INF3580/4580 – Semantic Technologies – Spring 2017

Lecture 10: OWL, the Web Ontology Language

Leif Harald Karlsen

20th March 2017

Department of Informatics

University of Oslo

Reminders

- Oblig. 5:
  First deadline tomorrow (21.03).

Reminders

- Oblig. 5:
  First deadline tomorrow (21.03).
- Oblig. 6: Will be published 03.04.

Today's Plan

1. Reminder: RDFS
2. Description Logics
3. Introduction to OWL

Outline

The RDFS vocabulary

- RDFS adds the concept of "classes" which are like *types* or *sets* of resources.

The RDFS vocabulary

- RDFS adds the concept of “classes” which are like *types* or *sets* of resources.
- A predefined vocabulary allows statements about classes.

The RDFS vocabulary

- RDFS adds the concept of "classes" which are like *types* or *sets* of resources.
- A predefined vocabulary allows statements about classes.
- Defined resources:

The RDFS vocabulary

- RDFS adds the concept of "classes" which are like *types* or *sets* of resources.
- A predefined vocabulary allows statements about classes.
- Defined resources:
  - `rdfs:Resource`: The class of resources, everything,

The RDFS vocabulary

- RDFS adds the concept of "classes" which are like *types* or *sets* of resources.
- A predefined vocabulary allows statements about classes.
- Defined resources:
    - `rdfs:Resource`: The class of resources, everything,
    - `rdfs:Class`: The class of classes,

The RDFS vocabulary

- RDFS adds the concept of "classes" which are like *types* or *sets* of resources.
- A predefined vocabulary allows statements about classes.
- Defined resources:
    - `rdfs:Resource`: The class of resources, everything,
    - `rdfs:Class`: The class of classes,
    - `rdf:Property`: The class of properties (from `rdf`).

The RDFS vocabulary

- RDFS adds the concept of “classes” which are like *types* or *sets* of resources.
- A predefined vocabulary allows statements about classes.
- Defined resources:
  - `rdfs:Resource`: The class of resources, everything,
  - `rdfs:Class`: The class of classes,
  - `rdf:Property`: The class of properties (from `rdf`).
- Defined properties:

The RDFS vocabulary

- RDFS adds the concept of "classes" which are like *types* or *sets* of resources.
- A predefined vocabulary allows statements about classes.
- Defined resources:
  - `rdfs:Resource`: The class of resources, everything,
  - `rdfs:Class`: The class of classes,
  - `rdf:Property`: The class of properties (from `rdf`).
- Defined properties:
  - `rdf:type`: relates resources to classes they are members of.

The RDFS vocabulary

- RDFS adds the concept of “classes” which are like *types* or *sets* of resources.
- A predefined vocabulary allows statements about classes.
- Defined resources:
  - `rdfs:Resource`: The class of resources, everything,
  - `rdfs:Class`: The class of classes,
  - `rdf:Property`: The class of properties (from `rdf`).
- Defined properties:
  - `rdf:type`: relates resources to classes they are members of.
  - `rdfs:domain`: The domain of a relation.

The RDFS vocabulary

- RDFS adds the concept of “classes” which are like *types* or *sets* of resources.
- A predefined vocabulary allows statements about classes.
- Defined resources:
  - `rdfs:Resource`: The class of resources, everything,
  - `rdfs:Class`: The class of classes,
  - `rdf:Property`: The class of properties (from `rdf`).
- Defined properties:
  - `rdf:type`: relates resources to classes they are members of.
  - `rdfs:domain`: The domain of a relation.
  - `rdfs:range`: The range of a relation.

The RDFS vocabulary

- RDFS adds the concept of "classes" which are like *types* or *sets* of resources.
- A predefined vocabulary allows statements about classes.
- Defined resources:
  - `rdfs:Resource`: The class of resources, everything,
  - `rdfs:Class`: The class of classes,
  - `rdf:Property`: The class of properties (from `rdf`).
- Defined properties:
  - `rdf:type`: relates resources to classes they are members of.
  - `rdfs:domain`: The domain of a relation.
  - `rdfs:range`: The range of a relation.
  - `rdfs:subClassOf`: Concept inclusion.

The RDFS vocabulary

- RDFS adds the concept of "classes" which are like *types* or *sets* of resources.
- A predefined vocabulary allows statements about classes.
- Defined resources:
  - `rdfs:Resource`: The class of resources, everything,
  - `rdfs:Class`: The class of classes,
  - `rdf:Property`: The class of properties (from `rdf`).
- Defined properties:
  - `rdf:type`: relates resources to classes they are members of.
  - `rdfs:domain`: The domain of a relation.
  - `rdfs:range`: The range of a relation.
  - `rdfs:subClassOf`: Concept inclusion.
  - `rdfs:subPropertyOf`: Property inclusion.

Clear semantics

- RDFS has formal semantics.

Clear semantics

- RDFS has formal semantics.
- Entailment is a *mathematically* defined relationship between RDF(S) graphs. E.g.,

Clear semantics

- RDFS has formal semantics.
- Entailment is a *mathematically* defined relationship between RDF(S) graphs. E.g.,
    - answers to SPARQL queries are well-defined, and

Clear semantics

- RDFS has formal semantics.
- Entailment is a *mathematically* defined relationship between RDF(S) graphs. E.g.,
  - answers to SPARQL queries are well-defined, and
  - the interpretation of blank nodes is clear.

Clear semantics

- RDFS has formal semantics.
- Entailment is a *mathematically* defined relationship between RDF(S) graphs. E.g.,
  - answers to SPARQL queries are well-defined, and
  - the interpretation of blank nodes is clear.
- The semantics allows for rules to reason about classes and properties and membership.

Clear semantics

- RDFS has formal semantics.
- Entailment is a *mathematically* defined relationship between RDF(S) graphs. E.g.,
  - answers to SPARQL queries are well-defined, and
  - the interpretation of blank nodes is clear.
- The semantics allows for rules to reason about classes and properties and membership.
- Using RDFS entailment rules we can infer:

Clear semantics

- RDFS has formal semantics.
- Entailment is a *mathematically* defined relationship between RDF(S) graphs. E.g.,
  - answers to SPARQL queries are well-defined, and
  - the interpretation of blank nodes is clear.
- The semantics allows for rules to reason about classes and properties and membership.
- Using RDFS entailment rules we can infer:
  - type propagation

Clear semantics

- RDFS has formal semantics.
- Entailment is a *mathematically* defined relationship between RDF(S) graphs. E.g.,
  - answers to SPARQL queries are well-defined, and
  - the interpretation of blank nodes is clear.
- The semantics allows for rules to reason about classes and properties and membership.
- Using RDFS entailment rules we can infer:
  - type propagation
  - property inheritance, and

Clear semantics

- RDFS has formal semantics.
- Entailment is a *mathematically* defined relationship between RDF(S) graphs. E.g.,
  - answers to SPARQL queries are well-defined, and
  - the interpretation of blank nodes is clear.
- The semantics allows for rules to reason about classes and properties and membership.
- Using RDFS entailment rules we can infer:
  - type propagation
  - property inheritance, and
  - domain and range reasoning.

Yet, it's inexpressive

- RDFS does not allow for complex definitions, other than multiple inheritance.

Yet, it's inexpressive

- RDFS does not allow for complex definitions, other than multiple inheritance.
- We cannot express negation in RDFS.

Yet, it's inexpressive

- RDFS does not allow for complex definitions, other than multiple inheritance.
- We cannot express negation in RDFS.
- Hence, because of OWA, all RDFS graphs are satisfiable.

Modelling patterns

Common modelling patterns cannot be expressed properly in RDFS:

- ✗ Every person has a mother.

Modelling patterns

Common modelling patterns cannot be expressed properly in RDFS:

- ✗ Every person has a mother.
- ✗ Penguins eat only fish. Horses eat only chocolate.

Modelling patterns

Common modelling patterns cannot be expressed properly in RDFS:

- ✗ Every person has a mother.
- ✗ Penguins eat only fish. Horses eat only chocolate.
- ✗ Every nuclear family has two parents, at least two children and a dog.

Modelling patterns

Common modelling patterns cannot be expressed properly in RDFS:

- ✗ Every person has a mother.
- ✗ Penguins eat only fish. Horses eat only chocolate.
- ✗ Every nuclear family has two parents, at least two children and a dog.
- ✗ No smoker is a non-smoker (and vice versa).

Modelling patterns

Common modelling patterns cannot be expressed properly in RDFS:

- ✗ Every person has a mother.
- ✗ Penguins eat only fish. Horses eat only chocolate.
- ✗ Every nuclear family has two parents, at least two children and a dog.
- ✗ No smoker is a non-smoker (and vice versa).
- ✗ Everybody loves Mary.

Modelling patterns

Common modelling patterns cannot be expressed properly in RDFS:

- ✗ Every person has a mother.
- ✗ Penguins eat only fish. Horses eat only chocolate.
- ✗ Every nuclear family has two parents, at least two children and a dog.
- ✗ No smoker is a non-smoker (and vice versa).
- ✗ Everybody loves Mary.
- ✗ Adam is not Eve (and vice versa).

Modelling patterns

Common modelling patterns cannot be expressed properly in RDFS:

- ✗ Every person has a mother.
- ✗ Penguins eat only fish. Horses eat only chocolate.
- ✗ Every nuclear family has two parents, at least two children and a dog.
- ✗ No smoker is a non-smoker (and vice versa).
- ✗ Everybody loves Mary.
- ✗ Adam is not Eve (and vice versa).
- ✗ Everything is black or white.

Modelling patterns

Common modelling patterns cannot be expressed properly in RDFS:

- ✗ Every person has a mother.
- ✗ Penguins eat only fish. Horses eat only chocolate.
- ✗ Every nuclear family has two parents, at least two children and a dog.
- ✗ No smoker is a non-smoker (and vice versa).
- ✗ Everybody loves Mary.
- ✗ Adam is not Eve (and vice versa).
- ✗ Everything is black or white.
- ✗ There is no such thing as a free lunch.

Modelling patterns

Common modelling patterns cannot be expressed properly in RDFS:

- ✗ Every person has a mother.
- ✗ Penguins eat only fish. Horses eat only chocolate.
- ✗ Every nuclear family has two parents, at least two children and a dog.
- ✗ No smoker is a non-smoker (and vice versa).
- ✗ Everybody loves Mary.
- ✗ Adam is not Eve (and vice versa).
- ✗ Everything is black or white.
- ✗ There is no such thing as a free lunch.
- ✗ Brothers of fathers are uncles.

Modelling patterns

Common modelling patterns cannot be expressed properly in RDFS:

- ✗ Every person has a mother.
- ✗ Penguins eat only fish. Horses eat only chocolate.
- ✗ Every nuclear family has two parents, at least two children and a dog.
- ✗ No smoker is a non-smoker (and vice versa).
- ✗ Everybody loves Mary.
- ✗ Adam is not Eve (and vice versa).
- ✗ Everything is black or white.
- ✗ There is no such thing as a free lunch.
- ✗ Brothers of fathers are uncles.
- ✗ My friend's friends are also my friends.

Modelling patterns

Common modelling patterns cannot be expressed properly in RDFS:

- ✗ Every person has a mother.
- ✗ Penguins eat only fish. Horses eat only chocolate.
- ✗ Every nuclear family has two parents, at least two children and a dog.
- ✗ No smoker is a non-smoker (and vice versa).
- ✗ Everybody loves Mary.
- ✗ Adam is not Eve (and vice versa).
- ✗ Everything is black or white.
- ✗ There is no such thing as a free lunch.
- ✗ Brothers of fathers are uncles.
- ✗ My friend's friends are also my friends.
- ✗ If Homer is married to Marge, then Marge is married to Homer.

Modelling patterns

Common modelling patterns cannot be expressed properly in RDFS:

- ✗ Every person has a mother.
- ✗ Penguins eat only fish. Horses eat only chocolate.
- ✗ Every nuclear family has two parents, at least two children and a dog.
- ✗ No smoker is a non-smoker (and vice versa).
- ✗ Everybody loves Mary.
- ✗ Adam is not Eve (and vice versa).
- ✗ Everything is black or white.
- ✗ There is no such thing as a free lunch.
- ✗ Brothers of fathers are uncles.
- ✗ My friend's friends are also my friends.
- ✗ If Homer is married to Marge, then Marge is married to Homer.
- ✗ If Homer is a parent of Bart, then Bart is a child of Homer.

And it's complicated

In the standardised RDFS semantics (not our simplified version):

- No clear ontology/data boundary

And it's complicated

In the standardised RDFS semantics (not our simplified version):

- No clear ontology/data boundary
  - No restrictions on the use of the built-ins.

And it's complicated

In the standardised RDFS semantics (not our simplified version):

- No clear ontology/data boundary
  - No restrictions on the use of the built-ins.
  - Can have relations between classes and relations:

```
:myCar      rdf:type      citroen:TwoCV .
rdf:type    rdfs:domain   rdfs:Resource .
```

And it's complicated

In the standardised RDFS semantics (not our simplified version):

- No clear ontology/data boundary
  - No restrictions on the use of the built-ins.
  - Can have relations between classes and relations:

```
:myCar      rdf:type      citroen:TwoCV .
rdf:type    rdfs:domain   rdfs:Resource .
```

  - Remember: in RDF, properties are resources,

And it's complicated

In the standardised RDFS semantics (not our simplified version):

- No clear ontology/data boundary
  - No restrictions on the use of the built-ins.
  - Can have relations between classes and relations:

```
:myCar      rdf:type      citroen:TwoCV .
rdf:type    rdfs:domain   rdfs:Resource .
```

  - Remember: in RDF, properties are resources,
  - so they can be subject or object of triples.

And it's complicated

In the standardised RDFS semantics (not our simplified version):

- No clear ontology/data boundary
  - No restrictions on the use of the built-ins.
  - Can have relations between classes and relations:

```
:myCar      rdf:type      citroen:TwoCV .
rdf:type    rdfs:domain   rdfs:Resource .
```

  - Remember: in RDF, properties are resources,
  - so they can be subject or object of triples.
  - Well, in RDFS, classes are resources,

And it's complicated

In the standardised RDFS semantics (not our simplified version):

- No clear ontology/data boundary
  - No restrictions on the use of the built-ins.
  - Can have relations between classes and relations:

```
:myCar      rdf:type      citroen:TwoCV .
rdf:type    rdfs:domain   rdfs:Resource .
```

  - Remember: in RDF, properties are resources,
  - so they can be subject or object of triples.
  - Well, in RDFS, classes are resources,
  - so they can also be subject or object of triples.

And it's complicated

In the standardised RDFS semantics (not our simplified version):

- No clear ontology/data boundary
  - No restrictions on the use of the built-ins.
  - Can have relations between classes and relations:

```
:myCar      rdf:type       citroen:TwoCV  .
rdf:type    rdfs:domain    rdfs:Resource  .
```

  - Remember: in RDF, properties are resources,
  - so they can be subject or object of triples.
  - Well, in RDFS, classes are resources,
  - so they can also be subject or object of triples.
- The RDFS entailment rules are incomplete.

And it's complicated

In the standardised RDFS semantics (not our simplified version):

- No clear ontology/data boundary
  - No restrictions on the use of the built-ins.
  - Can have relations between classes and relations:

```
:myCar      rdf:type       citroen:TwoCV .
rdf:type    rdfs:domain    rdfs:Resource .
```

  - Remember: in RDF, properties are resources,
  - so they can be subject or object of triples.
  - Well, in RDFS, classes are resources,
  - so they can also be subject or object of triples.
- The RDFS entailment rules are incomplete.
  - Can't derive all statements that are semantically valid.

Outline

Make it simple!

- Keep classes, properties, individuals and relationships apart.

Make it simple!

- Keep classes, properties, individuals and relationships apart.
- "Data level" with individuals and relationships between them.

Make it simple!

- Keep classes, properties, individuals and relationships apart.
- “Data level” with individuals and relationships between them.
- “Ontology level” with properties and classes.

Make it simple!

- Keep classes, properties, individuals and relationships apart.
- "Data level" with individuals and relationships between them.
- "Ontology level" with properties and classes.
- Use a fixed vocabulary of built-ins for relations between classes and properties, and their members—and nothing else.

Make it simple!

- Keep classes, properties, individuals and relationships apart.
- "Data level" with individuals and relationships between them.
- "Ontology level" with properties and classes.
- Use a fixed vocabulary of built-ins for relations between classes and properties, and their members—and nothing else.
- Interpret

Make it simple!

- Keep classes, properties, individuals and relationships apart.
- "Data level" with individuals and relationships between them.
- "Ontology level" with properties and classes.
- Use a fixed vocabulary of built-ins for relations between classes and properties, and their members—and nothing else.
- Interpret
  - classes as sets of individuals, and

Make it simple!

- Keep classes, properties, individuals and relationships apart.
- "Data level" with individuals and relationships between them.
- "Ontology level" with properties and classes.
- Use a fixed vocabulary of built-ins for relations between classes and properties, and their members—and nothing else.
- Interpret
  - classes as sets of individuals, and
  - properties as relations between individuals, i.e., sets of pairs

Make it simple!

- Keep classes, properties, individuals and relationships apart.
- "Data level" with individuals and relationships between them.
- "Ontology level" with properties and classes.
- Use a fixed vocabulary of built-ins for relations between classes and properties, and their members—and nothing else.
- Interpret
  - classes as sets of individuals, and
  - properties as relations between individuals, i.e., sets of pairs
  - —which is what we do in our simplified semantics.

Make it simple!

- Keep classes, properties, individuals and relationships apart.
- "Data level" with individuals and relationships between them.
- "Ontology level" with properties and classes.
- Use a fixed vocabulary of built-ins for relations between classes and properties, and their members—and nothing else.
- Interpret
  - classes as sets of individuals, and
  - properties as relations between individuals, i.e., sets of pairs
  - —which is what we do in our simplified semantics.
- A setting well-studied as *Description Logics*.

The $\mathcal{ALC}$ Description Logic

Vocabulary

Fix a set of *atomic concepts* $\{A_1, A_2, \ldots\}$, *roles* $\{R_1, R_2, \ldots\}$ and individuals $\{a_1, a_2, \ldots\}$.

The $\mathcal{ALC}$ Description Logic

Vocabulary

Fix a set of *atomic concepts* $\{A_1, A_2, \ldots\}$, *roles* $\{R_1, R_2, \ldots\}$ and individuals $\{a_1, a_2, \ldots\}$.

$\mathcal{ALC}$ concept descriptions

$$
\begin{array}{rlll}
C, D \rightarrow & A_i & \mid & \text{(atomic concept)} \\
& \top & \mid & \text{(universal concept)} \\
& \bot & \mid & \text{(bottom concept)}
\end{array}
$$

The $\mathcal{ALC}$ Description Logic

Vocabulary

Fix a set of *atomic concepts* $\{A_1, A_2, \ldots\}$, *roles* $\{R_1, R_2, \ldots\}$ and individuals $\{a_1, a_2, \ldots\}$.

$\mathcal{ALC}$ concept descriptions

$$
\begin{array}{rlcl}
C, D \rightarrow & A_i & | & \text{(atomic concept)} \\
& \top & | & \text{(universal concept)} \\
& \bot & | & \text{(bottom concept)} \\
& \neg C & | & \text{(negation)} \\
& C \sqcap D & | & \text{(intersection)} \\
& C \sqcup D & | & \text{(union)}
\end{array}
$$

The $\mathcal{ALC}$ Description Logic

Vocabulary

Fix a set of *atomic concepts* $\{A_1, A_2, \ldots\}$, *roles* $\{R_1, R_2, \ldots\}$ and individuals $\{a_1, a_2, \ldots\}$.

$\mathcal{ALC}$ concept descriptions

$$
\begin{array}{rlcl}
C, D \rightarrow & A_i & | & \text{(atomic concept)} \\
& \top & | & \text{(universal concept)} \\
& \bot & | & \text{(bottom concept)} \\
& \neg C & | & \text{(negation)} \\
& C \sqcap D & | & \text{(intersection)} \\
& C \sqcup D & | & \text{(union)} \\
& \forall R_i.C & | & \text{(value restriction)} \\
& \exists R_i.C & | & \text{(existential restriction)}
\end{array}
$$

The $\mathcal{ALC}$ Description Logic

Vocabulary

Fix a set of *atomic concepts* $\{A_1, A_2, \ldots\}$, *roles* $\{R_1, R_2, \ldots\}$ and individuals $\{a_1, a_2, \ldots\}$.

$\mathcal{ALC}$ concept descriptions

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

Axioms

- $C \sqsubseteq D$ and $C \equiv D$ for concept descriptions D and C.
- $C(a)$ and $R(a, b)$ for concept description C, atomic role R and individuals a, b.

$\mathcal{ALC}$ Examples

- $TwoCV \sqsubseteq Car$

$\mathcal{ALC}$ Examples

- $\mathit{TwoCV} \sqsubseteq \mathit{Car}$
  - Any 2CV is a car.

$\mathcal{ALC}$ Examples

- $TwoCV \sqsubseteq Car$
  - Any 2CV is a car.
- $TwoCV(myCar)$

$\mathcal{ALC}$ Examples

- $TwoCV \sqsubseteq Car$
  - Any 2CV is a car.
- $TwoCV(myCar)$
  - $myCar$ is a 2CV.

$\mathcal{ALC}$ Examples

- $\mathit{TwoCV} \sqsubseteq \mathit{Car}$
  - Any 2CV is a car.
- $\mathit{TwoCV}(\mathit{myCar})$
  - myCar is a 2CV.
- $\mathit{owns}(\mathit{martin}, \mathit{myCar})$

$\mathcal{ALC}$ Examples

- $TwoCV \sqsubseteq Car$
  - Any 2CV is a car.
- $TwoCV(myCar)$
  - $myCar$ is a 2CV.
- $owns(martin, myCar)$
  - $martin$ owns $myCar$.

$\mathcal{ALC}$ Examples

- $\mathit{TwoCV} \sqsubseteq \mathit{Car}$
  - Any 2CV is a car.
- $\mathit{TwoCV}(\mathit{myCar})$
  - myCar is a 2CV.
- $\mathit{owns}(\mathit{martin}, \mathit{myCar})$
  - martin owns myCar.
- $\mathit{TwoCV} \sqsubseteq \forall \mathit{driveAxle}.\mathit{FrontAxle}$

$\mathcal{ALC}$ Examples

- $TwoCV \sqsubseteq Car$
  - Any 2CV is a car.
- $TwoCV(myCar)$
  - $myCar$ is a 2CV.
- $owns(martin, myCar)$
  - $martin$ owns $myCar$.
- $TwoCV \sqsubseteq \forall driveAxle.FrontAxle$
  - All drive axles of 2CVs are front axles.

$\mathcal{ALC}$ Examples

- $TwoCV \sqsubseteq Car$
  - Any 2CV is a car.
- $TwoCV(myCar)$
  - $myCar$ is a 2CV.
- $owns(martin, myCar)$
  - $martin$ owns $myCar$.
- $TwoCV \sqsubseteq \forall driveAxle.FrontAxle$
  - All drive axles of 2CVs are front axles.
- $FrontDrivenCar \equiv Car \sqcap \forall driveAxle.FrontAxle$

$\mathcal{ALC}$ Examples

- $\mathit{TwoCV} \sqsubseteq \mathit{Car}$
  - Any 2CV is a car.
- $\mathit{TwoCV}(\mathit{myCar})$
  - *myCar* is a 2CV.
- $\mathit{owns}(\mathit{martin}, \mathit{myCar})$
  - *martin* owns *myCar*.
- $\mathit{TwoCV} \sqsubseteq \forall \mathit{driveAxle}.\mathit{FrontAxle}$
  - All drive axles of 2CVs are front axles.
- $\mathit{FrontDrivenCar} \equiv \mathit{Car} \sqcap \forall \mathit{driveAxle}.\mathit{FrontAxle}$
  - A front driven car is one where all drive axles are front axles.

$\mathcal{ALC}$ Examples

- $TwoCV \sqsubseteq Car$
  - Any 2CV is a car.
- $TwoCV(myCar)$
  - $myCar$ is a 2CV.
- $owns(martin, myCar)$
  - $martin$ owns $myCar$.
- $TwoCV \sqsubseteq \forall driveAxle.FrontAxle$
  - All drive axles of 2CVs are front axles.
- $FrontDrivenCar \equiv Car \sqcap \forall driveAxle.FrontAxle$
  - A front driven car is one where all drive axles are front axles.
- $FrontAxle \sqcap RearAxle \sqsubseteq \bot$ (disjointness)

$\mathcal{ALC}$ Examples

- $TwoCV \sqsubseteq Car$
  - Any 2CV is a car.
- $TwoCV(myCar)$
  - $myCar$ is a 2CV.
- $owns(martin, myCar)$
  - $martin$ owns $myCar$.
- $TwoCV \sqsubseteq \forall driveAxle.FrontAxle$
  - All drive axles of 2CVs are front axles.
- $FrontDrivenCar \equiv Car \sqcap \forall driveAxle.FrontAxle$
  - A front driven car is one where all drive axles are front axles.
- $FrontAxle \sqcap RearAxle \sqsubseteq \bot$ (disjointness)
  - Nothing is both a front axle and a rear axle.

$\mathcal{ALC}$ Examples

- $TwoCV \sqsubseteq Car$
  - Any 2CV is a car.
- $TwoCV(myCar)$
  - $myCar$ is a 2CV.
- $owns(martin, myCar)$
  - $martin$ owns $myCar$.
- $TwoCV \sqsubseteq \forall driveAxle.FrontAxle$
  - All drive axles of 2CVs are front axles.
- $FrontDrivenCar \equiv Car \sqcap \forall driveAxle.FrontAxle$
  - A front driven car is one where all drive axles are front axles.
- $FrontAxle \sqcap RearAxle \sqsubseteq \bot$ (disjointness)
  - Nothing is both a front axle and a rear axle.
- $FourWheelDrive \equiv \exists driveAxle.FrontAxle \sqcap \exists driveAxle.RearAxle$

$\mathcal{ALC}$ Examples

- $TwoCV \sqsubseteq Car$
  - Any 2CV is a car.
- $TwoCV(myCar)$
  - $myCar$ is a 2CV.
- $owns(martin, myCar)$
  - $martin$ owns $myCar$.
- $TwoCV \sqsubseteq \forall driveAxle.FrontAxle$
  - All drive axles of 2CVs are front axles.
- $FrontDrivenCar \equiv Car \sqcap \forall driveAxle.FrontAxle$
  - A front driven car is one where all drive axles are front axles.
- $FrontAxle \sqcap RearAxle \sqsubseteq \bot$ (disjointness)
  - Nothing is both a front axle and a rear axle.
- $FourWheelDrive \equiv \exists driveAxle.FrontAxle \sqcap \exists driveAxle.RearAxle$
  - A 4WD has at least one front drive axle and one rear drive axle.

$\mathcal{ALC}$ Semantics

Interpretation

An interpretation $\mathcal{I}$ fixes a set $\Delta^{\mathcal{I}}$, the *domain*, $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$ for each atomic concept A, $R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ for each role R, and $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$ for each individual a.

$\mathcal{ALC}$ Semantics

Interpretation

An interpretation $\mathcal{I}$ fixes a set $\Delta^{\mathcal{I}}$, the *domain*, $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$ for each atomic concept A, $R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ for each role R, and $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$ for each individual a.

Interpretation of concept descriptions

$$\begin{aligned} \top^{\mathcal{I}} &= \Delta^{\mathcal{I}} \\ \bot^{\mathcal{I}} &= \emptyset \end{aligned}$$

$\mathcal{ALC}$ Semantics

Interpretation

An interpretation $\mathcal{I}$ fixes a set $\Delta^\mathcal{I}$, the *domain*, $A^\mathcal{I} \subseteq \Delta^\mathcal{I}$ for each atomic concept A, $R^\mathcal{I} \subseteq \Delta^\mathcal{I} \times \Delta^\mathcal{I}$ for each role R, and $a^\mathcal{I} \in \Delta^\mathcal{I}$ for each individual a.

Interpretation of concept descriptions

$$
\begin{array}{rcl}
\top^\mathcal{I} & = & \Delta^\mathcal{I} \\
\bot^\mathcal{I} & = & \emptyset \\
(\neg C)^\mathcal{I} & = & \Delta^\mathcal{I} \setminus C^\mathcal{I} \\
(C \sqcap D)^\mathcal{I} & = & C^\mathcal{I} \cap D^\mathcal{I} \\
(C \sqcup D)^\mathcal{I} & = & C^\mathcal{I} \cup D^\mathcal{I}
\end{array}
$$

$\mathcal{ALC}$ Semantics

Interpretation

An interpretation $\mathcal{I}$ fixes a set $\Delta^\mathcal{I}$, the *domain*, $A^\mathcal{I} \subseteq \Delta^\mathcal{I}$ for each atomic concept A, $R^\mathcal{I} \subseteq \Delta^\mathcal{I} \times \Delta^\mathcal{I}$ for each role R, and $a^\mathcal{I} \in \Delta^\mathcal{I}$ for each individual a.

Interpretation of concept descriptions

$$
\begin{array}{rcl}
\top^\mathcal{I} & = & \Delta^\mathcal{I} \\
\bot^\mathcal{I} & = & \emptyset \\
(\neg C)^\mathcal{I} & = & \Delta^\mathcal{I} \setminus C^\mathcal{I} \\
(C \sqcap D)^\mathcal{I} & = & C^\mathcal{I} \cap D^\mathcal{I} \\
(C \sqcup D)^\mathcal{I} & = & C^\mathcal{I} \cup D^\mathcal{I} \\
(\forall R.C)^\mathcal{I} & = & \{a \in \Delta^\mathcal{I} \mid \text{for all } b, \text{ if } \langle a, b\rangle \in R^\mathcal{I} \text{ then } b \in C^\mathcal{I}\} \\
(\exists R.C)^\mathcal{I} & = & \{a \in \Delta^\mathcal{I} \mid \text{there is a } b \text{ where } \langle a, b\rangle \in R^\mathcal{I} \text{ and } b \in C^\mathcal{I}\}
\end{array}
$$

$\mathcal{ALC}$ Semantics

Interpretation

An interpretation $\mathcal{I}$ fixes a set $\Delta^{\mathcal{I}}$, the *domain*, $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$ for each atomic concept A, $R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ for each role R, and $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$ for each individual a.

Interpretation of concept descriptions

$$
\begin{array}{rcl}
\top^{\mathcal{I}} & = & \Delta^{\mathcal{I}} \\
\bot^{\mathcal{I}} & = & \emptyset \\
(\neg C)^{\mathcal{I}} & = & \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}} \\
(C \sqcap D)^{\mathcal{I}} & = & C^{\mathcal{I}} \cap D^{\mathcal{I}} \\
(C \sqcup D)^{\mathcal{I}} & = & C^{\mathcal{I}} \cup D^{\mathcal{I}} \\
(\forall R.C)^{\mathcal{I}} & = & \{a \in \Delta^{\mathcal{I}} \mid \text{for all } b \text{, if } \langle a, b\rangle \in R^{\mathcal{I}} \text{ then } b \in C^{\mathcal{I}}\} \\
(\exists R.C)^{\mathcal{I}} & = & \{a \in \Delta^{\mathcal{I}} \mid \text{there is a } b \text{ where } \langle a, b\rangle \in R^{\mathcal{I}} \text{ and } b \in C^{\mathcal{I}}\}
\end{array}
$$

Interpretation of Axioms

- $\mathcal{I} \models C \sqsubseteq D$ if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ and $\mathcal{I} \models C \equiv D$ if $C^{\mathcal{I}} = D^{\mathcal{I}}$
- $\mathcal{I} \models C(a)$ if $a^{\mathcal{I}} \in C^{\mathcal{I}}$ and $\mathcal{I} \models R(a, b)$ if $\langle a^{\mathcal{I}}, b^{\mathcal{I}}\rangle \in R^{\mathcal{I}}$.

Negation

- The interpretation $\mathcal{I}$ satisfies the axiom $C \equiv \neg D$:

$$\mathcal{I} \models C \equiv \neg D$$

Negation

- The interpretation $\mathcal{I}$ satisfies the axiom $C \equiv \neg D$:

$$\mathcal{I} \models C \equiv \neg D$$

Negation

- The interpretation $\mathcal{I}$ satisfies the axiom $C \equiv \neg D$:

$$
\begin{aligned}
\mathcal{I} \models C &\equiv \neg D \\
&\Leftrightarrow C^{\mathcal{I}} = (\neg D)^{\mathcal{I}}
\end{aligned}
$$

Negation

- The interpretation $\mathcal{I}$ satisfies the axiom $C \equiv \neg D$:

$$
\begin{aligned}
\mathcal{I} \models C \equiv \neg D & \\
& \Leftrightarrow C^{\mathcal{I}} = (\neg D)^{\mathcal{I}} \\
& \Leftrightarrow C^{\mathcal{I}} = (\Delta^{\mathcal{I}} \setminus D^{\mathcal{I}})
\end{aligned}
$$

Negation

- The interpretation $\mathcal{I}$ satisfies the axiom $C \equiv \neg D$:

$$\begin{aligned} \mathcal{I} \models C \equiv \neg D & \\ & \Leftrightarrow C^{\mathcal{I}} = (\neg D)^{\mathcal{I}} \\ & \Leftrightarrow C^{\mathcal{I}} = (\Delta^{\mathcal{I}} \setminus D^{\mathcal{I}}) \end{aligned}$$

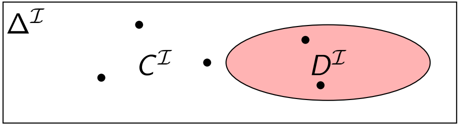
- "A C is not a D."

Negation

- The interpretation $\mathcal{I}$ satisfies the axiom $C \equiv \neg D$:

$$\begin{aligned} \mathcal{I} \models C \equiv \neg D \\ \Leftrightarrow C^{\mathcal{I}} &= (\neg D)^{\mathcal{I}} \\ \Leftrightarrow C^{\mathcal{I}} &= (\Delta^{\mathcal{I}} \setminus D^{\mathcal{I}}) \end{aligned}$$

- "A C is not a D."

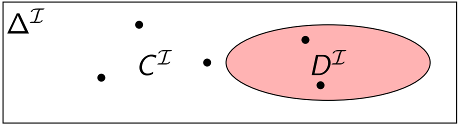


Negation

- The interpretation $\mathcal{I}$ satisfies the axiom $C \equiv \neg D$:

$$\begin{aligned} \mathcal{I} \models C \equiv \neg D \\ \Leftrightarrow C^{\mathcal{I}} = (\neg D)^{\mathcal{I}} \\ \Leftrightarrow C^{\mathcal{I}} = (\Delta^{\mathcal{I}} \setminus D^{\mathcal{I}}) \end{aligned}$$

- "A C is not a D."



- Example: $\mathit{EvenNo} \equiv \neg \mathit{OddNo}$, assuming the domain is $\mathbf{N}$.
"An even number is not an odd number."

Disjointness

- The interpretation $\mathcal{I}$ satisfies the axiom $C \sqcap D \sqsubseteq \bot$:

$$\mathcal{I} \models C \sqcap D \sqsubseteq \bot$$

Disjointness

- The interpretation $\mathcal{I}$ satisfies the axiom $C \sqcap D \sqsubseteq \bot$:

$$\mathcal{I} \models C \sqcap D \sqsubseteq \bot$$

Disjointness

- The interpretation $\mathcal{I}$ satisfies the axiom $C \sqcap D \sqsubseteq \bot$:

$$\begin{aligned} \mathcal{I} \models C \sqcap D \sqsubseteq \bot \\ \Leftrightarrow (C \sqcap D)^{\mathcal{I}} \subseteq \bot^{\mathcal{I}} \end{aligned}$$

Disjointness

- The interpretation $\mathcal{I}$ satisfies the axiom $C \sqcap D \sqsubseteq \bot$:

$$
\begin{aligned}
\mathcal{I} \models C \sqcap D \sqsubseteq \bot & \\
&\Leftrightarrow (C \sqcap D)^{\mathcal{I}} \subseteq \bot^{\mathcal{I}} \\
&\Leftrightarrow C^{\mathcal{I}} \cap D^{\mathcal{I}} \subseteq \emptyset
\end{aligned}
$$

Disjointness

- The interpretation $\mathcal{I}$ satisfies the axiom $C \sqcap D \sqsubseteq \bot$:

$$\begin{aligned} \mathcal{I} \models C \sqcap D \sqsubseteq \bot & \\ & \Leftrightarrow (C \sqcap D)^{\mathcal{I}} \subseteq \bot^{\mathcal{I}} \\ & \Leftrightarrow C^{\mathcal{I}} \cap D^{\mathcal{I}} \subseteq \emptyset \end{aligned}$$

- "Nothing is both a C and a D."

Disjointness

- The interpretation $\mathcal{I}$ satisfies the axiom $C \sqcap D \sqsubseteq \bot$:

$$\begin{aligned}\mathcal{I} \models C \sqcap D \sqsubseteq \bot & \\ & \Leftrightarrow (C \sqcap D)^{\mathcal{I}} \subseteq \bot^{\mathcal{I}} \\ & \Leftrightarrow C^{\mathcal{I}} \cap D^{\mathcal{I}} \subseteq \emptyset\end{aligned}$$

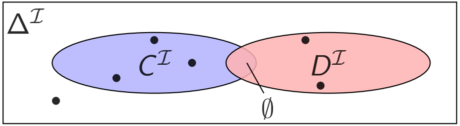
- "Nothing is both a C and a D."
- Equivalent to $C \sqsubseteq \neg D$ (and $D \sqsubseteq \neg C$).

Disjointness

- The interpretation $\mathcal{I}$ satisfies the axiom $C \sqcap D \sqsubseteq \bot$:

$$\begin{aligned} \mathcal{I} \models C \sqcap D \sqsubseteq \bot & \\ & \Leftrightarrow (C \sqcap D)^{\mathcal{I}} \subseteq \bot^{\mathcal{I}} \\ & \Leftrightarrow C^{\mathcal{I}} \cap D^{\mathcal{I}} \subseteq \emptyset \end{aligned}$$

- "Nothing is both a C and a D."
- Equivalent to $C \sqsubseteq \neg D$ (and $D \sqsubseteq \neg C$).

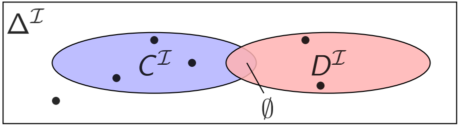


Disjointness

- The interpretation $\mathcal{I}$ satisfies the axiom $C \sqcap D \sqsubseteq \bot$:

$$\begin{aligned} \mathcal{I} \models C \sqcap D \sqsubseteq \bot & \\ & \Leftrightarrow (C \sqcap D)^{\mathcal{I}} \subseteq \bot^{\mathcal{I}} \\ & \Leftrightarrow C^{\mathcal{I}} \cap D^{\mathcal{I}} \subseteq \emptyset \end{aligned}$$

- "Nothing is both a C and a D."
- Equivalent to $C \sqsubseteq \neg D$ (and $D \sqsubseteq \neg C$).



- Example: *FrontAxle* $\sqcap$ *RearAxle* $\sqsubseteq \bot$.
  "A FrontAxle is not a RearAxle, and vice versa."

Existential restrictions

- The interpretation $\mathcal{I}$ satisfies the axiom $C \sqsubseteq \exists R.D$:

$$\mathcal{I} \models C \sqsubseteq \exists R.D$$

Existential restrictions

- The interpretation $\mathcal{I}$ satisfies the axiom $C \sqsubseteq \exists R.D$:

$$\mathcal{I} \models C \sqsubseteq \exists R.D$$

Existential restrictions

- The interpretation $\mathcal{I}$ satisfies the axiom $C \sqsubseteq \exists R.D$:

$$\begin{aligned} \mathcal{I} \models C \sqsubseteq \exists R.D \\ \Leftrightarrow C^{\mathcal{I}} \subseteq (\exists R.D)^{\mathcal{I}} \end{aligned}$$

Existential restrictions

- The interpretation $\mathcal{I}$ satisfies the axiom $C \sqsubseteq \exists R.D$:

$$\begin{aligned} \mathcal{I} \models C \sqsubseteq \exists R.D \\ &\Leftrightarrow C^{\mathcal{I}} \subseteq (\exists R.D)^{\mathcal{I}} \\ &\Leftrightarrow \end{aligned}$$

Existential restrictions

- The interpretation $\mathcal{I}$ satisfies the axiom $C \sqsubseteq \exists R.D$:

$$\begin{aligned}
\mathcal{I} \models C \sqsubseteq \exists R.D \\
&\Leftrightarrow C^{\mathcal{I}} \subseteq (\exists R.D)^{\mathcal{I}} \\
&\Leftrightarrow C^{\mathcal{I}} \subseteq \{a \in \Delta^{\mathcal{I}} \mid \text{there is a } b \text{ where } \langle a, b\rangle \in R^{\mathcal{I}} \text{ and } b \in D^{\mathcal{I}}\}
\end{aligned}$$

Existential restrictions

- The interpretation $\mathcal{I}$ satisfies the axiom $C \sqsubseteq \exists R.D$:

$$\begin{aligned} \mathcal{I} \models C \sqsubseteq \exists R.D & \\ & \Leftrightarrow C^{\mathcal{I}} \subseteq (\exists R.D)^{\mathcal{I}} \\ & \Leftrightarrow C^{\mathcal{I}} \subseteq \{a \in \Delta^{\mathcal{I}} \mid \text{there is a } b \text{ where } \langle a, b \rangle \in R^{\mathcal{I}} \text{ and } b \in D^{\mathcal{I}}\} \end{aligned}$$

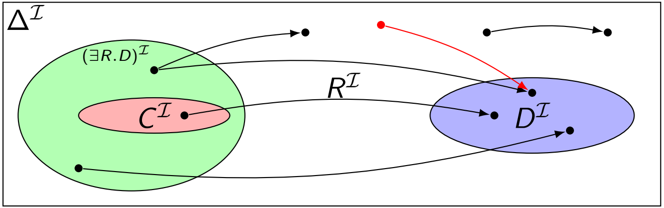
- "A C is R-related to (at least) a D."

Existential restrictions

- The interpretation $\mathcal{I}$ satisfies the axiom $C \sqsubseteq \exists R.D$:

$$\begin{aligned} \mathcal{I} \models C \sqsubseteq \exists R.D & \\ & \Leftrightarrow C^{\mathcal{I}} \subseteq (\exists R.D)^{\mathcal{I}} \\ & \Leftrightarrow C^{\mathcal{I}} \subseteq \{a \in \Delta^{\mathcal{I}} \mid \text{there is a } b \text{ where } \langle a, b\rangle \in R^{\mathcal{I}} \text{ and } b \in D^{\mathcal{I}}\} \end{aligned}$$

- "A C is R-related to (at least) a D."

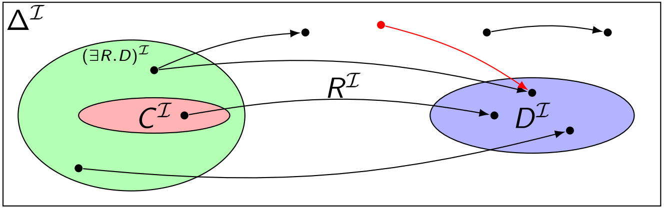


Existential restrictions

- The interpretation $\mathcal{I}$ satisfies the axiom $C \sqsubseteq \exists R.D$:

$$\begin{aligned} \mathcal{I} \models C \sqsubseteq \exists R.D & \\ & \Leftrightarrow C^{\mathcal{I}} \subseteq (\exists R.D)^{\mathcal{I}} \\ & \Leftrightarrow C^{\mathcal{I}} \subseteq \{a \in \Delta^{\mathcal{I}} \mid \text{there is a } b \text{ where } \langle a, b\rangle \in R^{\mathcal{I}} \text{ and } b \in D^{\mathcal{I}}\} \end{aligned}$$

- "A C is R-related to (at least) a D."



- Example: $Toyota \sqsubseteq \exists driveAxle.FrontAxle$.
  "A Toyota has a front axle as drive axle."

Universal restrictions

- The interpretation $\mathcal{I}$ satisfies the axiom $C \sqsubseteq \forall R.D$:

$$\mathcal{I} \models C \sqsubseteq \forall R.D$$

Universal restrictions

- The interpretation $\mathcal{I}$ satisfies the axiom $C \sqsubseteq \forall R.D$:

$$\mathcal{I} \models C \sqsubseteq \forall R.D$$

Universal restrictions

- The interpretation $\mathcal{I}$ satisfies the axiom $C \sqsubseteq \forall R.D$:

$$\begin{aligned} \mathcal{I} \models C \sqsubseteq \forall R.D \\ \Leftrightarrow C^{\mathcal{I}} \subseteq (\forall R.D)^{\mathcal{I}} \end{aligned}$$

Universal restrictions

- The interpretation $\mathcal{I}$ satisfies the axiom $C \sqsubseteq \forall R.D$:

$$
\begin{aligned}
\mathcal{I} \models C \sqsubseteq \forall R.D & \\
& \Leftrightarrow C^{\mathcal{I}} \subseteq (\forall R.D)^{\mathcal{I}} \\
& \Leftrightarrow
\end{aligned}
$$

Universal restrictions

- The interpretation $\mathcal{I}$ satisfies the axiom $C \sqsubseteq \forall R.D$:

$$
\begin{aligned}
\mathcal{I} \models C \sqsubseteq \forall R.D & \\
&\Leftrightarrow C^{\mathcal{I}} \subseteq (\forall R.D)^{\mathcal{I}} \\
&\Leftrightarrow C^{\mathcal{I}} \subseteq \{a \in \Delta^{\mathcal{I}} \mid \text{for all } b, \text{ if } \langle a, b\rangle \in R^{\mathcal{I}} \text{ then } b \in D^{\mathcal{I}}\}
\end{aligned}
$$

Universal restrictions

- The interpretation $\mathcal{I}$ satisfies the axiom $C \sqsubseteq \forall R.D$:

$$\begin{aligned}\mathcal{I} \models C \sqsubseteq \forall R.D & \\ &\Leftrightarrow C^{\mathcal{I}} \subseteq (\forall R.D)^{\mathcal{I}} \\ &\Leftrightarrow C^{\mathcal{I}} \subseteq \{a \in \Delta^{\mathcal{I}} \mid \text{for all } b, \text{ if } \langle a, b\rangle \in R^{\mathcal{I}} \text{ then } b \in D^{\mathcal{I}}\}\end{aligned}$$

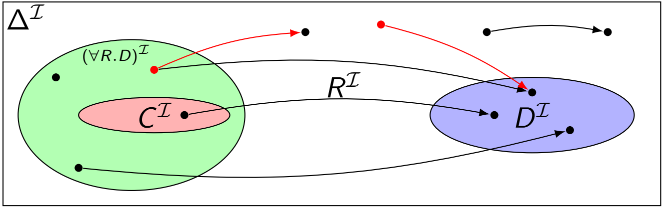
- A C has R-relationships to D's only.

Universal restrictions

- The interpretation $\mathcal{I}$ satisfies the axiom $C \sqsubseteq \forall R.D$:

$$\begin{aligned} \mathcal{I} \models C \sqsubseteq \forall R.D & \\ & \Leftrightarrow C^{\mathcal{I}} \subseteq (\forall R.D)^{\mathcal{I}} \\ & \Leftrightarrow C^{\mathcal{I}} \subseteq \{a \in \Delta^{\mathcal{I}} \mid \text{for all } b, \text{ if } \langle a, b\rangle \in R^{\mathcal{I}} \text{ then } b \in D^{\mathcal{I}}\} \end{aligned}$$

- A C has R-relationships to D's only.

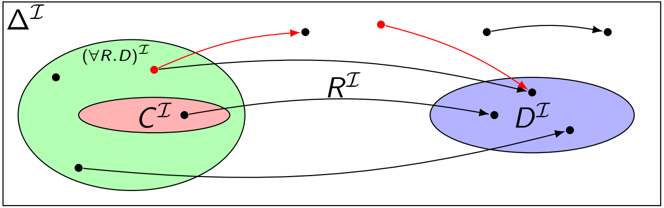


Universal restrictions

- The interpretation $\mathcal{I}$ satisfies the axiom $C \sqsubseteq \forall R.D$:

$$\begin{aligned}\mathcal{I} \models C \sqsubseteq \forall R.D & \\ \Leftrightarrow & C^{\mathcal{I}} \subseteq (\forall R.D)^{\mathcal{I}} \\ \Leftrightarrow & C^{\mathcal{I}} \subseteq \{a \in \Delta^{\mathcal{I}} \mid \text{for all } b, \text{ if } \langle a, b\rangle \in R^{\mathcal{I}} \text{ then } b \in D^{\mathcal{I}}\}\end{aligned}$$

- A C has R-relationships to D's only.



- Example: $\mathit{Lotus} \sqsubseteq \forall \mathit{driveAxle}.\mathit{RearAxle}$.
  "A Lotus has only rear axles as drive axles."

Example interpretation

Assume $\mathcal{K}$ is the knowledge base with the axioms:

$$
\begin{aligned}
\mathit{Donkey} &\sqsubseteq \mathit{Animal} \sqcap \mathit{Stubborn} \\
\mathit{Horse} &\equiv \mathit{Animal} \sqcap \forall \mathit{eats}.\mathit{Chocolate} \\
\mathit{Mule} &\equiv \exists \mathit{hasParent}.\mathit{Horse} \sqcap \exists \mathit{hasParent}.\mathit{Donkey} \\
\exists \mathit{hasParent}.\mathit{Mule} &\sqsubseteq \bot
\end{aligned}
$$

Example interpretation

Assume $\mathcal{K}$ is the knowledge base with the axioms:

$$\begin{aligned}
\mathit{Donkey} &\sqsubseteq \mathit{Animal} \sqcap \mathit{Stubborn} \\
\mathit{Horse} &\equiv \mathit{Animal} \sqcap \forall \mathit{eats}.\mathit{Chocolate} \\
\mathit{Mule} &\equiv \exists \mathit{hasParent}.\mathit{Horse} \sqcap \exists \mathit{hasParent}.\mathit{Donkey} \\
\exists \mathit{hasParent}.\mathit{Mule} &\sqsubseteq \bot
\end{aligned}$$

$$\mathit{Horse}(\mathit{mary}) \quad \mathit{Donkey}(\mathit{sven}) \quad \mathit{hasParent}(\mathit{hannah}, \mathit{mary}) \quad \mathit{hasParent}(\mathit{hannah}, \mathit{sven}) \quad \mathit{eats}(\mathit{mary}, \mathit{carl})$$

Universal Restrictions and `rdfs:range`

- If role R has the *range* C,

Universal Restrictions and `rdfs:range`

- If role R has the *range* C,
- then anything one can reach by R is in C, or

Universal Restrictions and `rdfs:range`

- If role R has the *range* C,
- then anything one can reach by R is in C, or
- for any a and b, if $\langle a, b\rangle \in R^{\mathcal{I}}$, then $b \in C^{\mathcal{I}}$, or

Universal Restrictions and `rdfs:range`

- If role R has the *range* C,
- then anything one can reach by R is in C, or
- for any a and b, if $\langle a, b\rangle \in R^{\mathcal{I}}$, then $b \in C^{\mathcal{I}}$, or
- any a is in the interpretation of $\forall R.C$, or

Universal Restrictions and `rdfs:range`

- If role R has the *range* C,
- then anything one can reach by R is in C, or
- for any a and b, if $\langle a, b \rangle \in R^{\mathcal{I}}$, then $b \in C^{\mathcal{I}}$, or
- any a is in the interpretation of $\forall R.C$, or
- the axiom $\top \sqsubseteq \forall R.C$ holds.

Universal Restrictions and `rdfs:range`

- If role R has the *range* C,
- then anything one can reach by R is in C, or
- for any a and b, if $\langle a, b\rangle \in R^{\mathcal{I}}$, then $b \in C^{\mathcal{I}}$, or
- any a is in the interpretation of $\forall R.C$, or
- the axiom $\top \sqsubseteq \forall R.C$ holds.
- "Everything has R-relationships to C's only."

Universal Restrictions and `rdfs:range`

- If role R has the *range* C,
- then anything one can reach by R is in C, or
- for any a and b, if $\langle a, b\rangle \in R^{\mathcal{I}}$, then $b \in C^{\mathcal{I}}$, or
- any a is in the interpretation of $\forall R.C$, or
- the axiom $\top \sqsubseteq \forall R.C$ holds.
- "Everything has R-relationships to C's only."
- Ranges can be expressed with universal restrictions.

Universal Restrictions and `rdfs:range`

- If role R has the *range* C,
- then anything one can reach by R is in C, or
- for any a and b, if $\langle a, b\rangle \in R^{\mathcal{I}}$, then $b \in C^{\mathcal{I}}$, or
- any a is in the interpretation of $\forall R.C$, or
- the axiom $\top \sqsubseteq \forall R.C$ holds.
- "Everything has R-relationships to C's only."
- Ranges can be expressed with universal restrictions.
- Example:

Universal Restrictions and `rdfs:range`

- If role R has the *range* C,
- then anything one can reach by R is in C, or
- for any a and b, if $\langle a, b\rangle \in R^{\mathcal{I}}$, then $b \in C^{\mathcal{I}}$, or
- any a is in the interpretation of $\forall R.C$, or
- the axiom $\top \sqsubseteq \forall R.C$ holds.
- "Everything has R-relationships to C's only."
- Ranges can be expressed with universal restrictions.
- Example:
  - a drive axle is either a front or a rear axle, so

Universal Restrictions and `rdfs:range`

- If role R has the *range* C,
- then anything one can reach by R is in C, or
- for any a and b, if $\langle a, b\rangle \in R^{\mathcal{I}}$, then $b \in C^{\mathcal{I}}$, or
- any a is in the interpretation of $\forall R.C$, or
- the axiom $\top \sqsubseteq \forall R.C$ holds.
- "Everything has R-relationships to C's only."
- Ranges can be expressed with universal restrictions.
- Example:
  - a drive axle is either a front or a rear axle, so
  - the range of *driveAxle* is *FrontAxle* $\sqcup$ *RearAxle*.

Universal Restrictions and `rdfs:range`

- If role R has the *range* C,
- then anything one can reach by R is in C, or
- for any a and b, if $\langle a, b \rangle \in R^{\mathcal{I}}$, then $b \in C^{\mathcal{I}}$, or
- any a is in the interpretation of $\forall R.C$, or
- the axiom $\top \sqsubseteq \forall R.C$ holds.
- "Everything has R-relationships to C's only."
- Ranges can be expressed with universal restrictions.
- Example:
  - a drive axle is either a front or a rear axle, so
  - the range of *driveAxle* is $\mathit{FrontAxle} \sqcup \mathit{RearAxle}$.
  - Axiom: $\top \sqsubseteq \forall \mathit{driveAxle}.(\mathit{FrontAxle} \sqcup \mathit{RearAxle})$.

Existential Restrictions and `rdfs:domain`

- If role R has the *domain* C,

Existential Restrictions and `rdfs:domain`

- If role R has the *domain* C,
- then anything from which one can go by R is in C, or

Existential Restrictions and `rdfs:domain`

- If role R has the *domain* C,
- then anything from which one can go by R is in C, or
- for any a, if there is a b with $\langle a, b\rangle \in R^{\mathcal{I}}$, then $a \in C^{\mathcal{I}}$, or

Existential Restrictions and `rdfs:domain`

- If role R has the *domain* C,
- then anything from which one can go by R is in C, or
- for any a, if there is a b with $\langle a, b\rangle \in R^{\mathcal{I}}$, then $a \in C^{\mathcal{I}}$, or
- any a in the interpretation of $\exists R.\top$ is in the interpretation of C, or

Existential Restrictions and `rdfs:domain`

- If role R has the *domain* C,
- then anything from which one can go by R is in C, or
- for any a, if there is a b with $\langle a, b \rangle \in R^{\mathcal{I}}$, then $a \in C^{\mathcal{I}}$, or
- any a in the interpretation of $\exists R.\top$ is in the interpretation of C, or
- the axiom $\exists R.\top \sqsubseteq C$ holds.

Existential Restrictions and `rdfs:domain`

- If role R has the *domain* C,
- then anything from which one can go by R is in C, or
- for any a, if there is a b with $\langle a, b\rangle \in R^{\mathcal{I}}$, then $a \in C^{\mathcal{I}}$, or
- any a in the interpretation of $\exists R.\top$ is in the interpretation of C, or
- the axiom $\exists R.\top \sqsubseteq C$ holds.
- "Everything which is R-related (to a thing) is a C."

Existential Restrictions and `rdfs:domain`

- If role R has the *domain* C,
- then anything from which one can go by R is in C, or
- for any a, if there is a b with $\langle a, b\rangle \in R^{\mathcal{I}}$, then $a \in C^{\mathcal{I}}$, or
- any a in the interpretation of $\exists R.\top$ is in the interpretation of C, or
- the axiom $\exists R.\top \sqsubseteq C$ holds.
- "Everything which is R-related (to a thing) is a C."
- Domains can be expressed with existential restrictions.

Existential Restrictions and `rdfs:domain`

- If role R has the *domain* C,
- then anything from which one can go by R is in C, or
- for any a, if there is a b with $\langle a, b\rangle \in R^{\mathcal{I}}$, then $a \in C^{\mathcal{I}}$, or
- any a in the interpretation of $\exists R.\top$ is in the interpretation of C, or
- the axiom $\exists R.\top \sqsubseteq C$ holds.
- "Everything which is R-related (to a thing) is a C."
- Domains can be expressed with existential restrictions.
- Example:

Existential Restrictions and `rdfs:domain`

- If role R has the *domain* C,
- then anything from which one can go by R is in C, or
- for any a, if there is a b with $\langle a, b\rangle \in R^{\mathcal{I}}$, then $a \in C^{\mathcal{I}}$, or
- any a in the interpretation of $\exists R.\top$ is in the interpretation of C, or
- the axiom $\exists R.\top \sqsubseteq C$ holds.
- "Everything which is R-related (to a thing) is a C."
- Domains can be expressed with existential restrictions.
- Example:
  - a drive axle is something cars have, so

Existential Restrictions and `rdfs:domain`

- If role R has the *domain* C,
- then anything from which one can go by R is in C, or
- for any a, if there is a b with $\langle a, b\rangle \in R^{\mathcal{I}}$, then $a \in C^{\mathcal{I}}$, or
- any a in the interpretation of $\exists R.\top$ is in the interpretation of C, or
- the axiom $\exists R.\top \sqsubseteq C$ holds.
- "Everything which is R-related (to a thing) is a C."
- Domains can be expressed with existential restrictions.
- Example:
  - a drive axle is something cars have, so
  - the domain of *driveAxle* is *Car*.

Existential Restrictions and `rdfs:domain`

- If role R has the *domain* C,
- then anything from which one can go by R is in C, or
- for any a, if there is a b with $\langle a, b\rangle \in R^{\mathcal{I}}$, then $a \in C^{\mathcal{I}}$, or
- any a in the interpretation of $\exists R.\top$ is in the interpretation of C, or
- the axiom $\exists R.\top \sqsubseteq C$ holds.
- "Everything which is R-related (to a thing) is a C."
- Domains can be expressed with existential restrictions.
- Example:
  - a drive axle is something cars have, so
  - the domain of *driveAxle* is *Car*.
  - Axiom: $\exists driveAxle.\top \sqsubseteq Car$.

What is the score?

- We still express $C(a)$, $R(x, y)$, $C \sqsubseteq D$ like we did in RDFS,

What is the score?

- We still express $C(a)$, $R(x, y)$, $C \sqsubseteq D$ like we did in RDFS,
- but now we can express complex C's and D's.

What is the score?

- We still express $C(a)$, $R(x, y)$, $C \sqsubseteq D$ like we did in RDFS,
- but now we can express complex C's and D's.
- A concept can be defined by use of other concepts and roles.

What is the score?

- We still express $C(a)$, $R(x,y)$, $C \sqsubseteq D$ like we did in RDFS,
- but now we can express complex C's and D's.
- A concept can be defined by use of other concepts and roles.
- Examples:

What is the score?

- We still express $C(a)$, $R(x, y)$, $C \sqsubseteq D$ like we did in RDFS,
- but now we can express complex C's and D's.
- A concept can be defined by use of other concepts and roles.
- Examples:
  - $Person \sqsubseteq \exists hasMother.\top$ (or $Person \sqsubseteq \exists hasParent.Woman$)

What is the score?

- We still express $C(a)$, $R(x, y)$, $C \sqsubseteq D$ like we did in RDFS,
- but now we can express complex C's and D's.
- A concept can be defined by use of other concepts and roles.
- Examples:
  - $Person \sqsubseteq \exists hasMother.\top$ (or $Person \sqsubseteq \exists hasParent.Woman$)
  - $Penguin \sqsubseteq \forall eats.Fish$

What is the score?

- We still express $C(a)$, $R(x, y)$, $C \sqsubseteq D$ like we did in RDFS,
- but now we can express complex C's and D's.
- A concept can be defined by use of other concepts and roles.
- Examples:
  - $Person \sqsubseteq \exists hasMother.\top$ (or $Person \sqsubseteq \exists hasParent.Woman$)
  - $Penguin \sqsubseteq \forall eats.Fish$
  - $NonSmoker \sqsubseteq \neg Smoker$ (or $NonSmoker \sqcap Smoker \sqsubseteq \bot$)

What is the score?

- We still express $C(a)$, $R(x, y)$, $C \sqsubseteq D$ like we did in RDFS,
- but now we can express complex C's and D's.
- A concept can be defined by use of other concepts and roles.
- Examples:
  - $Person \sqsubseteq \exists hasMother.\top$ (or $Person \sqsubseteq \exists hasParent.Woman$)
  - $Penguin \sqsubseteq \forall eats.Fish$
  - $NonSmoker \sqsubseteq \neg Smoker$ (or $NonSmoker \sqcap Smoker \sqsubseteq \bot$)
  - $\top \sqsubseteq BlackThing \sqcup WhiteThing$

What is the score?

- We still express $C(a)$, $R(x, y)$, $C \sqsubseteq D$ like we did in RDFS,
- but now we can express complex C's and D's.
- A concept can be defined by use of other concepts and roles.
- Examples:
  - $\mathit{Person} \sqsubseteq \exists \mathit{hasMother}.\top$ (or $\mathit{Person} \sqsubseteq \exists \mathit{hasParent}.\mathit{Woman}$)
  - $\mathit{Penguin} \sqsubseteq \forall \mathit{eats}.\mathit{Fish}$
  - $\mathit{NonSmoker} \sqsubseteq \neg \mathit{Smoker}$ (or $\mathit{NonSmoker} \sqcap \mathit{Smoker} \sqsubseteq \bot$)
  - $\top \sqsubseteq \mathit{BlackThing} \sqcup \mathit{WhiteThing}$
  - $\mathit{FreeLunch} \sqsubseteq \bot$

Modelling patterns

So, what can we say with $\mathcal{ALC}$?

✓ Every person has a mother.

Modelling patterns

So, what can we say with $\mathcal{ALC}$?

- ✓ Every person has a mother.
- ✓ Penguins eat only fish. Horses eat only chocolate.

Modelling patterns

So, what can we say with $\mathcal{ALC}$?

- ✓ Every person has a mother.
- ✓ Penguins eat only fish. Horses eat only chocolate.
- ✗ Every nuclear family has two parents, at least two children and a dog.

Modelling patterns

So, what can we say with $\mathcal{ALC}$?

- ✓ Every person has a mother.
- ✓ Penguins eat only fish. Horses eat only chocolate.
- ✗ Every nuclear family has two parents, at least two children and a dog.
- ✓ No smoker is a non-smoker (and vice versa).

Modelling patterns

So, what can we say with $\mathcal{ALC}$?

- ✓ Every person has a mother.
- ✓ Penguins eat only fish. Horses eat only chocolate.
- ✗ Every nuclear family has two parents, at least two children and a dog.
- ✓ No smoker is a non-smoker (and vice versa).
- ✗ Everybody loves Mary.

Modelling patterns

So, what can we say with $\mathcal{ALC}$?

- ✓ Every person has a mother.
- ✓ Penguins eat only fish. Horses eat only chocolate.
- ✗ Every nuclear family has two parents, at least two children and a dog.
- ✓ No smoker is a non-smoker (and vice versa).
- ✗ Everybody loves Mary.
- ✗ Adam is not Eve (and vice versa).

Modelling patterns

So, what can we say with $\mathcal{ALC}$?

- ✓ Every person has a mother.
- ✓ Penguins eat only fish. Horses eat only chocolate.
- ✗ Every nuclear family has two parents, at least two children and a dog.
- ✓ No smoker is a non-smoker (and vice versa).
- ✗ Everybody loves Mary.
- ✗ Adam is not Eve (and vice versa).
- ✓ Everything is black or white.

Modelling patterns

So, what can we say with $\mathcal{ALC}$?

- ✓ Every person has a mother.
- ✓ Penguins eat only fish. Horses eat only chocolate.
- ✗ Every nuclear family has two parents, at least two children and a dog.
- ✓ No smoker is a non-smoker (and vice versa).
- ✗ Everybody loves Mary.
- ✗ Adam is not Eve (and vice versa).
- ✓ Everything is black or white.
- ✓ There is no such thing as a free lunch.

Modelling patterns

So, what can we say with $\mathcal{ALC}$?

- ✓ Every person has a mother.
- ✓ Penguins eat only fish. Horses eat only chocolate.
- ✗ Every nuclear family has two parents, at least two children and a dog.
- ✓ No smoker is a non-smoker (and vice versa).
- ✗ Everybody loves Mary.
- ✗ Adam is not Eve (and vice versa).
- ✓ Everything is black or white.
- ✓ There is no such thing as a free lunch.
- ✗ Brothers of fathers are uncles.

Modelling patterns

So, what can we say with $\mathcal{ALC}$?

- ✓ Every person has a mother.
- ✓ Penguins eat only fish. Horses eat only chocolate.
- ✗ Every nuclear family has two parents, at least two children and a dog.
- ✓ No smoker is a non-smoker (and vice versa).
- ✗ Everybody loves Mary.
- ✗ Adam is not Eve (and vice versa).
- ✓ Everything is black or white.
- ✓ There is no such thing as a free lunch.
- ✗ Brothers of fathers are uncles.
- ✗ My friend's friends are also my friends.

Modelling patterns

So, what can we say with $\mathcal{ALC}$?

- ✓ Every person has a mother.
- ✓ Penguins eat only fish. Horses eat only chocolate.
- ✗ Every nuclear family has two parents, at least two children and a dog.
- ✓ No smoker is a non-smoker (and vice versa).
- ✗ Everybody loves Mary.
- ✗ Adam is not Eve (and vice versa).
- ✓ Everything is black or white.
- ✓ There is no such thing as a free lunch.
- ✗ Brothers of fathers are uncles.
- ✗ My friend's friends are also my friends.
- ✗ If Homer is married to Marge, then Marge is married to Homer.

Modelling patterns

So, what can we say with $\mathcal{ALC}$?

- ✓ Every person has a mother.
- ✓ Penguins eat only fish. Horses eat only chocolate.
- ✗ Every nuclear family has two parents, at least two children and a dog.
- ✓ No smoker is a non-smoker (and vice versa).
- ✗ Everybody loves Mary.
- ✗ Adam is not Eve (and vice versa).
- ✓ Everything is black or white.
- ✓ There is no such thing as a free lunch.
- ✗ Brothers of fathers are uncles.
- ✗ My friend's friends are also my friends.
- ✗ If Homer is married to Marge, then Marge is married to Homer.
- ✗ If Homer is a parent of Bart, then Bart is a child of Homer.

Little Boxes

- Historically, description logic axioms and assertions are put in *boxes*.

Little Boxes

- Historically, description logic axioms and assertions are put in *boxes*.
- The TBox

Little Boxes

- Historically, description logic axioms and assertions are put in *boxes*.
- The TBox
    - is for *terminological knowledge*,

Little Boxes

- Historically, description logic axioms and assertions are put in *boxes*.
- The TBox
  - is for *terminological knowledge*,
  - is independent of any actual instance data, and

Little Boxes

- Historically, description logic axioms and assertions are put in *boxes*.
- The TBox
  - is for *terminological knowledge*,
  - is independent of any actual instance data, and
  - for $\mathcal{ALC}$, it is a set of $\sqsubseteq$ axioms and $\equiv$ axioms.

Little Boxes

- Historically, description logic axioms and assertions are put in *boxes*.
- The TBox
  - is for *terminological knowledge*,
  - is independent of any actual instance data, and
  - for $\mathcal{ALC}$, it is a set of $\sqsubseteq$ axioms and $\equiv$ axioms.
  - Example TBox axioms:

Little Boxes

- Historically, description logic axioms and assertions are put in *boxes*.
- The TBox
  - is for *terminological knowledge*,
  - is independent of any actual instance data, and
  - for $\mathcal{ALC}$, it is a set of $\sqsubseteq$ axioms and $\equiv$ axioms.
  - Example TBox axioms:
    - $\mathit{TwoCV} \sqsubseteq \forall \mathit{driveAxle}.\mathit{FrontAxle}$

Little Boxes

- Historically, description logic axioms and assertions are put in *boxes*.
- The TBox
  - is for *terminological knowledge*,
  - is independent of any actual instance data, and
  - for $\mathcal{ALC}$, it is a set of $\sqsubseteq$ axioms and $\equiv$ axioms.
  - Example TBox axioms:
    - $\mathit{TwoCV} \sqsubseteq \forall \mathit{driveAxle}.\mathit{FrontAxle}$
    - $\mathit{FrontDrivenCar} \equiv \mathit{Car} \sqcap \forall \mathit{driveAxle}.\mathit{FrontAxle}$.

Little Boxes

- Historically, description logic axioms and assertions are put in *boxes*.
- The TBox
  - is for *terminological knowledge*,
  - is independent of any actual instance data, and
  - for $\mathcal{ALC}$, it is a set of $\sqsubseteq$ axioms and $\equiv$ axioms.
  - Example TBox axioms:
    - $\mathit{TwoCV} \sqsubseteq \forall \mathit{driveAxle}.\mathit{FrontAxle}$
    - $\mathit{FrontDrivenCar} \equiv \mathit{Car} \sqcap \forall \mathit{driveAxle}.\mathit{FrontAxle}$.
- The ABox

Little Boxes

- Historically, description logic axioms and assertions are put in *boxes*.
- The TBox
  - is for *terminological knowledge*,
  - is independent of any actual instance data, and
  - for $\mathcal{ALC}$, it is a set of $\sqsubseteq$ axioms and $\equiv$ axioms.
  - Example TBox axioms:
    - $\mathit{TwoCV} \sqsubseteq \forall \mathit{driveAxle}.\mathit{FrontAxle}$
    - $\mathit{FrontDrivenCar} \equiv \mathit{Car} \sqcap \forall \mathit{driveAxle}.\mathit{FrontAxle}$.
- The ABox
  - is for *assertional knowledge*,

Little Boxes

- Historically, description logic axioms and assertions are put in *boxes*.
- The TBox
  - is for *terminological knowledge*,
  - is independent of any actual instance data, and
  - for $\mathcal{ALC}$, it is a set of $\sqsubseteq$ axioms and $\equiv$ axioms.
  - Example TBox axioms:
    - $\mathit{TwoCV} \sqsubseteq \forall \mathit{driveAxle}.\mathit{FrontAxle}$
    - $\mathit{FrontDrivenCar} \equiv \mathit{Car} \sqcap \forall \mathit{driveAxle}.\mathit{FrontAxle}$.
- The ABox
  - is for *assertional knowledge*,
  - contains facts about concrete instances a, b, c,

Little Boxes

- Historically, description logic axioms and assertions are put in *boxes*.
- The TBox
  - is for *terminological knowledge*,
  - is independent of any actual instance data, and
  - for $\mathcal{ALC}$, it is a set of $\sqsubseteq$ axioms and $\equiv$ axioms.
  - Example TBox axioms:
    - $\mathit{TwoCV} \sqsubseteq \forall \mathit{driveAxle}.\mathit{FrontAxle}$
    - $\mathit{FrontDrivenCar} \equiv \mathit{Car} \sqcap \forall \mathit{driveAxle}.\mathit{FrontAxle}$.
- The ABox
  - is for *assertional knowledge*,
  - contains facts about concrete instances a, b, c,
  - a set of concept membership assertions $C(a)$,

Little Boxes

- Historically, description logic axioms and assertions are put in *boxes*.
- The TBox
  - is for *terminological knowledge*,
  - is independent of any actual instance data, and
  - for $\mathcal{ALC}$, it is a set of $\sqsubseteq$ axioms and $\equiv$ axioms.
  - Example TBox axioms:
    - $\mathit{TwoCV} \sqsubseteq \forall \mathit{driveAxle}.\mathit{FrontAxle}$
    - $\mathit{FrontDrivenCar} \equiv \mathit{Car} \sqcap \forall \mathit{driveAxle}.\mathit{FrontAxle}$.
- The ABox
  - is for *assertional knowledge*,
  - contains facts about concrete instances a, b, c,
  - a set of concept membership assertions $C(a)$,
  - and role assertions $R(b, c)$.

Little Boxes

- Historically, description logic axioms and assertions are put in *boxes*.
- The TBox
  - is for *terminological knowledge*,
  - is independent of any actual instance data, and
  - for $\mathcal{ALC}$, it is a set of $\sqsubseteq$ axioms and $\equiv$ axioms.
  - Example TBox axioms:
    - $\mathit{TwoCV} \sqsubseteq \forall \mathit{driveAxle}.\mathit{FrontAxle}$
    - $\mathit{FrontDrivenCar} \equiv \mathit{Car} \sqcap \forall \mathit{driveAxle}.\mathit{FrontAxle}$.
- The ABox
  - is for *assertional knowledge*,
  - contains facts about concrete instances a, b, c,
  - a set of concept membership assertions $C(a)$,
  - and role assertions $R(b, c)$.
  - Example ABox axioms:

Little Boxes

- Historically, description logic axioms and assertions are put in *boxes*.
- The TBox
  - is for *terminological knowledge*,
  - is independent of any actual instance data, and
  - for $\mathcal{ALC}$, it is a set of $\sqsubseteq$ axioms and $\equiv$ axioms.
  - Example TBox axioms:
    - $\mathit{TwoCV} \sqsubseteq \forall \mathit{driveAxle}.\mathit{FrontAxle}$
    - $\mathit{FrontDrivenCar} \equiv \mathit{Car} \sqcap \forall \mathit{driveAxle}.\mathit{FrontAxle}$.
- The ABox
  - is for *assertional knowledge*,
  - contains facts about concrete instances a, b, c,
  - a set of concept membership assertions $C(a)$,
  - and role assertions $R(b, c)$.
  - Example ABox axioms:
    - $\mathit{driveAxle}(\mathit{myCar}, \mathit{axle})$

Little Boxes

- Historically, description logic axioms and assertions are put in *boxes*.
- The TBox
  - is for *terminological knowledge*,
  - is independent of any actual instance data, and
  - for $\mathcal{ALC}$, it is a set of $\sqsubseteq$ axioms and $\equiv$ axioms.
  - Example TBox axioms:
    - $\mathit{TwoCV} \sqsubseteq \forall \mathit{driveAxle}.\mathit{FrontAxle}$
    - $\mathit{FrontDrivenCar} \equiv \mathit{Car} \sqcap \forall \mathit{driveAxle}.\mathit{FrontAxle}$.
- The ABox
  - is for *assertional knowledge*,
  - contains facts about concrete instances a, b, c,
  - a set of concept membership assertions $C(a)$,
  - and role assertions $R(b, c)$.
  - Example ABox axioms:
    - $\mathit{driveAxle}(\mathit{myCar}, \mathit{axle})$
    - $(\mathit{FrontAxle} \sqcup \mathit{RearAxle})(\mathit{axle})$.

TBox Reasoning

Remainder: Entailment

A *entails* B, written $A \models B$, if

$\mathcal{I} \models B$ for all interpretations where $\mathcal{I} \models A$.

TBox Reasoning

Remainder: Entailment

A *entails* B, written $A \models B$, if

$\mathcal{I} \models B$ for all interpretations where $\mathcal{I} \models A$.

- Many reasoning tasks use only the TBox:

TBox Reasoning

Remainder: Entailment

A *entails* B, written $A \models B$, if
$\mathcal{I} \models B$ for all interpretations where $\mathcal{I} \models A$.

- Many reasoning tasks use only the TBox:
- Concept unsatisfiability: Given C, does $\mathcal{T} \models C \sqsubseteq \bot$?

TBox Reasoning

Remainder: Entailment

A *entails* B, written $A \models B$, if

$\mathcal{I} \models B$ for all interpretations where $\mathcal{I} \models A$.

- Many reasoning tasks use only the TBox:
- Concept unsatisfiability: Given C, does $\mathcal{T} \models C \sqsubseteq \bot$?
- Concept subsumption: Given C and D, does $\mathcal{T} \models C \sqsubseteq D$?

TBox Reasoning

Remainder: Entailment

A *entails* B, written $A \models B$, if

$\mathcal{I} \models B$ for all interpretations where $\mathcal{I} \models A$.

- Many reasoning tasks use only the TBox:
- Concept unsatisfiability: Given C, does $\mathcal{T} \models C \sqsubseteq \bot$?
- Concept subsumption: Given C and D, does $\mathcal{T} \models C \sqsubseteq D$?
- Concept equivalence: Given C and D, does $\mathcal{T} \models C \equiv D$?

TBox Reasoning

Remainder: Entailment

A *entails* B, written $A \models B$, if

$\mathcal{I} \models B$ for all interpretations where $\mathcal{I} \models A$.

- Many reasoning tasks use only the TBox:
- Concept unsatisfiability: Given C, does $\mathcal{T} \models C \sqsubseteq \bot$?
- Concept subsumption: Given C and D, does $\mathcal{T} \models C \sqsubseteq D$?
- Concept equivalence: Given C and D, does $\mathcal{T} \models C \equiv D$?
- Concept disjointness: Given C and D, does $\mathcal{T} \models C \sqcap D \sqsubseteq \bot$?

ABox Reasoning

- ABox consistency: Is there a model of $(\mathcal{T}, \mathcal{A})$, i.e., is there an interpretation $\mathcal{I}$ such that $\mathcal{I} \models (\mathcal{T}, \mathcal{A})$?

ABox Reasoning

- ABox consistency: Is there a model of $(\mathcal{T}, \mathcal{A})$, i.e., is there an interpretation $\mathcal{I}$ such that $\mathcal{I} \models (\mathcal{T}, \mathcal{A})$?
- Concept membership: Given C and a, does $(\mathcal{T}, \mathcal{A}) \models C(a)$?

ABox Reasoning

- ABox consistency: Is there a model of $(\mathcal{T}, \mathcal{A})$, i.e., is there an interpretation $\mathcal{I}$ such that $\mathcal{I} \models (\mathcal{T}, \mathcal{A})$?
- Concept membership: Given C and a, does $(\mathcal{T}, \mathcal{A}) \models C(a)$?
- Retrieval: Given C, find all a such that $(\mathcal{T}, \mathcal{A}) \models C(a)$.

ABox Reasoning

- ABox consistency: Is there a model of $(\mathcal{T}, \mathcal{A})$, i.e., is there an interpretation $\mathcal{I}$ such that $\mathcal{I} \models (\mathcal{T}, \mathcal{A})$?
- Concept membership: Given C and a, does $(\mathcal{T}, \mathcal{A}) \models C(a)$?
- Retrieval: Given C, find all a such that $(\mathcal{T}, \mathcal{A}) \models C(a)$.
- Conjunctive Query Answering (SPARQL).

More Expressive Description Logics

- There are description logics including axioms about

More Expressive Description Logics

- There are description logics including axioms about
  - roles, e.g., hierarchy, transitivity

More Expressive Description Logics

- There are description logics including axioms about
  - roles, e.g., hierarchy, transitivity
  - cardinality

More Expressive Description Logics

- There are description logics including axioms about
  - roles, e.g., hierarchy, transitivity
  - cardinality
  - data types, e.g., numbers, strings

More Expressive Description Logics

- There are description logics including axioms about
  - roles, e.g., hierarchy, transitivity
  - cardinality
  - data types, e.g., numbers, strings
  - individuals

More Expressive Description Logics

- There are description logics including axioms about
  - roles, e.g., hierarchy, transitivity
  - cardinality
  - data types, e.g., numbers, strings
  - individuals
  - etc.

More Expressive Description Logics

- There are description logics including axioms about
  - roles, e.g., hierarchy, transitivity
  - cardinality
  - data types, e.g., numbers, strings
  - individuals
  - etc.
- We'll see more in later lectures.

More Expressive Description Logics

- There are description logics including axioms about
  - roles, e.g., hierarchy, transitivity
  - cardinality
  - data types, e.g., numbers, strings
  - individuals
  - etc.
- We'll see more in later lectures.
- The balance of expressivity and complexity is important.

More Expressive Description Logics

- There are description logics including axioms about
  - roles, e.g., hierarchy, transitivity
  - cardinality
  - data types, e.g., numbers, strings
  - individuals
  - etc.
- We'll see more in later lectures.
- The balance of expressivity and complexity is important.
- Too much expressivity makes reasoning tasks

More Expressive Description Logics

- There are description logics including axioms about
  - roles, e.g., hierarchy, transitivity
  - cardinality
  - data types, e.g., numbers, strings
  - individuals
  - etc.
- We'll see more in later lectures.
- The balance of expressivity and complexity is important.
- Too much expressivity makes reasoning tasks
  - first more expensive,

More Expressive Description Logics

- There are description logics including axioms about
  - roles, e.g., hierarchy, transitivity
  - cardinality
  - data types, e.g., numbers, strings
  - individuals
  - etc.
- We'll see more in later lectures.
- The balance of expressivity and complexity is important.
- Too much expressivity makes reasoning tasks
  - first more expensive,
  - then undecidable.

More Expressive Description Logics

- There are description logics including axioms about
  - roles, e.g., hierarchy, transitivity
  - cardinality
  - data types, e.g., numbers, strings
  - individuals
  - etc.
- We'll see more in later lectures.
- The balance of expressivity and complexity is important.
- Too much expressivity makes reasoning tasks
  - first more expensive,
  - then undecidable.
- Much research on how expressivity affects complexity/decidability.

Outline

Quick facts

OWL:

- Acronym for *The Web Ontology Language*.

Quick facts

OWL:

- Acronym for *The Web Ontology Language*.
- Became a W3C recommendation in 2004.

Quick facts

OWL:

- Acronym for *The Web Ontology Language*.
- Became a W3C recommendation in 2004.
- The undisputed standard ontology language.

Quick facts

OWL:

- Acronym for *The Web Ontology Language*.
- Became a W3C recommendation in 2004.
- The undisputed standard ontology language.
- Superseded by OWL 2;

Quick facts

OWL:

- Acronym for *The Web Ontology Language*.
- Became a W3C recommendation in 2004.
- The undisputed standard ontology language.
- Superseded by OWL 2;
    - a backwards compatible extension that adds new capabilities.

Quick facts

OWL:

- Acronym for *The Web Ontology Language*.
- Became a W3C recommendation in 2004.
- The undisputed standard ontology language.
- Superseded by OWL 2;
  - a backwards compatible extension that adds new capabilities.
- Built on Description Logics.

Quick facts

OWL:

- Acronym for *The Web Ontology Language*.
- Became a W3C recommendation in 2004.
- The undisputed standard ontology language.
- Superseded by OWL 2;
  - a backwards compatible extension that adds new capabilities.
- Built on Description Logics.
- Combines DL expressiveness with RDF technology (e.g., URIs, namespaces).

Quick facts

OWL:

- Acronym for *The Web Ontology Language*.
- Became a W3C recommendation in 2004.
- The undisputed standard ontology language.
- Superseded by OWL 2;
  - a backwards compatible extension that adds new capabilities.
- Built on Description Logics.
- Combines DL expressiveness with RDF technology (e.g., URIs, namespaces).
- Extends RDFS with boolean operations, universal/existential restrictions and more.

OWL Syntaxes

- Reminder: RDF is an abstract construction, several concrete syntaxes: RDF/XML, Turtle,...

OWL Syntaxes

- Reminder: RDF is an abstract construction, several concrete syntaxes: RDF/XML, Turtle,. . .
- Same for OWL:

OWL Syntaxes

- Reminder: RDF is an abstract construction, several concrete syntaxes: RDF/XML, Turtle,...
- Same for OWL:
- Defined as set of things that can be said about classes, properties, instances.

OWL Syntaxes

- Reminder: RDF is an abstract construction, several concrete syntaxes: RDF/XML, Turtle,...
- Same for OWL:
- Defined as set of things that can be said about classes, properties, instances.
- DL symbols ($\sqcap, \sqcup, \exists, \forall$) hard to find on keyboard.

OWL Syntaxes

- Reminder: RDF is an abstract construction, several concrete syntaxes: RDF/XML, Turtle,...
- Same for OWL:
- Defined as set of things that can be said about classes, properties, instances.
- DL symbols ($\sqcap, \sqcup, \exists, \forall$) hard to find on keyboard.
- OWL/RDF: Uses RDF to express OWL ontologies.

OWL Syntaxes

- Reminder: RDF is an abstract construction, several concrete syntaxes: RDF/XML, Turtle,...
- Same for OWL:
- Defined as set of things that can be said about classes, properties, instances.
- DL symbols ($\sqcap, \sqcup, \exists, \forall$) hard to find on keyboard.
- OWL/RDF: Uses RDF to express OWL ontologies.
    - Then use any of the RDF serializations.

OWL Syntaxes

- Reminder: RDF is an abstract construction, several concrete syntaxes: RDF/XML, Turtle,. . .
- Same for OWL:
- Defined as set of things that can be said about classes, properties, instances.
- DL symbols ($\sqcap, \sqcup, \exists, \forall$) hard to find on keyboard.
- OWL/RDF: Uses RDF to express OWL ontologies.
  - Then use any of the RDF serializations.
- OWL/XML: a non-RDF XML format.

OWL Syntaxes

- Reminder: RDF is an abstract construction, several concrete syntaxes: RDF/XML, Turtle,...
- Same for OWL:
- Defined as set of things that can be said about classes, properties, instances.
- DL symbols ($\sqcap, \sqcup, \exists, \forall$) hard to find on keyboard.
- OWL/RDF: Uses RDF to express OWL ontologies.
  - Then use any of the RDF serializations.
- OWL/XML: a non-RDF XML format.
- Functional OWL syntax: simple, used in definition.

OWL Syntaxes

- Reminder: RDF is an abstract construction, several concrete syntaxes: RDF/XML, Turtle,. . .
- Same for OWL:
- Defined as set of things that can be said about classes, properties, instances.
- DL symbols ($\sqcap, \sqcup, \exists, \forall$) hard to find on keyboard.
- OWL/RDF: Uses RDF to express OWL ontologies.
  - Then use any of the RDF serializations.
- OWL/XML: a non-RDF XML format.
- Functional OWL syntax: simple, used in definition.
- Manchester OWL syntax: close to DL, but text, used in some tools.

OWL vocabulary in OWL/RDF

- New: `owl:Ontology`, `owl:Class`, `owl:Thing`, properties (next slide), restrictions (`owl:allValuesFrom`, `owl:unionOf`, ...), annotations (`owl:versionInfo`, ...).

OWL vocabulary in OWL/RDF

- New: `owl:Ontology`, `owl:Class`, `owl:Thing`, properties (next slide), restrictions (`owl:allValuesFrom`, `owl:unionOf`, ...), annotations (`owl:versionInfo`, ...).
- From RDF: `rdf:type`, ~~`rdf:Property`~~

OWL vocabulary in OWL/RDF

- New: `owl:Ontology`, `owl:Class`, `owl:Thing`, properties (next slide), restrictions (`owl:allValuesFrom`, `owl:unionOf`, ...), annotations (`owl:versionInfo`, ...).
- From RDF: `rdf:type`, ~~`rdf:Property`~~
- From RDFS: ~~`rdfs:Class`~~, `rdfs:subClassOf`, `rdfs:subPropertyOf`, `rdfs:domain`, `rdfs:range`, `rdfs:label`, `rdfs:comment`, ...

OWL vocabulary in OWL/RDF

- New: `owl:Ontology`, `owl:Class`, `owl:Thing`, properties (next slide), restrictions (`owl:allValuesFrom`, `owl:unionOf`, ...), annotations (`owl:versionInfo`, ...).
- From RDF: `rdf:type`, ~~`rdf:Property`~~
- From RDFS: ~~`rdfs:Class`~~, `rdfs:subClassOf`, `rdfs:subPropertyOf`, `rdfs:domain`, `rdfs:range`, `rdfs:label`, `rdfs:comment`, ...
- (XSD datatypes: `xsd:string`, ...)

Properties in OWL

Three kinds of *mutually disjoint* properties in OWL:

1. `owl:DatatypeProperty`

Properties in OWL

Three kinds of *mutually disjoint* properties in OWL:

1. `owl:DatatypeProperty`
   - link individuals to data values, e.g., `xsd:string`.

Properties in OWL

Three kinds of *mutually disjoint* properties in OWL:

1. `owl:DatatypeProperty`
   - link individuals to data values, e.g., `xsd:string`.
   - Examples: `:hasAge`, `:hasSurname`.

Properties in OWL

Three kinds of *mutually disjoint* properties in OWL:

1. `owl:DatatypeProperty`
   - link individuals to data values, e.g., `xsd:string`.
   - Examples: `:hasAge`, `:hasSurname`.
2. `owl:ObjectProperty`

Properties in OWL

Three kinds of *mutually disjoint* properties in OWL:

1. `owl:DatatypeProperty`
   - link individuals to data values, e.g., `xsd:string`.
   - Examples: `:hasAge`, `:hasSurname`.
2. `owl:ObjectProperty`
   - link individuals to individuals.

Properties in OWL

Three kinds of *mutually disjoint* properties in OWL:

1. `owl:DatatypeProperty`
    - link individuals to data values, e.g., `xsd:string`.
    - Examples: `:hasAge`, `:hasSurname`.
2. `owl:ObjectProperty`
    - link individuals to individuals.
    - Example: `:hasFather`, `:driveAxle`.

Properties in OWL

Three kinds of *mutually disjoint* properties in OWL:

1. `owl:DatatypeProperty`
   - link individuals to data values, e.g., `xsd:string`.
   - Examples: `:hasAge`, `:hasSurname`.
2. `owl:ObjectProperty`
   - link individuals to individuals.
   - Example: `:hasFather`, `:driveAxle`.
3. `owl:AnnotationProperty`

Properties in OWL

Three kinds of *mutually disjoint* properties in OWL:

1. `owl:DatatypeProperty`
   - link individuals to data values, e.g., `xsd:string`.
   - Examples: `:hasAge`, `:hasSurname`.
2. `owl:ObjectProperty`
   - link individuals to individuals.
   - Example: `:hasFather`, `:driveAxle`.
3. `owl:AnnotationProperty`
   - has no logical implication, ignored by reasoners.

Properties in OWL

Three kinds of *mutually disjoint* properties in OWL:

1. `owl:DatatypeProperty`
   - link individuals to data values, e.g., `xsd:string`.
   - Examples: `:hasAge`, `:hasSurname`.
2. `owl:ObjectProperty`
   - link individuals to individuals.
   - Example: `:hasFather`, `:driveAxle`.
3. `owl:AnnotationProperty`
   - has no logical implication, ignored by reasoners.
   - anything can be annotated.

Properties in OWL

Three kinds of *mutually disjoint* properties in OWL:

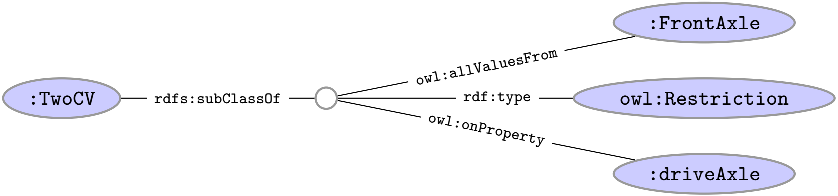
1. `owl:DatatypeProperty`
   - link individuals to data values, e.g., `xsd:string`.
   - Examples: `:hasAge`, `:hasSurname`.
2. `owl:ObjectProperty`
   - link individuals to individuals.
   - Example: `:hasFather`, `:driveAxle`.
3. `owl:AnnotationProperty`
   - has no logical implication, ignored by reasoners.
   - anything can be annotated.
   - Examples: `rdfs:label`, `dc:creator`.

Example: Universal Restrictions in OWL/RDF

- $TwoCV \sqsubseteq \forall driveAxle.FrontAxle$

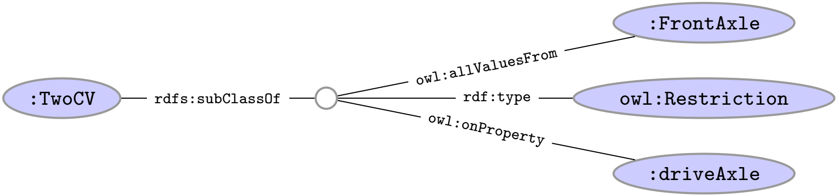
Example: Universal Restrictions in OWL/RDF

- $TwoCV \sqsubseteq \forall driveAxle.FrontAxle$



Example: Universal Restrictions in OWL/RDF

- $TwoCV \sqsubseteq \forall driveAxle.FrontAxle$



- In Turtle syntax:

```
:TwoCV rdfs:subClassOf [ rdf:type owl:Restriction ;
                         owl:onProperty :driveAxle ;
                         owl:allValuesFrom :FrontAxle
                       ] .
```

Example: Universal Restrictions in Other Formats

- $TwoCV \sqsubseteq \forall driveAxle.FrontAxle$

Example: Universal Restrictions in Other Formats

- $TwoCV \sqsubseteq \forall driveAxle.FrontAxle$
- In OWL/XML syntax:

```
<SubClassOf>
  <Class URI=":TwoCV"/>
  <ObjectAllValuesFrom>
     <ObjectProperty URI=":driveAxle"/>
     <Class URI=":FrontAxle"/>
  </ObjectAllValuesFrom>
</SubClassOf>
```

Example: Universal Restrictions in Other Formats

- $\mathit{TwoCV} \sqsubseteq \forall \mathit{driveAxle}.\mathit{FrontAxle}$
- In OWL/XML syntax:

```
<SubClassOf>
  <Class URI=":TwoCV"/>
  <ObjectAllValuesFrom>
     <ObjectProperty URI=":driveAxle"/>
     <Class URI=":FrontAxle"/>
  </ObjectAllValuesFrom>
</SubClassOf>
```

- In OWL Functional syntax:

```
SubClassOf(TwoCV ObjectAllValuesFrom(driveAxle FrontAxle))
```

Manchester OWL Syntax

- Used in Protégé for concept descriptions.

Manchester OWL Syntax

- Used in Protégé for concept descriptions.
- Also has a syntax for axioms, less used.

Manchester OWL Syntax

- Used in Protégé for concept descriptions.
- Also has a syntax for axioms, less used.
- Correspondence to DL constructs:

DL	Manchester
$C \sqcap D$	C and D
$C \sqcup D$	C or D
$\neg C$	not C
$\forall R.C$	R only C
$\exists R.C$	R some C

Manchester OWL Syntax

- Used in Protégé for concept descriptions.
- Also has a syntax for axioms, less used.
- Correspondence to DL constructs:

DL	Manchester
$C \sqcap D$	C and D
$C \sqcup D$	C or D
$\neg C$	not C
$\forall R.C$	R only C
$\exists R.C$	R some C

- Examples:

DL	Manchester
$FrontAxle \sqcup RearAxle$	FrontAxle or RearAxle
$\forall driveAxle.FrontAxle$	driveAxle only FrontAxle
$\exists driveAxle.RearAxle$	driveAxle some RearAxle

Demo: Using Protégé

- Create a Car class.
- Create an Axle class.
- Create FrontAxle and RearAxle as subclasses.
- Make the axle classes disjoint.
- Add a driveAxle object property.
- Add domain Car and range Axle.
- Add 2CV, subclass of Car.
- Add superclass driveAxle only FrontAxle.
- Add Lotus, subclass of Car.
- Add superclass driveAxle only RearAxle.
- Add LandRover, subclass of Car.
- Add superclass driveAxle some FrontAxle.
- Add superclass driveAxle some RearAxle.
- Add 4WD as subclass of Thing.
- Make equivalent to driveAxle some RearAxle and driveAxle some FrontAxle.
- Classify.
- Show inferred class hierarchy: Car $\sqsupseteq$ 4WD $\sqsupseteq$ LandRover.
- Tell story of 2CV Sahara, which is a 2CV with two motors, one front, one back.
- Add Sahara as subclass of 2CV.
- Add 4WD as superclass of Sahara.
- Classify.
- Show that Sahara is equivalent to bottom.
- Explain why. In particular, disjointness of front and rear axles.

The Relationship to Description Logics

- Protégé presents ontologies almost like an OO modelling tool.

The Relationship to Description Logics

- Protégé presents ontologies almost like an OO modelling tool.
- Everything can be mapped to DL axioms!

The Relationship to Description Logics

- Protégé presents ontologies almost like an OO modelling tool.
- Everything can be mapped to DL axioms!
- We have seen how domain and range become ex./univ. restrictions.

The Relationship to Description Logics

- Protégé presents ontologies almost like an OO modelling tool.
- Everything can be mapped to DL axioms!
- We have seen how domain and range become ex./univ. restrictions.
- C and D disjoint: $C \sqsubseteq \neg D$.

The Relationship to Description Logics

- Protégé presents ontologies almost like an OO modelling tool.
- Everything can be mapped to DL axioms!
- We have seen how domain and range become ex./univ. restrictions.
- C and D disjoint: $C \sqsubseteq \neg D$.
- Many ways of saying the same thing in OWL, more in Protégé.

The Relationship to Description Logics

- Protégé presents ontologies almost like an OO modelling tool.
- Everything can be mapped to DL axioms!
- We have seen how domain and range become ex./univ. restrictions.
- C and D disjoint: $C \sqsubseteq \neg D$.
- Many ways of saying the same thing in OWL, more in Protégé.
- Reasoning (e.g., Classification) maps everything to DL first.

OWL in Jena

- Can use usual Jena API to build OWL/RDF ontologies.

OWL in Jena

- Can use usual Jena API to build OWL/RDF ontologies.
- Cumbersome and error prone!

OWL in Jena

- Can use usual Jena API to build OWL/RDF ontologies.
- Cumbersome and error prone!
- Jena class `OntModel` provides convenience methods to create OWL/RDF ontologies, e.g.,

```
OntModel model = ModelFactory.createOntologyModel();
Property driveAxle = model.createProperty(CARS+"driveAxle");
OntClass car = model.createClass(CARS+"Car");
OntClass frontAxle = model.createClass(CARS+"FrontAxle");
Resource r = model.createAllValuesFromRestriction(
                      null, driveAxle, frontAxle);
car.addSuperClass(r);
```

OWL in Jena

- Can use usual Jena API to build OWL/RDF ontologies.
- Cumbersome and error prone!
- Jena class `OntModel` provides convenience methods to create OWL/RDF ontologies, e.g.,

```
OntModel model = ModelFactory.createOntologyModel();
Property driveAxle = model.createProperty(CARS+"driveAxle");
OntClass car = model.createClass(CARS+"Car");
OntClass frontAxle = model.createClass(CARS+"FrontAxle");
Resource r = model.createAllValuesFromRestriction(
                        null, driveAxle, frontAxle);
car.addSuperClass(r);
```

- Can be combined with inferencing mechanisms from lecture 7.

OWL in Jena

- Can use usual Jena API to build OWL/RDF ontologies.
- Cumbersome and error prone!
- Jena class `OntModel` provides convenience methods to create OWL/RDF ontologies, e.g.,

```
OntModel model = ModelFactory.createOntologyModel();
Property driveAxle = model.createProperty(CARS+"driveAxle");
OntClass car = model.createClass(CARS+"Car");
OntClass frontAxle = model.createClass(CARS+"FrontAxle");
Resource r = model.createAllValuesFromRestriction(
                      null, driveAxle, frontAxle);
car.addSuperClass(r);
```

- Can be combined with inferencing mechanisms from lecture 7.
    - See class `OntModelSpec`.

The OWL API

- OWL in Jena means OWL expressed as RDF.

The OWL API

- OWL in Jena means OWL expressed as RDF.
- Still somewhat cumbersome, tied to OWL/RDF peculiarities.

The OWL API

- OWL in Jena means OWL expressed as RDF.
- Still somewhat cumbersome, tied to OWL/RDF peculiarities.
- For pure ontology programming, consider OWL API:

  `http://owlapi.sourceforge.net/`

The OWL API

- OWL in Jena means OWL expressed as RDF.
- Still somewhat cumbersome, tied to OWL/RDF peculiarities.
- For pure ontology programming, consider OWL API:

  `http://owlapi.sourceforge.net/`

- Works on the level of concept descriptions and axioms.

The OWL API

- OWL in Jena means OWL expressed as RDF.
- Still somewhat cumbersome, tied to OWL/RDF peculiarities.
- For pure ontology programming, consider OWL API:

  `http://owlapi.sourceforge.net/`

- Works on the level of concept descriptions and axioms.
- Can parse and write all mentioned OWL formats, and then some.

Next lecture

More about OWL and OWL 2:

- Individuals:

Next lecture

More about OWL and OWL 2:

- Individuals:
  - $=$ and $\neq$, and

Next lecture

More about OWL and OWL 2:

- Individuals:
  - $=$ and $\neq$, and
  - for class and property definition.

Next lecture

More about OWL and OWL 2:

- Individuals:
  - $=$ and $\neq$, and
  - for class and property definition.
- Properties:

Next lecture

More about OWL and OWL 2:

- Individuals:
  - $=$ and $\neq$, and
  - for class and property definition.
- Properties:
  - cardinality,

Next lecture

More about OWL and OWL 2:

- Individuals:
  - $=$ and $\neq$, and
  - for class and property definition.
- Properties:
  - cardinality,
  - transitive, inverse, symmetric, functional properties, and

Next lecture

More about OWL and OWL 2:

- Individuals:
  - $=$ and $\neq$, and
  - for class and property definition.
- Properties:
  - cardinality,
  - transitive, inverse, symmetric, functional properties, and
  - property chains.

Next lecture

More about OWL and OWL 2:

- Individuals:
  - $=$ and $\neq$, and
  - for class and property definition.
- Properties:
  - cardinality,
  - transitive, inverse, symmetric, functional properties, and
  - property chains.
- Datatypes.

Next lecture

More about OWL and OWL 2:

- Individuals:
  - $=$ and $\neq$, and
  - for class and property definition.
- Properties:
  - cardinality,
  - transitive, inverse, symmetric, functional properties, and
  - property chains.
- Datatypes.
- Work through some modelling problems.