina}) \]

- being married to somebody implies loving them:
Modeling with DLs: Motivating Examples, cont’d.

- being married to Brad is a property only applying to Angelina:
  \[ \exists \text{marriedTo}. \{ \text{brad} \} \subseteq \{ \text{angelina} \} \]
  \[ \exists x \ (\text{marriedTo}(x, \text{brad}) \rightarrow x = \text{angelina}) \]

- being married to somebody implies loving them:
  \[ \text{marriedTo} \subseteq \text{loves} \]
  \[ \forall x \forall y \ \text{married}(x, y) \rightarrow \text{loves}(x, y) \]
Modeling with DLs: Motivating Examples, cont’d.

- being married to Brad is a property only applying to Angelina:
  \[ \exists \text{marriedTo}. \{ \text{brad} \} \sqsubseteq \{ \text{angelina} \} \]
  \[ \exists x (\text{marriedTo}(x, \text{brad}) \rightarrow x = \text{angelina}) \]

- being married to somebody implies loving them:
  \[ \text{marriedTo} \sqsubseteq \text{loves} \]
  \[ \forall x \forall y \text{married}(x, y) \rightarrow \text{loves}(x, y) \]

- the child of somebody I am a child of is my sibling:

\[ \text{hasChild} \sqsubseteq \text{hasChild} \sqcap \text{hasSibling} \]
\[ \forall x \forall y \forall z (\text{hasChild}(y, x) \land \text{hasChild}(y, z) \rightarrow \text{hasSibling}(x, z)) \]
Modeling with DLs: Motivating Examples, cont’d.

- being married to Brad is a property only applying to Angelina:
  \[ \exists \text{marriedTo.}\{\text{brad}\} \sqsubseteq \{\text{angelina}\} \]
  \[ \exists x (\text{marriedTo}(x, \text{brad}) \rightarrow x = \text{angelina}) \]

- being married to somebody implies loving them:
  \[ \text{marriedTo} \sqsubseteq \text{loves} \]
  \[ \forall x \forall y \text{ married}(x, y) \rightarrow \text{loves}(x, y) \]

- the child of somebody I am a child of is my sibling:
  \[ \text{hasChild} \circ \text{hasChild} \sqsubseteq \text{hasSibling} \]
  \[ \forall x \forall y \forall z (\text{hasChild}(y, x) \land \text{hasChild}(y, z) \rightarrow \text{hasSibling}(x, z)) \]
Frequent Modeling Features

- domain
- range
- or
- concept disjointness
- or
- role symmetry
- role transitivity

\[
\exists \text{authorOf}. \top \sqsubseteq \text{Person} \\
\top \sqsubseteq \forall \text{authorOf}. \text{Publication} \\
\exists \text{authorOf}^\sim. \top \sqsubseteq \text{Publication} \\
\text{Male} \sqcap \text{Female} \sqsubseteq \bot \\
\text{Male} \sqsubseteq \neg \text{Female} \\
\text{marriedWith} \sqsubseteq \text{marriedWith}^\sim \\
\text{partOf} \circ \text{partOf} \sqsubseteq \text{partOf}
\]
Number Restrictions

- allow for defining that a role is functional

\[ \top \sqsubseteq \leq 1 \text{hasFather}.\top \]

- ...or inverse functional

\[ \top \sqsubseteq \leq 1 \text{hasFather}^{-}.\top \]

- allow for enforcing an infinite domain

\[ (\forall \text{succ}^{-}.\bot) (\text{zero}) \quad \top \sqsubseteq \exists \text{succ}.\top \quad \top \sqsubseteq \leq 1.\text{succ}^{-}.\top \]

- Consequently, DLs with number restrictions and inverses do not have the finite model property.
Nominal Concept and Universal Role

- allow to restrict the size of concepts

\[ AtMostTwo \subseteq \{\text{one, two}\} \quad AtMostTwo \subseteq \leq 2u.\top \]

- even allow to restrict the size of the domain

\[ \top \subseteq \{\text{one, two}\} \quad \top \subseteq \leq 2u.\top \]
Self-Restriction

- allows to define a role as reflexive

$$\top \subseteq \exists \text{knows} \cdot \text{Self}$$

- allows to define a role as irreflexive

$$\exists \text{betterThan} \cdot \text{Self} \subseteq \bot$$

- together with number restrictions, we can even axiomatize equality

$$\top \subseteq \exists \text{equals} \cdot \text{Self} \quad \top \subseteq \leq 1 \exists \text{equals} \cdot \top$$
Axioms vs. Constraints

**Note:** GCIs may not serve as constraints that eliminate models.

- every employee must have a social security number (SSN)
  - informal constraint: “if employee $x$ has no social security number, then infer contradiction (falsity)"
  - transcribed into a DL axiom: $\text{Employee} \sqcap \neg \exists \text{hasSSN} \sqsubseteq \bot$
  - let $KB = \{\text{Employee}(Joe), \alpha\}$
    - this knowledge base is **consistent**!
    - it assigns informally a null-value to *Joe’s* SSN
      - $\alpha$ is logically equivalent to $\text{Employee} \sqsubseteq \exists \text{hasSSN}$
    - no possibility to express (independent of concrete data) that every employee has a *known* SSN
Axioms vs. Constraints, cont’d

Uniqueness constraints might not eliminate models either

- define that \( \text{hasSSN} \) is inverse functional, and add some data:

- for

\[
KB = \left\{ \top \sqsubseteq \leq 1 \, \text{hasSSN}^- \cdot \top, \right.
\]

\[
\text{hasSSN}(\text{Joe}, 4711), \quad \text{hasSSN}(\text{Jeff}, 4711)
\]

we can conclude that \( KB \models \text{Joe} \approx \text{Jeff} \)

- there is no **Unique Name Assumption (UNA)** by default

**Reminder: UNA**

If \( c_1 \) and \( c_2 \) are two individuals such that \( c_1 \neq c_2 \), then \( c_1^\top \neq c_2^\top \)
Description Logics and OWL
Resource Description Framework (RDF): triple statements \((S, P, O)\)

RDF Schema (RDFS): simple classes/property taxonomies

Web Ontology Language (OWL): more constructs, based on DLs

SPARQL is a RDF query language (also used for RDFS and OWL)

RIF is a rule interchange format
Semantic Web

- Resource Description Framework (RDF): triple statements \((S, P, O)\)
- RDF Schema (RDFS): simple classes/property taxonomies
- Web Ontology Language (OWL): more constructs, based on DLs
- SPARQL is a RDF query language (also used for RDFS and OWL)
- RIF is a rule interchange format
OWL Ontologies

- **2004**: Web Ontology Language (OWL): W3C standard
- Knowledge about concepts, individuals, their properties and relationships

OWL syntax is based on RDF, common: RDF/XML, turtle syntax (in this lecture turtle is used just for demonstration)
OWL Ontologies

- **2004**: Web Ontology Language (OWL): W3C standard
- Knowledge about concepts, individuals, their properties and relationships
- Three increasingly expressive sublanguages

- **OWL Lite**: concept hierarchies, simple constraint features \(\iff SHIF\) with datatypes
- **OWL DL**: nominals, number restriction \(\iff SHOIN\) with datatypes\(^a\)
- **OWL Full**: allow, e.g. to treat classes as individuals

\(^a\)Datatype can be seen as a unary predicate with a built-in interpretation (e.g., the `xsd:integer` datatype is interpreted as the set of all integer values)
OWL Ontologies

- **2004**: Web Ontology Language (OWL): W3C standard
- Knowledge about concepts, individuals, their properties and relationships
- Three increasingly expressive sublanguages

- **OWL Lite**: concept hierarchies, simple constraint features \((\equiv SHIF \text{ with datatypes})\)
- **OWL DL**: nominals, number restriction \((\equiv SHOIN \text{ with datatypes}^a)\)
- **OWL Full**: allow, e.g. to treat classes as individuals

\^Datatype can be seen as a unary predicate with a built-in interpretation (e.g., the `xsd:integer` datatype is interpreted as the set of all integer values)

- **2009**: OWL2: redefines OWL DL and adds profiles EL, QL, RL
- **OWL syntax** is based on RDF, common: RDF/XML, turtle syntax (in this lecture turtle is used just for demonstration)
  
  http://www.w3.org/TR/turtle/
How Do OWL2 and DLs Relate?

- **OWL2 DL** is essentially
  - *SROIQ* in disguise
  - plus extended datatype support
  - plus extralogical features such as annotations, versioning, etc.
How Do OWL2 and DLs Relate?

- **OWL2 DL** is essentially
  - \textit{SROIQ} in disguise
  - plus extended datatype support
  - plus extralogical features such as annotations, versioning, etc.

- tractable **OWL2** profiles **EL**, **QL** and **RL** correspond roughly to
description logics \(\mathcal{EL}^{++}\), **DL-Lite**, and description logic programs
How Do OWL2 and DLs Relate?

- **OWL2 DL** is essentially
  - *SROIQ* in disguise
  - plus extended datatype support
  - plus extralogical features such as annotations, versioning, etc.

- tractable **OWL2** profiles **EL**, **QL** and **RL** correspond roughly to description logics \( \mathcal{EL}^{++} \), **DL-Lite**, and description logic programs

- **OWL** and **DL** terminologies slightly differ for historical reasons

<table>
<thead>
<tr>
<th>OWL</th>
<th>DL</th>
<th>FOL</th>
</tr>
</thead>
<tbody>
<tr>
<td>class name</td>
<td>concept name</td>
<td>unary predicate</td>
</tr>
<tr>
<td>class</td>
<td>concept</td>
<td>formula with one free variable</td>
</tr>
<tr>
<td>object property name</td>
<td>role name</td>
<td>binary predicate</td>
</tr>
<tr>
<td>object property</td>
<td>role</td>
<td>formula with two free variables</td>
</tr>
<tr>
<td>ontology</td>
<td>knowledge base</td>
<td>theory</td>
</tr>
<tr>
<td>axiom</td>
<td>axiom</td>
<td>sentence</td>
</tr>
<tr>
<td>vocabulary</td>
<td>vocabulary/signature</td>
<td>signature</td>
</tr>
</tbody>
</table>
Translating DL into OWL

Next to the logic part, an OWL ontology features a preamble and a declaration part (turtle syntax):

$$[\mathcal{KB}] = \text{Pre} + \text{Dec}(\mathcal{KB}) + \sum_{\alpha \in \mathcal{KB}} [\alpha]$$

\[\text{Pre} = \begin{cases} 
\text{@prefix owl: } <\text{http://www.w3.org/2002/07/owl#}> \\
\text{@prefix rdfs: } <\text{http://www.w3.org/2000/01/rdf-schema#}> \\
\text{@prefix rdf: } <\text{http://www.w3.org/1999/02/22-rdf-syntax-ns#}> \\
\text{@prefix xsd: } <\text{http://www.w3.org/2001/XMLSchema#}> 
\end{cases}\]

\[\text{Dec}(\mathcal{KB}) = \sum_{A \in N_C(\mathcal{KB})} A \text{ rdf:type owl:Class} + \sum_{r \in N_R(\mathcal{KB})} r \text{ rdf:type owl:ObjectProperty}\]
Translating DL Axioms into OWL

• Following the Semantic Web rationale, OWL axioms are expressed using RDF, i.e. as triples. As far as possible, RDFS vocabulary is reused

\[ r_1 \circ \ldots \circ r_n \sqsubseteq r \]  \overset{\cdot}{=}  r \ \text{owl:propertyChainAxiom} (r_1, \ldots, r_n).

\[ \text{Dis}(r, r') \]  \overset{\cdot}{=}  r \ \text{owl:propertyDisjointWith} r'.

\[ C \sqsubseteq D \]  \overset{\cdot}{=}  C \ \text{owl:subClassOf} D.

\[ C(a) \]  \overset{\cdot}{=}  a \ \text{rdf:type} C.

\[ r(a, b) \]  \overset{\cdot}{=}  a \ r \ b.

\[ r^{-}(a, b) \]  \overset{\cdot}{=}  b \ r \ a.

\[ \neg r(a, b) \]  \overset{\cdot}{=}  [] \ \text{rdf:type} \ \text{owl:NegativePropertyAssertion};

\ \text{owl:assertionProperty} \ r;

\ \text{owl:sourceIndividual} \ a; \ \text{owl:targetValue} \ b.

\[ a \approx b \]  \overset{\cdot}{=}  a \ \text{owl:sameAs} b.

\[ a \nsim b \]  \overset{\cdot}{=}  a \ \text{owl:differentFrom} b.
Translating DL Axioms into OWL, cont’d.

\[ u \doteq owl:\text{topObjectProperty} \]
\[ r \doteq r \]
\[ r^- \doteq [ owl:\text{inverseOf} : r ] \]
\[ A \doteq A \]
\[ \top \doteq owl:\text{Thing} \]
\[ \bot \doteq owl:\text{Nothing} \]
\[ \{ a_1, \ldots, a_n \} \doteq [ rdf:\text{type} owl:\text{Class} ; owl:\text{oneOf} (: a_1 \ldots : a_n) ] \]
\[ \neg C \doteq [ rdf:\text{type} owl:\text{Class} ; owl:\text{complementOf} C ] \]
\[ C_1 \sqcap \ldots \sqcap C_n \doteq [ rdf:\text{type} owl:\text{Class} ; owl:\text{intersectionOf} (C_1 \ldots C_n) ] \]
\[ C_1 \sqcup \ldots \sqcup C_n \doteq [ rdf:\text{type} owl:\text{Class} ; owl:\text{unionOf} (C_1 \ldots C_n) ] \]
Translating DL Axioms into OWL ctd.

\[ \exists r.C \rightarrow [ \text{rdf:type } \text{owl:Restriction} ; \\
\text{owl:onProperty } r ; \text{owl:someValuesFrom } C ] \].

\[ \forall r.C \rightarrow [ \text{rdf:type } \text{owl:Restriction} ; \\
\text{owl:onProperty } r ; \text{owl:allValuesFrom } C ] \].

\[ \exists r.\text{Self} \rightarrow [ \text{rdf:type } \text{owl:Restriction} ; \\
\text{owl:onProperty } r ; \text{owl:hasSelf ’’true’’ xsd:boolean } ] \].

\[ \geq r.n.C \rightarrow [ \text{rdf:type } \text{owl:Restriction} ; \\
\text{owl:minQualifiedCardinality } n \text{ xsd:nonNegativeInteger} ; \\
\text{owl:onProperty } r ; \text{owl:onClass } C ] \].

\[ \leq r.n.C \rightarrow [ \text{rdf:type } \text{owl:Restriction} ; \\
\text{owl:maxQualifiedCardinality } n \text{ xsd:nonNegativeInteger} ; \\
\text{owl:onProperty } r ; \text{owl:onClass } C ] \].
Behind the Scenes: Cat Example

\[ RBox \mathcal{R} \]

\[ \text{owns} \sqsubseteq \text{caresFor} \]

"If somebody owns something, s/he cares for it."

\[ TBox \mathcal{T} \]

\[ \text{Healthy} \sqsubseteq \neg \text{Dead} \]

"Healthy beings are not dead."

\[ \text{Cat} \sqsubseteq \text{Dead} \sqcap \text{Alive} \]

"Every cat is dead or alive."

\[ \text{HappyCatOwner} \sqsubseteq \exists \text{owns}. \text{Cat} \sqcap \forall \text{caresFor}. \text{Healthy} \]

"A happy cat owner owns a cat and all beings he cares for are healthy."

\[ ABox \mathcal{A} \]

\[ \text{HappyCatOwner}(\text{schroedinger}) \]

"Schrödinger is a happy cat owner."
Behind the Scenes ctd.

:owns rdfs:SubPropertyOf :caresFor .

:Healthy rdfs:subClassOf [ owl:complementOf :Dead ].


:HappyCatOwner rdfs:subClassOf
[ owl:intersectionOf
  ( [ rdf:type owl:restriction ;
    owl:onProperty :owns ; owl:someValuesFrom :Cat ]
  [ rdf:type owl:Restriction ;
    owl:onProperty :caresFor ; owl:allValuesFrom :Healthy ] )
] .

:schroedinger rdf:type :HappyCatOwner .
Behind the Scenes ctd.

owns rdfs:SubPropertyOf :caresFor .

Healthy rdfs:subClassOf [ owl:complementOf :Dead ].

Cat rdfs:subClassOf [ owl:unionOf (:Dead :Alive) ] .

HappyCatOwner rdfs:subClassOf

[ owl:intersectionOf
  ( [ rdf:type owl:restriction ;
    owl:onProperty :owns ; owl:someValuesFrom :Cat ]
  [ rdf:type owl:Restriction;
    owl:onProperty :caresFor ; owl:allValuesFrom :Healthy ] )
]

schroedinger rdf:type :HappyCatOwner .
## Paraphrasing OWL Axioms in DL

<table>
<thead>
<tr>
<th>Axiom type</th>
<th>Turtle notation</th>
<th>DL paraphrase</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class Equivalence</td>
<td>$C \text{ owl:equivalentClass } D$</td>
<td>$C \sqsubseteq D, D \sqsubseteq C$</td>
</tr>
<tr>
<td>Class Disjointness</td>
<td>$C \text{ owl:disjointWith } D$</td>
<td>$C \sqcap D \sqsubseteq \bot$</td>
</tr>
<tr>
<td>Disjoint Classes</td>
<td>$\sqcap \text{ rdf:type owl:AllDisjointClasses ; }$</td>
<td>$C_i \sqcap C_j \sqsubseteq \bot$, for all $1 \leq i &lt; j \leq n$</td>
</tr>
<tr>
<td></td>
<td>$\text{owl:members } (C_1 \ldots C_n)$.</td>
<td></td>
</tr>
<tr>
<td>Disjoint Union</td>
<td>$C \text{ owl:disjointUnionOf } (C_1 \ldots C_n)$.</td>
<td>$\bigcup_{i&lt;j} C_i \sqsubseteq C$, $C_i \sqcap C_j \sqsubseteq \bot$, for all $1 \leq i &lt; j \leq n$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$r \sqsubseteq s$, $s \sqsubseteq r$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Dis$(r_i, r_j)$, for all $1 \leq i &lt; j \leq n$</td>
</tr>
<tr>
<td>Property Equivalence</td>
<td>$r \text{ owl:equivalentProperty } s$.</td>
<td>$\text{Inv}(r) \sqsubseteq s$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\exists r. \top \sqsubseteq C$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\top \sqsubseteq \forall r.C$</td>
</tr>
<tr>
<td>Disjoint Properties</td>
<td>$\sqcap \text{ rdf:type owl:AllDisjointProperties ; }$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$\text{owl:members } (r_1 \ldots r_n)$.</td>
<td></td>
</tr>
<tr>
<td>Inverse Properties</td>
<td>$r \text{ owl:inverseOf } s$.</td>
<td></td>
</tr>
<tr>
<td>Property Domain</td>
<td>$r \text{ rdfs:domain } C$.</td>
<td></td>
</tr>
<tr>
<td>Property Range</td>
<td>$r \text{ rdfs:range } C$.</td>
<td></td>
</tr>
</tbody>
</table>
**Paraphrasing OWL Axioms in DL ctd.**

<table>
<thead>
<tr>
<th>Axiom type</th>
<th>Turtle notation</th>
<th>DL paraphrase</th>
</tr>
</thead>
<tbody>
<tr>
<td>Functional Property</td>
<td>( r \text{ rdf:type owl:FunctionalProperty} . )</td>
<td>( \top \sqsubseteq 1 r . \top )</td>
</tr>
<tr>
<td>Inverse Functional Property</td>
<td>( r \text{ rdf:type } )</td>
<td>( \top \sqsubseteq 1 \text{Inv}(r) . \top )</td>
</tr>
<tr>
<td>Property</td>
<td>( \text{owl:InverseFunctionalProperty} . )</td>
<td>( \top \sqsubseteq \exists r . \text{Self} )</td>
</tr>
<tr>
<td>Reflexive Property</td>
<td>( r \text{ rdf:type owl:ReflexiveProperty} . )</td>
<td>( \top \sqsubseteq \exists r . \text{Self} \sqsubseteq \bot )</td>
</tr>
<tr>
<td>Irreflexive Property</td>
<td>( r \text{ rdf:type owl:IrreflexiveProperty} . )</td>
<td>( \text{Inv}(r) \sqsubseteq r )</td>
</tr>
<tr>
<td>Symmetric Property</td>
<td>( r \text{ rdf:type owl:SymmetricProperty} . )</td>
<td>( \text{Dis(Inv}(r), r) )</td>
</tr>
<tr>
<td>Asymmetric Property</td>
<td>( r \text{ rdf:type owl:AsymmetricProperty} . )</td>
<td>( r \circ r \sqsubseteq r )</td>
</tr>
<tr>
<td>Transitive Property</td>
<td>( r \text{ rdf:type owl:TransitiveProperty} . )</td>
<td>( a_i \not\approx a_j ), for all ( 1 \leq i &lt; j \leq n )</td>
</tr>
<tr>
<td>Different Individuals</td>
<td>( [ \text{ rdf:type owl:AllDifferent; owl:members(a}_1 \ldots a_n) . )</td>
<td>( \top )</td>
</tr>
</tbody>
</table>
OWL Profiles

- OWL2 is highly intractable in general! (standard reasoning is 2NEXPTIME-complete).
- **Design principle for profiles:** Identify maximal OWL sublanguages that are still implementable in PTime.
- **Main source of intractability:** non-determinism (reasoning requires guessing/backtracking)
  - owl:unionOf, or owl:complementOf and owl:intersectionOf
  - Max. cardinality restrictions
  - Combining existentials (owl:someValuesFrom) and universals (owl:allValuesFrom) in superclasses
  - Non-unary finite class expressions (owl:oneOf) or datatype expressions

→ features that are not allowed in any OWL profile.
Many further features can lead to non-determinism - care needed!
**OWL2 EL**

- **OWL profile based on description logic $\mathcal{EL}^{++}$**
- **Intuition**: focus on terminological expressivity used for light-weight ontologies
- Allow `owl:someValuesFrom` (existential) but not `owl:allValuesFrom` (universal)
- Property domains, class/property hierarchies, class intersections, disjoint classes/properties, property chains, `owl:hasSelf`, `owl:hasValue`, and keys fully supported
- No inverse or symmetric properties
- `rdfs:range` allowed but with some restrictions
- No `owl:unionOf` or `owl:complementOf`
- Various restrictions on available datatypes
OWL2 QL

- **OWL profile that can be used to query data-rich applications**
  - often used for OBDA (ontology based data access)

- Intuition: use OWL concepts as light-weight queries, allow query answering using rewriting in SQL on top of relational DBs

- Different restrictions on subclasses and superclasses of rdfs:SubclassOf
  - subclasses can only be class names or owl:someValuesFrom (existential) with unrestricted (owl:Thing) filler
  - superclasses can be class names, owl:someValuesFrom or owl:intersectionOf with superclass filler (recursive), or owl:complementOf with subclass filler
OWL2 QL ctd.

- Property hierarchies, disjointness, inverses, (a)symmetry supported, restrictions on range and domain
- Disjoint or equivalence of classes only for subclass-type expressions
- No owl:unionOf, owl:allValuesFrom, owl:hasSelf, owl:hasKey, owl:hasValue, owl:oneOf, owl:sameAs, owl:propertyChainAxiom, owl:TransitiveProperty, cardinalities, functional properties
- Some restrictions on available datatypes
OWL2 RL

- **OWL profile** that resembles an OWL-based rule language

- **Intuition**: subclass axioms in OWL RL can be understood as rule-like implications with head (superclass) and body (subclass)

- **Different restrictions** on subclasses and superclasses of \texttt{rdfs:SubclassOf}
  - subclasses can only be class names, \texttt{owl:oneOf}, \texttt{owl:hasValue}, \texttt{owl:intersectionOf}, \texttt{owl:unionOf}, \texttt{owl:someValuesFrom} if applied only to subclass-type expressions
  - superclasses can be class names, \texttt{owl:allValuesFrom} or \texttt{owl:hasValue}; also max. cardinalities of 0 or 1 are allowed, all with superclass-type filler expressions only
OWL2 RL cdt.

- Property domains and ranges only for subclass-type expressions
- Full support of property hierarchies, disjointness, inverses, (a)symmetry, transitivity, chains, (inverse)functionality, irreflexivity
- Disjoint classes and classes in keys need subclass-type expressions
- Equivalence only for expressions that are sub- and superclass-type, no restrictions on `owl:sameAs`
- Some restrictions on available datatypes

**Important feature:** as in relational databases, only “named” individuals matter:

\[ Person \sqsubseteq \exists father \quad Person(joe). \]

intuitively, father of Joe is “unnamed”; axiom is not allowed in RL.
Do We Really Need So Many OWLs?

Three new OWL profiles with somewhat complex descriptions...
Why not just one?

- The union of any two of the profiles is no longer light-weight!
  - each of $\text{EL} + \text{RL}$, $\text{QL} + \text{EL}$, $\text{RL} + \text{EL}$ is ExpTime-hard

- Restricting to fewer profiles: give up potentially useful feature combinations

- **Rationale:**
  - profiles are "maximal" (well, not quite) well-behaved fragments of OWL 2
  - pick suitable feature set for applications

- In particular, nobody is forced to implement *all* of a profile
OWL in Practice: Tools

- Most common editor: Protégé 4
- Other tools: TopBraid Composer, NeOn toolkit, etc.
- Special purpose apps, esp. for light-weight ontologies (e.g. FOAF editors)
- Reasoners
  - OWL DL: Pellet, HermiT, FaCT++, RacerPro
  - OWL2 EL: CEL, SHER, snorocket, ELK
  - OWL2 RL: OWLIM, Jena, Oracle Prime (part of O 11g)
  - OWL2 QL: Owlgres, QuOnto, Quill
- Many tools use the OWL API library (Java)
- Note: many other Semantic Web tools are found online

http://www.w3.org/2001/sw/wiki/OWL/Implementations
http://semanticweb.org/wiki/Tools.html
Summary

1. Modeling
   - Frequent modeling features
   - Number restrictions
   - Nominal concepts
   - Self restrictions

2. Description Logics and OWL
   - Relation between DLs and OWL
   - Translating DLs into OWL
   - OWL profiles
   - Tools
Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter Patel-Schneider, editors.

*The Description Logic Handbook: Theory, Implementation and Applications.*

Pascal Hitzler, Markus Krötzsch, and Sebastian Rudolph.

*Foundations of Semantic Web Technologies.*

Sebastian Rudolph.

Foundations of description logics.