Week 3 Lecture 31: Advanced OWL

Description Logics, Restrictions & Property Characteristics
FINAL EXAM COMPREHENSIVE GUIDE

Contents

1 Big Picture and Exam Strategy

1.1 What This Lecture Covers

This is the **MOST CRITICAL** lecture for advanced OWL reasoning. Exam questions will test:

1. **Logic Foundations:**
   - Types of logic (Propositional, First-Order, Modal, Temporal, Fuzzy)
   - Description Logics (DL) as a subset of First-Order Logic
   - DL syntax and semantics
2. **OWL Class Restrictions:**
   - Intersection (`owl:intersectionOf`)
   - Union (`owl:unionOf`)
   - Complement (`owl:complementOf`)
   - Enumerated classes (`owl:oneOf`)
3. **Property Restrictions:**
   - Existential (`owl:someValuesFrom`) - "at least one"
   - Universal (`owl:allValuesFrom`) - "only"
   - Value restriction (`owl:hasValue`)
   - Closure (combining `some` and `only`)
4. **Cardinality Restrictions:**
   - Minimum (`owl:minCardinality`, `owl:minQualifiedCardinality`)
   - Maximum (`owl:maxCardinality`, `owl:maxQualifiedCardinality`)
   - Exact (`owl:cardinality`, `owl:qualifiedCardinality`)
   - Qualified vs. unqualified cardinality
5. **Advanced Property Characteristics:**
   - Inverse (`owl:inverseOf`)
   - Symmetric/Asymmetric
   - Disjoint properties
   - Reflexive/Irreflexive
   - Functional/InverseFunctional
   - Transitive
   - Property chains (`owl:propertyChainAxiom`)
6. **Individual Assertions:**
   - `owl:sameAs` vs. `owl:differentFrom`
   - `owl:AllDifferent`

1.2 Critical Exam Mindset

YOU MUST BE ABLE TO:

1. **Read** any OWL restriction and translate it to English.
2. **Write** OWL axioms given an English description.
3. **Reason** what a reasoner will infer from restrictions.
4. **Detect** inconsistencies in ontologies.
5. **Distinguish** between:
   - SubClassOf ($\sqsubseteq$) vs. EquivalentTo ($\equiv$)
   - Existential (`some`) vs. Universal (`only`)
   - Qualified vs. Unqualified cardinality
   - Functional vs. InverseFunctional
6. **Apply** property characteristics to infer new facts.

If you master this lecture, you can handle 40-50% of the exam!

2 Part 1: Logic Foundations

2.1 What is Logic?

Definition: A formal way to describe a domain using:

- **Axioms:** Assumptions or rules accepted as true
- **Primitives:** Basic things we can talk about

Two Key Commitments:

1. **Ontological Commitment:** *What exists in the model?*
   - Facts, objects, relations, time, events, etc.
2. **Epistemological Commitment:** *What can we know about it?*
   - True/False, Possibility/Necessity, Belief/Disbelief, Certainty/Uncertainty

Analogy: Building a model of a city

- **Ontological:** Do we include roads, buildings, people, traffic?
- **Epistemological:** Do we model traffic as deterministic or probabilistic?

2.2 Types of Logic (EXAM TABLE!)

Logic	Ontological	Epistemological	Example
Propositional	Facts	True/False/Unknown	"It is raining AND streets are wet"
First-Order (FOL)	Facts, Objects, Relations	True/False/Unknown	"All cats are mammals" $\forall x\,(\mathrm{Cat}(x) \to \mathrm{Mammal}(x))$
Modal	Possibility, Necessity	True/False/Unknown	"It is *possible* rain tomorrow" $\Diamond$Rains_tomorrow
Deontic	Obligation, Permission, Forbidden	True/False/Unknown	"It is *forbidden* to plagiarize" $F(\mathrm{plagiarize})$
Temporal	Facts, Objects, Relations, Time	True/False/Unknown	"I am *always* hungry" (Always operator)
Probability	Facts	Certainty [0,1]	"70% chance of rain tomorrow"
Fuzzy	Partial Truth	Truth value [0,1]	"Is it cold?" Very Much / 0.9 Little / 0.25

Key Points:

- **Propositional:** Simplest, no objects/relations (just propositions)
- **FOL:** Introduces objects, quantifiers ($\forall, \exists$)
- **Modal:** $\Box$ (necessity), $\Diamond$ (possibility)
- **Deontic:** O (obligation), P (permission), F (forbidden)
- **Question:** Is $P(\mathrm{plagiarize})$ equivalent to $\neg F(\mathrm{plagiarize})$? **Not necessarily!** Permission $\neq$ not-forbidden (there might be no rule).

2.3 Components of a Logic

Component	Description
Syntax	Rules specifying what a well-formed sentence looks like. Tells us how to build a knowledge base.
Semantics	Rules specifying what a sentence really *means* in the world. Tells us what the knowledge base means.

2.4 Propositional Logic vs. First-Order Logic

Example Statement: "If Jane is younger than Lisa, then Lisa is older than Jane."

Propositional Logic:

```
p: Jane is younger than Lisa
q: Lisa is older than Jane
p => q
```

Problem: No internal structure! Cannot reason about the relationship between "younger" and "older".

First-Order Logic:

```
Younger(Jane, Lisa) => Older(Lisa, Jane)
```

Advantage: We can now define:

$$\forall x, y\,(\mathrm{Younger}(x, y) \leftrightarrow \mathrm{Older}(y, x))$$

2.5 First-Order Logic (FOL) Syntax

Element	Examples
Constants	John, Richard, 2, HeriotWatt
Predicates	Brother, Father, GreaterThan (>), Teaches
Functions	Sqrt, FatherOf, AgeOf
Variables	x, y, a, b
Connectives	$\neg$ (NOT), $\land$ (AND), $\lor$ (OR), $\rightarrow$ (IMPLIES), $\leftrightarrow$ (IFF)
Equality	$=$
Quantifiers	$\forall$ (for all), $\exists$ (there exists)

Key Point: A **predicate** is a property an object can have or a relationship between objects.
FOL Examples:

- $\neg\text{Bird}(\text{Tom})$ - "Tom is not a bird"
- $\neg\text{isMarried}(\text{Alice}, \text{Bob})$ - "Alice is not married to Bob"
- $\forall x\, (\text{Person}(x) \rightarrow \exists y\, (\text{Pet}(y) \land \text{Dog}(y) \land \text{owns}(x, y)))$
  "Every person owns at least one pet that is a dog"

2.6 Description Logics (DL)

What is DL?

Description Logics are a **family of formal knowledge representation languages** used in AI to describe and reason about concepts.

Key Trade-off:

Expressiveness $\overset{\text{Balance}}{\leftrightarrow}$ Decidability

- **Expressiveness:** Ability to represent complex structures and constraints
- **Decidability:** Ability to perform reasoning efficiently

Why Not Full FOL?
Full First-Order Logic offers:

✓ High expressiveness

× Undecidable reasoning (computationally prohibitive)

Solution: DL uses *selected fragments* of FOL to maintain practical reasoning!

2.7 DL Basic Elements

Element	Description
Individuals	Specific objects (e.g., Alice, F20BD, Dubai)
Concepts (Classes)	Groups of individuals (e.g., Person, Course, City)
Roles (Properties)	Relationships between individuals (e.g., teaches, enrolledIn, livesIn)

2.8 Terminology Mapping (MEMORIZE THIS!)

First-Order Logic	Description Logic	OWL
Constant	Individual	Individual
Unary Predicate	Concept	Class
Binary Predicate	Role	Property

2.9 Description Logic Languages

Basic DL Languages:

Language	Name	Allows
AL	Attributive Language	Atomic negation, intersection, restrictions, limited existential quantification
FL	Frame-Based	Concept intersection, restrictions, limited existential quantification, role restriction
EL	Existential Language	Concept intersection, existential restrictions

Note: **S** is used as abbreviation for **ALC**.

2.9.1 DL Extensions (EXAM CRITICAL!)

Symbol	Name	Feature
F	Functional properties	Uniqueness (at most one value)
E	Existential qualification	"There exists at least one individual"
U	Concept union	Class union (`owl:unionOf`)
H	Role hierarchy	`rdfs:subPropertyOf`
C	Role negation	$\neg p(x, y)$
R	Limited role inclusion	Reflexivity, disjointness
O	Nominals	`owl:oneOf`, `owl:hasValue`
I	Inverse property	$p(x, y) \Rightarrow q(y, x)$
N	Cardinality	`owl:cardinality` (counting)
Q	Qualified cardinality	OWL 2 qualified cardinality
(D)	Datatype properties	Data values, datatypes

Example DL Expression:

$$\exists \text{hasPart}.(\text{Wheel} \sqcap \text{HasColor.Red})$$

Meaning: "Has at least one part that is both a Wheel AND has the color Red."

2.10 DL Use Cases

1. **EL (Existential Language):**
   - Fragment of OWL for conceptual modeling.
   - Example: SNOMED CT ontology (300,000+ classes in healthcare).
   - Appendicitis $\equiv$ Inflammation $\sqcap$ $\exists$findingSite.Appendix
2. **RL (Rule Language):**
   - Rule-based OWL for inferencing.
   - SWRL (Semantic Web Rule Language) is used in practice.
   - Example: `Person(?x)` $\wedge$ `EmployedBy(?x, ?company)` $\wedge$ `LocatedIn(?company, London)` $\rightarrow$ `LondonBasedEmpl`
3. **QL (Query Language):**
   - For ontology-based data access (OBDA).
   - Queries roles and individuals (ABox).
   - SPARQL is more general (queries TBox and ABox).

2.11 OWL Flavors/Versions

OWL Version	DL	Characteristics
OWL Full	SROIQ(D)	Very high expressiveness, prohibitive reasoning
OWL-DL	SHOIN	Lower expressiveness, decidable
OWL 2	SROIQ	High expressiveness, very complex reasoning
OWL-Lite	SHIF	Low expressiveness, high reasoning efficiency

Important: Protégé supports SHOIN (OWL-DL).

3 Part 2: OWL Class Restrictions

3.1 Intersection (`owl:intersectionOf`) [AND]

Definition: A class defined as the intersection of two or more classes contains individuals belonging to *all* intersected classes.

DL Notation: $A \sqcap B$

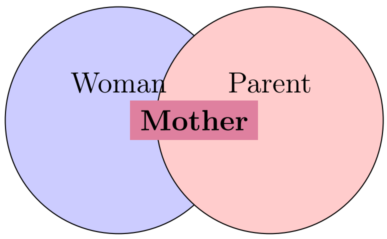
OWL Syntax (Manchester):

```
Class: Mother
  EquivalentTo: Woman and Parent
```

Turtle:

```
:Mother owl:equivalentClass [
  rdf:type owl:Class ;
  owl:intersectionOf ( :Woman :Parent )
] .
```

Visual Representation:



Reasoning:

✓ If Alice is a Mother → Alice is a Woman AND Alice is a Parent

✓ If Alice is a Woman AND Alice is a Parent → Alice is a Mother (only if using `EquivalentTo`)

Exam Trap: If using `SubClassOf` instead of `EquivalentTo`:

```
Mother SubClassOf: Woman and Parent
```

Then:

✓ Mother → Woman AND Parent

× Woman AND Parent $\not\rightarrow$ Mother

3.2 One-Way vs. Two-Way Statements ($\sqsubseteq$ vs. $\equiv$)

CRITICAL DISTINCTION:

One-Way (SubClassOf) $\sqsubseteq$	Two-Way (EquivalentTo) $\equiv$
$A \sqsubseteq B$ means: If x is $A \rightarrow x$ must be B	$A \equiv B$ means: $A \sqsubseteq B$ AND $B \sqsubseteq A$
✓ From $x : A$ infer $x : B$ × From $x : B$ cannot infer $x : A$	✓ From $x : A$ infer $x : B$ ✓ From $x : B$ infer $x : A$
Constraint/Implication	**Definition/Equivalence**

Protégé Syntax:

- $A \sqsubseteq B \rightarrow$ `Class: A SubClassOf: B`
- $A \equiv B \rightarrow$ `Class: A EquivalentTo: B`

3.3 Union (`owl:unionOf`) [OR]

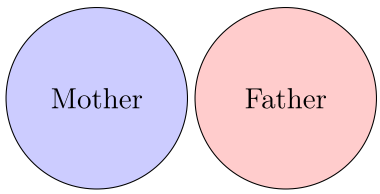
Definition: A union class contains individuals belonging to *at least one* of the union classes.
DL Notation: $A \sqcup B$
Example: Parent is Mother OR Father
OWL Syntax (Manchester):

```
Class: Parent
  EquivalentTo: Mother or Father
```

Turtle:

```
:Parent owl:equivalentClass [
  rdf:type owl:Class ;
  owl:unionOf ( :Mother :Father )
] .
```

Visual Representation:



Parent = Mother ∪ Father

Reasoning:

✓ If Alice is Mother → Alice is Parent

✓ If Bob is Father → Bob is Parent

✓ If Alice is Parent → Alice is (Mother OR Father)

3.4 Complement (`owl:complementOf`) [NOT]

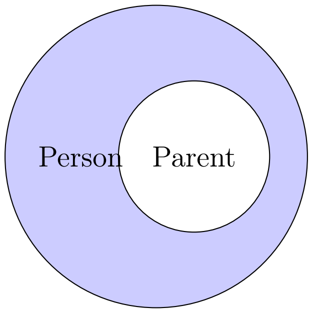
Definition: A complement class contains individuals that belong to one class but *NOT* to another.
DL Notation: $\neg A$
Example: ChildlessPerson = Person AND NOT Parent
OWL Syntax (Manchester):

```
Class: ChildlessPerson
  EquivalentTo: Person and not Parent
```

Turtle:

```
:ChildlessPerson owl:equivalentClass [
  rdf:type owl:Class ;
  owl:intersectionOf (
    :Person
    [ owl:complementOf :Parent ]
  )
] .
```

Visual Representation:



Shaded area = ChildlessPerson

3.5 SubClassOf Complex Class

Definition: A class can be a subclass of an intersection (necessary but NOT sufficient condition).
Example: GrandFather is a subclass of Man AND Parent
OWL Syntax:

```
Class: GrandFather
  SubClassOf: Man and Parent
```

Reasoning:

✓ If John is GrandFather → John is Man AND John is Parent

× If John is Man AND John is Parent → John is **NOT necessarily** GrandFather (could just be Father!)

Key Point: This expresses a **necessary condition** (one-way rule), not sufficient.

3.6 Enumerated Class (`owl:oneOf`)

Definition: Define a class by listing its exact members (closed list).
Example: PartyGuests contains exactly Bill, John, and Mary
OWL Syntax (Manchester):

```
Class: PartyGuests
  EquivalentTo: {Bill, John, Mary}
```

Turtle:

```
:PartyGuests owl:equivalentClass [
  rdf:type owl:Class ;
  owl:oneOf ( :Bill :John :Mary )
] .
```

Use Cases:

- Days of the week: `{Monday, Tuesday, ..., Sunday}`
- Card suits: `{Hearts, Diamonds, Clubs, Spades}`

4 Part 3: Individual Assertions

4.1 `owl:differentFrom`

Problem: OWL does **NOT** assume different names = different entities (no "Unique Name Assumption").
Example: Without explicit statement, Alex and John might be considered the same individual!
Solution: Assert they are different.
OWL Syntax:

```
Individual: Alex
  DifferentFrom: John, Bill
```

Turtle:

```
:Alex owl:differentFrom :John, :Bill .
```

For multiple individuals:

```
[ rdf:type owl:AllDifferent ;
  owl:distinctMembers ( :Alex :John :Bill :Mary )
] .
```

4.2 `owl:sameAs`

Definition: Assert that two individuals are the same.
Example: Alex and Alexander are the same person.
OWL Syntax:

```
Individual: Alex
  SameAs: Alexander
```

Turtle:

```
:Alex owl:sameAs :Alexander .
```

Effect: The two become *indistinguishable*. All properties are combined.
Alternative (if you don't want full merge):

- `skos:exactMatch` - concepts are essentially identical
- `skos:closeMatch` - concepts are similar but not quite identical

5 Part 4: Property Restrictions (THE CORE!)

5.1 Existential Restriction (`owl:someValuesFrom`) [∃]

Definition: "There exists at least ONE value of property P that is in class C"
DL Notation: $\exists P.C$
OWL Syntax (Manchester):

```
P some C
```

Example: Every Parent has at least one child who is a Person.
OWL:

```
Class: Parent
  SubClassOf: hasChild some Person
```

Turtle:

```
:Parent rdfs:subClassOf [
  rdf:type owl:Restriction ;
  owl:onProperty :hasChild ;
  owl:someValuesFrom :Person
] .
```

FOL Translation:

$$\text{Parent}(x) \rightarrow \exists y\, (\text{hasChild}(x, y) \wedge \text{Person}(y))$$

Read as: "If x is a Parent, then there exists at least one y such that x hasChild y AND y is a Person."
Critical Note: `some` guarantees **existence**!

5.2 Universal Restriction (`owl:allValuesFrom`) [∀]

Definition: "ALL values of property P (if any) must be in class C"
DL Notation: $\forall P.C$
OWL Syntax (Manchester):

```
P only C
```

Example: A HappyPerson's children (if any) must all be HappyPerson.
OWL:

```
Class: HappyPerson
  SubClassOf: hasChild only HappyPerson
```

Turtle:

```
:HappyPerson rdfs:subClassOf [
  rdf:type owl:Restriction ;
  owl:onProperty :hasChild ;
  owl:allValuesFrom :HappyPerson
] .
```

FOL Translation:

$$\text{HappyPerson}(x) \rightarrow \forall y\, (\text{hasChild}(x, y) \rightarrow \text{HappyPerson}(y))$$

Read as: "If x is a HappyPerson, then for all y, if x hasChild y, then y must be HappyPerson."
CRITICAL WARNING: `only` does **NOT** guarantee existence!
Example: A HappyPerson with NO children still satisfies `hasChild only HappyPerson`.
Think: "I don't care how many there are (0, 1, or 100), but IF they exist, they must ALL be of class C."

5.3 Closure (Combining `some` and `only`)

Problem: Using ONLY `only` can lead to unintended consequences.
Bad Example: MeatLoversPizza

```
Class: MeatLoversPizza
  SubClassOf: hasTopping only Meat
```

Problem: A plain crust (NO toppings) technically satisfies this! (vacuous truth)
Solution: Combine `some` and `only`!

```
Class: MeatLoversPizza
  SubClassOf: (hasTopping some Meat) and (hasTopping only Meat)
```

Meaning:

- `some Meat`: You **must have** at least one meat topping (existence)
- `only Meat`: You are **forbidden** from having anything other than meat

Turtle:

```
:MeatLoversPizza rdfs:subClassOf [
  rdf:type owl:Class ;
  owl:intersectionOf (
    [ owl:onProperty :hasTopping ; owl:someValuesFrom :Meat ]
    [ owl:onProperty :hasTopping ; owl:allValuesFrom :Meat ]
  )
] .
```

General Pattern:

$$P \text{ some } X \sqcap P \text{ only } X$$

= "At least one P-value is in X, AND all P-values (if any) are in X."

5.4 Value Restriction (`owl:hasValue`)

Definition: "To be in this class, you MUST be connected to specific individual 'a'."
DL Notation: $\exists P.\{a\}$ (existential restriction to a singleton)
OWL Syntax (Manchester):

```
P value a
```

Example 1: JohnChildren = people whose parent is John
OWL:

```
Class: JohnChildren
  EquivalentTo: hasParent value John
```

Turtle:

```
:JohnChildren owl:equivalentClass [
  rdf:type owl:Restriction ;
  owl:onProperty :hasParent ;
  owl:hasValue :John
] .
```

Reasoning (if using EquivalentTo):

✓ If Mary hasParent John → Mary is JohnChildren

✓ If Mary is JohnChildren → Mary hasParent John (reasoner infers this!)

Example 2: ItalianMade = things made in Italy

```
Class: ItalianMade
  EquivalentTo: madeIn value Italy
```

6 Part 5: Cardinality Restrictions (EXAM HEAVY!)

6.1 Overview of Cardinality

Purpose: Count how many DISTINCT individuals are related via a property.

Type	Meaning
`min n`	At least n distinct values
`max n`	At most n distinct values
`exactly n`	Exactly n distinct values (= min + max)

Qualified vs. Unqualified:

Qualified (type matters)	Unqualified (any type)
`P min n C` At least n values of P that are in class C	`P min n` At least n values of P (any type)
Example: `hasChild min 2 Parent` "At least 2 children who are Parents"	Example: `hasChild min 2` "At least 2 children (any type)"

6.2 Minimum Cardinality (`owl:minCardinality`)

Qualified Minimum:
OWL Syntax:

```
P min n C
```

Example: John has at least 2 children who are Parents.
Manchester:

```
Individual: John
  Types: hasChild min 2 Parent
```

Turtle:

```
:John rdf:type [
  rdf:type owl:Restriction ;
  owl:onProperty :hasChild ;
  owl:minQualifiedCardinality "2"^^xsd:nonNegativeInteger ;
  owl:onClass :Parent
] .
```

Unqualified Minimum:

```
Individual: John
  Types: hasChild min 2
```

Turtle:

```
:John rdf:type [
  rdf:type owl:Restriction ;
  owl:onProperty :hasChild ;
  owl:minCardinality "2"^^xsd:nonNegativeInteger
] .
```

6.3 Maximum Cardinality (`owl:maxCardinality`)

Definition: At most n DISTINCT values.

Example: John has at most 4 children who are Parents.

Manchester:

```
Individual: John
  Types: hasChild max 4 Parent
```

Turtle (Qualified):

```
:John rdf:type [
  rdf:type owl:Restriction ;
  owl:onProperty :hasChild ;
  owl:maxQualifiedCardinality "4"^^xsd:nonNegativeInteger ;
  owl:onClass :Parent
] .
```

Turtle (Unqualified):

```
:John rdf:type [
  rdf:type owl:Restriction ;
  owl:onProperty :hasChild ;
  owl:maxCardinality "4"^^xsd:nonNegativeInteger
] .
```

6.4 Exact Cardinality (`owl:cardinality`)

Definition: `P exactly n C` = `(P min n C) AND (P max n C)`

Example: John has exactly 3 children who are Parents.

Manchester:

```
Individual: John
  Types: hasChild exactly 3 Parent
```

Turtle (Qualified):

```
:John rdf:type [
  rdf:type owl:Restriction ;
  owl:onProperty :hasChild ;
  owl:qualifiedCardinality "3"^^xsd:nonNegativeInteger ;
  owl:onClass :Parent
] .
```

Turtle (Unqualified):

```
:John rdf:type [
  rdf:type owl:Restriction ;
  owl:onProperty :hasChild ;
  owl:cardinality "3"^^xsd:nonNegativeInteger
] .
```

WARNING: Using `exactly`, `min`, and `max` in an ontology increases its complexity!

6.5 Cardinality Reasoning Under OWA

Open World Assumption (OWA): We cannot assume information is complete!

Example Scenario:

Data:

```
John hasChild ChildA .
John hasChild ChildB .
John hasChild Child3 .
```

Question: Does John have 4 children?

Answer Under OWA: We don't know!

- We only know John has **at least 3** children.
- There might be a 4th child not stated in the data.
- OWL/Semantic Web operates under OWA.

Contrast with Closed World Assumption (CWA):

- Relational databases use CWA.
- If not stated, it's assumed false.
- Under CWA: John has exactly 3 children (no more).

6.6 Qualified vs. Unqualified Cardinality (Visual Comparison)

Qualified:

```
Person SubClassOf: hasPet min 1 Cat
```

= "Every person has at least 1 pet that is specifically a Cat."

Unqualified:

```
Person SubClassOf: hasPet min 1
```

= "Every person has at least 1 pet (of any type: Cat, Dog, Bird, etc.)."

Summary Table:

Feature	Qualified	Unqualified
Counts specific type?	Yes	No
Example	`hasChild min 2 Parent`	`hasChild min 2`
OWL Property	`owl:minQualifiedCardinality`	`owl:minCardinality`
Requires `owl:onClass`?	Yes	No

7 Part 6: Advanced Property Characteristics

7.1 Inverse Property (`owl:inverseOf`)

Definition: Edge reversal in a graph. If p is the inverse of q, then:

$$p(x, y) \leftrightarrow q(y, x)$$

Example: `hasParent` is inverse of `hasChild`

OWL:

```
ObjectProperty: hasParent
  InverseOf: hasChild
```

Turtle:

```
:hasParent owl:inverseOf :hasChild .
```

Reasoning:

```
Data: Alice hasParent Bob .
Inference: Bob hasChild Alice .
```

Reverse also holds:

```
Data: Bob hasChild Alice .
Inference: Alice hasParent Bob .
```

7.2 Symmetric Property (`owl:SymmetricProperty`)

Definition: If $P(x, y)$ then $P(y, x)$ (relationship holds in both directions).
Example: `hasSpouse` is symmetric.
OWL:

```
ObjectProperty: hasSpouse
  Characteristics: Symmetric
```

Turtle:

```
:hasSpouse rdf:type owl:SymmetricProperty .
```

Reasoning:

```
Data: John hasSpouse Mary .
Inference: Mary hasSpouse John .
```

Key Question: If `hasWife` inverseOf `hasHusband`, and we have:

```
John hasWife Mary
```

What can we infer?
Answer:

```
Mary hasHusband John
```

7.3 Asymmetric Property (`owl:AsymmetricProperty`)

Definition: If $P(x, y)$, then NOT $P(y, x)$.
Example: `hasChild` is asymmetric.
OWL:

```
ObjectProperty: hasChild
  Characteristics: Asymmetric
```

Turtle:

```
:hasChild rdf:type owl:AsymmetricProperty .
```

Inconsistency Detection:

```
Data: John hasChild Mary .
      Mary hasChild John .
Result: INCONSISTENT! (violates asymmetry)
```

7.4 Disjoint Properties (`owl:propertyDisjointWith`)

Definition: Two properties P and Q are disjoint if they can NEVER hold for the same pair (x, y).
Formal: NOT $(P(x, y) \wedge Q(x, y))$
Example: `hasChild` disjoint with `hasSpouse`
OWL:

```
ObjectProperty: hasChild
  DisjointWith: hasSpouse
```

Turtle:

```
:hasChild owl:propertyDisjointWith :hasSpouse .
```

Inconsistency:

```
Data: John hasChild Mary .
      John hasSpouse Mary .
Result: INCONSISTENT!
```

Valid:

```
Data: John hasChild Mary .
      John hasSpouse Anna .
Result: CONSISTENT (different objects)
```

7.5 Reflexive Property (`owl:ReflexiveProperty`)

Definition: A property is reflexive if it can be applied to yourself. For every individual x: $P(x, x)$.
Example: `knows`, `isSameNationalityAs`
OWL:

```
ObjectProperty: knows
  Characteristics: Reflexive
```

Turtle:

```
:knows rdf:type owl:ReflexiveProperty .
```

Valid Examples:

- `John knows John`
- `John isSameNationalityAs John`

7.6 Irreflexive Property (`owl:IrreflexiveProperty`)

Definition: The domain and range **cannot** be the same.
Example: `hasChild`, `isTallerThan`
OWL:

```
ObjectProperty: hasChild
  Characteristics: Irreflexive
```

Turtle:

```
:hasChild rdf:type owl:IrreflexiveProperty .
```

Inconsistency:

```
Data: John hasChild John .
Result: INCONSISTENT! (violates irreflexivity)
```

7.7 Functional Property (`owl:FunctionalProperty`)

Definition: One individual from the domain can only be related to **ONE** individual from the range.
Visual: "Source" (tail of arrow) can only have ONE arrow coming out.
Example: `hasHusband` is functional (in monogamous model).
OWL:

```
ObjectProperty: hasHusband
  Characteristics: Functional
```

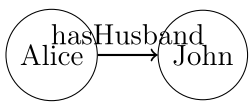
Turtle:

```
:hasHusband rdf:type owl:FunctionalProperty .
```

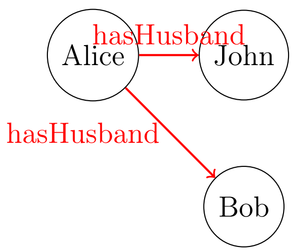
Reasoning:

```
Data: Alice hasHusband John .
      Alice hasHusband Bob .
Inference: John owl:sameAs Bob .
(If they're different individuals → INCONSISTENT!)
```

Diagram:



Valid



Invalid

7.8 Inverse Functional Property (`owl:InverseFunctionalProperty`)

Definition: One individual in the range can only be related to from **ONE** individual in the domain.
Visual: "Target" (head of arrow) can only have ONE arrow pointing into it.
Example: `hasWife` is inverse functional (in monogamous model).
OWL:

```
ObjectProperty: hasWife
  Characteristics: InverseFunctional
```

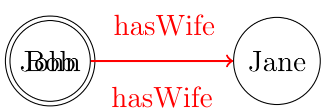
Turtle:

```
:hasWife rdf:type owl:InverseFunctionalProperty .
```

Reasoning:

```
Data: John hasWife Jane .
      Bob hasWife Jane .
Inference: John owl:sameAs Bob .
(If they're different → INCONSISTENT!)
```

Diagram:



Invalid (InverseFunctional)

7.9 Functional vs. Inverse Functional (Deep Dive)

Example 1: `hasBirthCity`

- **Functional?** Yes! A person has only one birth city. (Source $\rightarrow$ 1 Target)
- **Inverse Functional?** No! Millions of people can be born in the same city.

Example 2: `ownsCarWithPlate`

- **Inverse Functional?** Yes! A specific license plate belongs to only one owner. (1 Source $\leftarrow$ Target)
- **Functional?** No! One person can own multiple cars with different plates.

Can Both Be True? YES! 1:1 relationship.
Example: `hasHusband` (strict monogamy)

- Functional: A wife has only one husband.
- Inverse Functional: A husband has only one wife.
- Result: Perfect 1:1 "marriage" of rules!

7.10 Transitive Property (`owl:TransitiveProperty`)

Definition: If x is related to y, and y is related to z, then x is related to z.
Formal: $P(x,y) \wedge P(y,z) \rightarrow P(x,z)$
Example: `isOlderThan`
OWL:

```
ObjectProperty: isOlderThan
  Characteristics: Transitive
```

Turtle:

```
:isOlderThan rdf:type owl:TransitiveProperty .
```

Reasoning:

```
Data: John isOlderThan Charlie .
      Charlie isOlderThan Bob .
Inference: John isOlderThan Bob .
```

7.11 Property Chain (`owl:propertyChainAxiom`)

Definition: Define a property as a chain (sequence) of other properties. If $p(x,y)$ AND $q(y,z)$, then $r(x,z)$.
Example: `hasGrandparent` = chain of `hasParent` and `hasParent`
OWL:

```
ObjectProperty: hasGrandparent
  SubPropertyChain: hasParent o hasParent
```

Turtle:

```
:hasGrandparent owl:propertyChainAxiom ( :hasParent :hasParent ) .
```

Reasoning:

```
Data: Alice hasParent Bob .
      Bob hasParent Carol .
Inference: Alice hasGrandparent Carol .
```

Note: Property chain is **NOT** "transitive"! It creates a **NEW** property!

8 Part 7: Data Properties and Restrictions

8.1 Data Property Restrictions

Data properties connect individuals to **literal values** (not other individuals).

Common Datatypes:

- `xsd:int`, `xsd:float`, `xsd:double`
- `xsd:string`, `xsd:boolean`
- `xsd:date`, `xsd:dateTime`

Example 1: Age must be an integer

```
DataProperty: hasAge
  Range: xsd:int
```

Example 2: Adults have age at least 18

```
Class: Adult
  SubClassOf: hasAge some xsd:int[>= 18]
```

Turtle:

```
:Adult rdfs:subClassOf [
  rdf:type owl:Restriction ;
  owl:onProperty :hasAge ;
  owl:someValuesFrom [
    rdf:type rdfs:Datatype ;
    owl:onDatatype xsd:int ;
    owl:withRestrictions ( [ xsd:minInclusive "18"^^xsd:int ] )
  ]
] .
```

9 Part 8: SWRL (Semantic Web Rule Language)

9.1 What is SWRL?

SWRL = Semantic Web Rule Language

Used to express rules that OWL alone cannot express.

Syntax:

```
Body → Head
```

If the conditions in the Body are true, then infer the Head.

9.2 SWRL Example: Inherited Hobby

Rule Name: S2 – Inherited Hobby

Human-Readable:

- **If:**
  - ?x is a child of ?y
  - ?y likes sport ?s
- **Then:**

- ?x also likes sport ?s

SWRL Syntax:

```
childOf(?x, ?y) ^ likeSport(?y, ?s) -> likeSport(?x, ?s)
```

Reasoning Example:

```
Data: Alice childOf Bob .
      Bob likeSport Football .
Inference: Alice likeSport Football .
```

10 Part 9: Complete Worked Examples

10.1 Example 1: Pizza Ontology

Scenario: Define different types of pizzas based on toppings.
Classes:

- Pizza
- Topping (subclasses: Meat, Vegetable, Cheese)

Property:

- `hasTopping` (Pizza → Topping)

1. VegetarianPizza:

```
Class: VegetarianPizza
  EquivalentTo: Pizza and (hasTopping only Vegetable)
```

Problem: Plain crust with NO toppings would be vegetarian!
Correct:

```
Class: VegetarianPizza
  EquivalentTo: Pizza
    and (hasTopping some Vegetable)
    and (hasTopping only Vegetable)
```

2. MargheritaPizza:

```
Class: MargheritaPizza
  EquivalentTo: Pizza
    and (hasTopping some TomatoTopping)
    and (hasTopping some MozzarellaTopping)
    and (hasTopping only (TomatoTopping or MozzarellaTopping))
```

Meaning: Has at least one tomato, at least one mozzarella, and ONLY those two types.

10.2 Example 2: Family Ontology

Scenario: Model family relationships.
Classes: Person, Man, Woman, Parent
Properties: hasParent, hasChild, hasSibling
Axioms:
1. Parent Definition:

```
Class: Parent
  EquivalentTo: Person and (hasChild some Person)
```

2. **Mother:**

```
Class: Mother
  EquivalentTo: Woman and Parent
```

3. **Father:**

```
Class: Father
  EquivalentTo: Man and Parent
```

4. **Property Characteristics:**

```
ObjectProperty: hasParent
  InverseOf: hasChild

ObjectProperty: hasSibling
  Characteristics: Symmetric

ObjectProperty: hasChild
  Characteristics: Asymmetric, Irreflexive
```

5. **Cardinality:**

```
Class: Person
  SubClassOf: hasParent exactly 2 Person
```

(Every person has exactly 2 biological parents)

Reasoning Example:

```
Data: Alice hasParent Bob .
      Alice hasParent Carol .
      Bob hasChild Mary .

Inferences:
1. Bob hasChild Alice (inverse of hasParent)
2. Carol hasChild Alice (inverse)
3. Bob rdf:type Parent (has at least one child)
4. Carol rdf:type Parent
5. If Bob rdf:type Man → Bob rdf:type Father
```

10.3 Example 3: Detecting Inconsistencies

Ontology:

```
Class: Man
  DisjointWith: Woman

ObjectProperty: hasChild
  Characteristics: Asymmetric, Irreflexive
  DisjointWith: hasSpouse

Class: Parent
  EquivalentTo: Person and (hasChild some Person)
```

Test Case 1: Gender Inconsistency

```
Individual: John
  Types: Man, Woman
Result: INCONSISTENT! (Man and Woman are disjoint)
```

Test Case 2: Asymmetry Violation

```
John hasChild Mary .
Mary hasChild John .
Result: INCONSISTENT! (hasChild is asymmetric)
```

Test Case 3: Disjoint Properties

```
John hasChild Mary .
John hasSpouse Mary .
Result: INCONSISTENT! (hasChild and hasSpouse are disjoint)
```

Test Case 4: Valid

```
John hasChild Mary .
John hasSpouse Anna .
Mary hasSibling Tom .
Result: CONSISTENT
```

11 Part 10: Exam Strategy and Checklists

11.1 Property Characteristics Quick Reference

Characteristic	Formal	Example
Inverse	$p(x, y) \leftrightarrow q(y, x)$	hasParent hasChild
Symmetric	$p(x, y) \to p(y, x)$	hasSpouse, hasSibling
Asymmetric	$p(x, y) \to \neg p(y, x)$	hasChild, isOlderThan
Transitive	$p(x, y) \wedge p(y, z) \to p(x, z)$	isAncestorOf, isPartOf
Functional	Each x has 1 y	hasHusband, hasBirthdate
InverseFunctional	Each y from 1 x	hasWife, hasSSN
Reflexive	$\forall x\, p(x, x)$	knows, isSameNationalityAs
Irreflexive	$\forall x\, \neg p(x, x)$	hasChild, isTallerThan

11.2 Restriction Types Summary

Restriction	Syntax	Meaning
Existential	`P some C`	At least one P-value is in C
Universal	`P only C`	All P-values (if any) are in C
Value	`P value a`	Must be connected to individual a
Min Cardinality	`P min n C`	At least n distinct values in C
Max Cardinality	`P max n C`	At most n distinct values in C
Exact Cardinality	`P exactly n C`	Exactly n distinct values in C

11.3 Common Exam Mistakes

1. **Confusing `some` and `only`:**
   - `some` = existence guaranteed
   - `only` = restriction on all (if any exist)
2. **Forgetting closure:**
   - Using only `only` without `some` allows empty sets!
3. **Mixing up Functional vs. InverseFunctional:**
   - Functional: Source → 1 Target

- InverseFunctional: 1 Source ← Target

4. **Ignoring Open World Assumption:**
   - Cannot assume data is complete!
5. **Confusing SubClassOf and EquivalentTo:**
   - SubClassOf: one-way implication
   - EquivalentTo: two-way equivalence

11.4 Final Exam Checklist

Before the exam, you MUST be able to:

1. Translate between English, DL notation, Manchester syntax, and Turtle.
2. Apply all 8 property characteristics and predict inferences.
3. Write restrictions using `some`, `only`, `value`.
4. Use qualified and unqualified cardinality correctly.
5. Combine intersection, union, complement for complex class definitions.
6. Detect inconsistencies from:
   - Disjoint classes
   - Property characteristics (asymmetric, functional, etc.)
   - Cardinality violations
7. Distinguish between:
   - Existential vs. Universal quantification
   - SubClassOf vs. EquivalentTo
   - Functional vs. InverseFunctional
   - Reflexive vs. Irreflexive
8. Explain why property chains are not transitive.
9. Understand Open World Assumption in reasoning.

12 Ultra-Short Revision (Last-Minute Study)

12.1 Class Restrictions

- **Intersection:** `A and B` - must be in both
- **Union:** `A or B` - must be in at least one
- **Complement:** `not A` - must not be in A
- **Enumeration:** `{a, b, c}` - exact members

12.2 Property Restrictions

- **some:** at least one (existence guaranteed)
- **only:** all (if any) must be
- **value:** connected to specific individual
- **min n:** n distinct values
- **max n:** n distinct values
- **exactly n:** = n distinct values

12.3 8 Key Characteristics

1. **Inverse:** $p(x,y)\ \ q(y,x)$
2. **Symmetric:** $p(x,y) \rightarrow p(y,x)$
3. **Asymmetric:** $p(x,y) \rightarrow \neg p(y,x)$
4. **Transitive:** $p(x,y)\ \ p(y,z) \rightarrow p(x,z)$
5. **Functional:** $x \rightarrow$ max 1 y
6. **InverseFunctional:** $y \leftarrow$ max 1 x
7. **Reflexive:** $x\ p(x,x)$
8. **Irreflexive:** $x\ \neg p(x,x)$

12.4 Critical Distinctions

- **SubClassOf ($\sqsubseteq$):** $A \rightarrow B$ (one-way)
- **EquivalentTo ($\equiv$):** $A\ \ B$ (two-way)
- **Qualified:** counts specific type
- **Unqualified:** counts any type
- **OWA:** cannot assume completeness
- **CWA:** assumes completeness (databases)

13 Conclusion

You've now covered:

1. Logic foundations (Propositional, FOL, DL)
2. Class restrictions (intersection, union, complement, enumeration)
3. Property restrictions (existential, universal, value)
4. Cardinality (min, max, exactly; qualified vs. unqualified)
5. All 8 property characteristics
6. Individual assertions (sameAs, differentFrom)
7. SWRL rules

8. Complete worked examples

MASTER THIS LECTURE = EXAM SUCCESS!

Practice: Write axioms, perform reasoning, detect inconsistencies!