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

- individuals \textit{angelina} and \textit{brad} are in the relation of being married:
  \[ \text{married(angelina, brad)} \]
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:
  \[ \forall x. \text{Actor}(x) \rightarrow \text{Artist}(x) \]
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:
  \[ \text{Actor} \sqsubseteq \text{Artist} \]

\[ \forall x. \text{Actor}(x) \rightarrow \text{Artist}(x) \]
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

• 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:
  \[ \text{Actor} \sqcap \text{USGovernor} \sqsubseteq \text{Bodybuilder} \sqcup \neg \text{Austrian} \]
  \[ \forall x. (\text{Actor}(x) \land \text{USGovernor}(x)) \rightarrow (\text{BodyBuilder}(x) \lor \neg \text{Austrian}(x)) \]
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:
  \[
  \exists \text{hasChild}. \top \sqsubseteq \forall \text{hasChild} \rightarrow \exists \text{Grandparent}.
  \]
  \[
  \forall x (\exists y (\text{hasChild}(x, y)) \rightarrow \\
  \forall z (\text{hasChild}(z, x) \rightarrow \text{Grandparent}(x)))
  \]
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:
  \[
  \exists \text{hasChild}.\top \sqsubseteq \forall \text{hasChild}^-\text{.Grandparent}
  \]
  \[
  \forall x (\exists y (\text{hasChild}(x, y)) \rightarrow \\
  \forall z (\text{hasChild}(z, x) \rightarrow \text{Grandparent}(x)))
  \]
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:
  \[
  \exists \text{knows}. \text{Actor} \sqsubseteq \forall \text{hasfriend}. \text{Envious}
  \]
  \[
  \forall x (\exists y (\text{knows}(x, y) \land \text{Actor}(y)) \rightarrow \\
  \quad \forall z (\text{hasfriend}(x, z) \rightarrow \text{Envious}(z)))
  \]

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

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

- being married to Brad is a property only applying to Angelina:
Modeling with DLs: Motivating Examples, cont’d.

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

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

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

  \[ \forall x \forall y \forall z (\text{hasChild}(y, x) \land \text{hasChild}(y, z) \rightarrow \text{hasSibling}(x, z)) \]
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:
  \[ \text{hasChild}^{-} \circ \text{hasChild} \sqsubseteq \text{hasSibling} \]
  \[ \forall x \forall y \forall z (\text{hasChild}(y, x) \wedge \text{hasChild}(y, z) \rightarrow \text{hasSibling}(x, z)) \]
Frequent Modeling Features

- domain
  \[ \exists \text{authorOf}. \top \sqsubseteq \text{Person} \]

- range
  \[ \top \sqsubseteq \forall \text{authorOf}. \text{Publication} \]

- concept disjointness
  \[ \text{Male} \sqcap \text{Female} \sqsubseteq \bot \]
  \[ \text{Male} \sqsubseteq \neg \text{Female} \]

- role symmetry
  \[ \text{marriedWith} \sqsubseteq \text{marriedWith}^\neg \]

- role transitivity
  \[ \text{partOf} \circ \text{partOf} \sqsubseteq \text{partOf} \]
Number Restrictions

- allow for defining that a role is functional

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

- ...or inverse functional

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

- allow for enforcing an infinite domain

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

- 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

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

- even allow to restrict the size of the domain

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

• allows to define a role as reflexive

\[ \top \sqsubseteq \exists \text{knows}.\text{Self} \]

• allows to define a role as irreflexive

\[ \exists \text{betterThan}.\text{Self} \sqsubseteq \bot \]

• together with number restrictions, we can even axiomatize equality

\[ \top \sqsubseteq \exists \text{equals}.\text{Self} \quad \top \sqsubseteq \leq 1 \text{equals}.\top \]
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
Uniqueness constraints might not eliminate models either

- define that $hasSSN$ is inverse functional, and add some data:

- for

$$KB = \left\{ \begin{array}{l} \top \sqsubseteq \leq 1 \ hasSSN^{-}\top, \\ hasSSN(Joe, 4711), \\ hasSSN(Jeff, 4711) \end{array} \right\}$$

we can conclude that $KB \models Joe \approx Jeff$

- 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
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 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 \((\rightleftharpoons SHIF \text{ with datatypes})\)
  - **OWL DL**: nominals, number restriction \((\rightleftharpoons SHOIN \text{ with datatypes}^a)\)
  - **OWL Full**: allow, e.g. to treat classes as individuals

\^aDatatype 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

\(^a\)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)

- **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):

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

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

\[
\text{Dec}(KB) = \sum_{A \in NC(KB)} A \text{ rdf:type owl:Class} \\
+ \sum_{r \in NR(KB)} r \text{ rdf:type owl:ObjectProperty}
\]
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

\[
\begin{align*}
    r_1 \circ \ldots \circ r_n \sqsubseteq r & \quad \equiv \quad r \ \text{owl:propertyChainAxiom} \ (r_1, \ldots, r_n). \\
    \text{Dis}(r, r') & \quad \equiv \quad r \ \text{owl:propertyDisjointWith} \ r'. \\
    C \sqsubseteq D & \quad \equiv \quad C \ \text{owl:subClassOf} \ D. \\
    C(a) & \quad \equiv \quad a \ \text{rdf:type} \ C. \\
    r(a, b) & \quad \equiv \quad a \ r \ b. \\
    r^-(a, b) & \quad \equiv \quad b \ r \ a. \\
    \neg r(a, b) & \quad \equiv \quad [\text{rdf:type} \ \text{owl:NegativePropertyAssertion}; \\
    & \quad \quad \text{owl:assertionProperty} \ r; \\
    & \quad \quad \text{owl:sourceIndividual} \ a; \ \text{owl:targetValue} \ b. \\
    a \approx b & \quad \equiv \quad a \ \text{owl:sameAs} \ b. \\
    a \not\approx b & \quad \equiv \quad a \ \text{owl:differentFrom} \ b.
\end{align*}
\]
Translating DL Axioms into OWL, cont’d.

\[ u \doteq \text{owl:topObjectProperty} . \]

\[ r \doteq r . \]

\[ r^- \doteq [ \text{owl:inverseOf} : r ] . \]

\[ A \doteq A . \]

\[ \top \doteq \text{owl:Thing} . \]

\[ \bot \doteq \text{owl:Nothing} . \]

\[ \{ a_1, \ldots, a_n \} \doteq [ \text{rdf:type owl:Class} ; \text{owl:oneOf} (: a_1 \ldots : a_n) ] . \]

\[ \neg C \doteq [ \text{rdf:type owl:Class} ; \text{owl:complementOf} C ] . \]

\[ C_1 \sqcap \ldots \sqcap C_n \doteq [ \text{rdf:type owl:Class} ; \text{owl:intersectionOf} (C_1 \ldots C_n) ] . \]

\[ C_1 \sqcup \ldots \sqcup C_n \doteq [ \text{rdf:type owl:Class} ; \text{owl:unionOf} (C_1 \ldots C_n) ] . \]
Translating DL Axioms into OWL ctd.

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

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

\[ \exists r.\text{Self} \triangleright \left[ \text{rdf:type } \text{owl:Restriction}; \right. \\
\left. \text{owl:onProperty } r; \text{owl:hasSelf } 'true' \text{ xsd:boolean } \right].\]

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

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

### RBox \( \mathcal{R} \)

<table>
<thead>
<tr>
<th>Predicate</th>
<th>Subsumed By Predicates</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{owns} )</td>
<td>( \sqsubseteq \text{caresFor} )</td>
</tr>
</tbody>
</table>

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

### TBox \( \mathcal{T} \)

<table>
<thead>
<tr>
<th>Concept</th>
<th>Subsumed By Concepts</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{Healthy} )</td>
<td>( \sqsubseteq \neg \text{Dead} )</td>
</tr>
</tbody>
</table>

"Healthy beings are not dead."

<table>
<thead>
<tr>
<th>Concept</th>
<th>Subsumed By Concepts</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{Cat} )</td>
<td>( \sqsubseteq \text{Dead} \sqcup \text{Alive} )</td>
</tr>
</tbody>
</table>

"Every cat is dead or alive."

<table>
<thead>
<tr>
<th>Concept</th>
<th>Subsumed By Concepts</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{HappyCatOwner} )</td>
<td>( \sqsubseteq \exists \text{owns} \cdot \text{Cat} \sqcap \forall \text{caresFor} \cdot \text{Healthy} )</td>
</tr>
</tbody>
</table>

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

### ABox \( \mathcal{A} \)

<table>
<thead>
<tr>
<th>Concept</th>
<th>Members</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{HappyCatOwner} ) (\text{schroedinger})</td>
<td></td>
</tr>
</tbody>
</table>

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

```rdfs
: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 .
```
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 rdfs:SubclassOf
  - subclasses can only be class names, owl:oneOf, owl:hasValue, owl:intersectionOf, owl:unionOf, owl:someValuesFrom if applied only to subclass-type expressions
  - superclasses can be class names, owl:allValuesFrom or 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:

\[ \text{Person} \sqsubseteq \exists \text{father } \text{Person}(joe). \]

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 $\text{ExpTime-hard}$

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

- **Rationale:**
  - profiles are ”maximal” (well, not quite) well-behaved fragments of $\text{OWL 2}$
  - 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
Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter Patel-Schneider, editors.


Pascal Hitzler, Markus Krötzsch, and Sebastian Rudolph.

Foundations of Semantic Web Technologies.

Sebastian Rudolph.

Foundations of description logics.