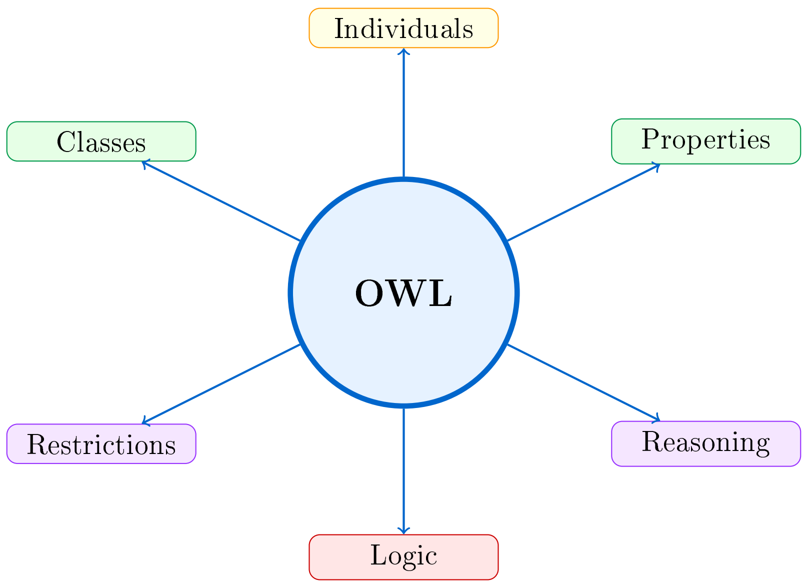
F20BD - Big Data Management

Week 3: Advanced OWL

Ontology Web Language - Advanced Features



Exam-Focused Study Notes

Heriot-Watt University

February 2, 2026

Contents

1 Understanding Logic - The Foundation

KEY CONCEPT

What is Logic? Logic is a formal (precise, mathematical) way to describe knowledge about a domain (a specific area or topic). Think of it as a set of rules that help computers understand and reason about information.

1.1 Why Do We Need Logic in Knowledge Graphs?

Imagine you're building a database about a university:

- You want to store facts: "John is a student", "Mary teaches AI"
- You want the computer to **infer** (figure out) new facts automatically
- Example: If "All students are people" and "John is a student", the computer should conclude "John is a person"

Logic gives us the rules for how to do this correctly!

1.2 Components of Any Logic System

Table 1: Two Main Parts of Logic

lightblue **Component**	**What It Does**	**Real-World Analogy**
Syntax	Rules for writing valid sentences	Grammar rules in English
Semantics	Rules for what sentences mean	Dictionary definitions

EXAMPLE

English Example:

- **Syntax (Grammar):** "The cat sits mat" is WRONG (missing "on the")
- **Semantics (Meaning):** "The cat sits on the mat" tells us WHERE the cat is

Logic Example:

- **Syntax:** $\forall x\ (\text{Cat}(x) \rightarrow \text{Animal}(x))$ is a valid formula
- **Semantics:** This means "Every cat is an animal"

1.3 Types of Logic - Comparison Table

Table 2: Types of Logic (EXAM IMPORTANT)

lightblue **Logic Type**	**What It Handles**	**Example**	**Use Case**
Propositional	Simple true/false statements	"It is raining AND streets are wet"	Basic conditions
First Order Logic (FOL)	Objects, relations, properties	"All cats are mammals"	OWL is based on this!
Modal	Possibility and necessity	"It might rain tomorrow"	Uncertain knowledge
Temporal	Time-based facts	"I am always hungry"	Scheduling, events
Fuzzy	Partial truth (0 to 1 scale)	"It's 0.7 cold" (pretty cold)	Uncertain values

EXAM TIP

For the exam: Remember that OWL uses **Description Logic (DL)**, which is a subset of **First Order Logic**. DL gives us good reasoning power while keeping computations manageable.

2 Propositional Logic vs First Order Logic

2.1 Propositional Logic - The Simple One

DEFINITION

Propositional Logic deals with simple statements that are either TRUE or FALSE. No objects, no relationships - just statements.

Building Blocks (Operators):

Table 3: Propositional Logic Operators

lightblue **Symbol**	**Name**	**Meaning**	**Example**
$\neg P$	Negation	NOT P	¬Raining = "It's NOT raining"
$P \wedge Q$	Conjunction	P AND Q	Raining ∧ Cold = "Raining AND Cold"
$P \vee Q$	Disjunction	P OR Q (or both)	Sunny ∨ Cloudy = "Sunny OR Cloudy"
$P \Rightarrow Q$	Implication	IF P THEN Q	Raining ⇒ Wet = "If raining, then wet"
$P \Leftrightarrow Q$	Biconditional	P IF AND ONLY IF Q	Same as: (P⇒Q) AND (Q⇒P)

EXAMPLE

Propositional Logic Example:
Statement: "If Jane is younger than Lisa, then Lisa is older than Jane"
Let:

- p = "Jane is younger than Lisa"
- q = "Lisa is older than Jane"

Formula: $p \Rightarrow q$
Problem: This doesn't capture WHY this is true (the relationship between ages).

2.2 First Order Logic (FOL) - The Powerful One

DEFINITION

First Order Logic adds:

- **Objects** (individual things): John, Mary, Course123
- **Predicates** (properties/relationships): isStudent(), teaches(), olderThan()
- **Quantifiers**: ∀ (for all) and ∃ (there exists)

EXAMPLE

Same Example in First Order Logic:
Statement: "If Jane is younger than Lisa, then Lisa is older than Jane"
Predicates:

- Younger(x, y) = "x is younger than y"
- Older(x, y) = "x is older than y"

Formula: Younger(Jane, Lisa) ⇒ Older(Lisa, Jane)
Better! Now we can see the actual relationship between the people.

2.3 FOL Syntax - The Building Blocks

Table 4: First Order Logic Elements

lightblue **Element**	**Description**	**Examples**
Constants	Specific objects	John, Mary, 2, Edinburgh
Variables	Placeholders for any object	x, y, a, b
Predicates	Properties or relationships	Student(x), teaches(x,y)
Functions	Return a value	age(x), motherOf(x)
∀	Universal quantifier ("for all")	$\forall x$ means "for every x"
∃	Existential quantifier ("exists")	$\exists x$ means "there exists some x"

EXAMPLE

Complete FOL Example - University Scenario:
English Statements:

1. "Ander is Chinese"
2. "All Chinese people are Persons"
3. "Alona is Ander's daughter"
4. "Children of football fans are football fans"

FOL Translation:

1. Chinese(Ander)
2. $\forall x$ (Chinese(x) ⇒ Person(x))
3. hasDaughter(Ander, Alona)

4. $\forall x, y$ (childOf(y, x) $\wedge$ likes(x, Football) $\Rightarrow$ likes(y, Football))

Reading #4: "For any x and y, if y is a child of x AND x likes Football, THEN y likes Football"

3 Description Logics (DL) - The Heart of OWL

KEY CONCEPT

Why Description Logic?
First Order Logic is very powerful BUT:

- It's too powerful - some problems take forever to solve (undecidable)
- We need something that balances POWER with SPEED

Description Logic = A "smart subset" of FOL that computers can actually work with!

3.1 The Trade-off Diagram

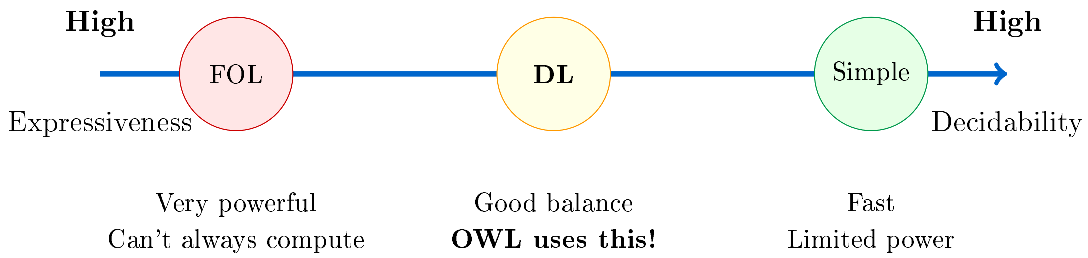


Figure 1: The Expressiveness vs Decidability Trade-off

3.2 DL Terminology - Mapping to OWL

Table 5: Terminology Translation (MEMORIZE THIS!)

lightblue **First Order Logic**	**Description Logic**	**OWL**
Constant	Individual	Individual
Unary Predicate (1 argument)	Concept	Class
Binary Predicate (2 arguments)	Role	Property

EXAMPLE

Same Thing, Three Names:

- FOL: "Student(John)" - John satisfies the Student predicate
- DL: "John is an individual of concept Student"
- OWL: "John is an instance of class Student"

They all mean the same thing!

3.3 DL Language Names - What the Letters Mean

EXAM TIP

The exam may ask about DL language names like SHOIN or SROIQ. Each letter means a specific feature!

Table 6: DL Language Letters Explained

lightblue **Letter**	**Feature**	**What It Allows**
$\mathcal{S}$	$\mathcal{ALC}$ + Transitivity	Chains like: ancestor of ancestor = ancestor
$\mathcal{H}$	Role Hierarchies	subPropertyOf (e.g., hasMother subPropertyOf hasParent)
$\mathcal{O}$	Nominals	owl:oneOf, owl:hasValue (specific individuals)
$\mathcal{I}$	Inverse Properties	hasChild inverse of hasParent
$\mathcal{N}$	Cardinality	Counting: min, max, exactly
$\mathcal{Q}$	Qualified Cardinality	Counting with types: "at least 2 children who are Students"
($\mathcal{D}$)	Datatypes	Strings, integers, dates, etc.
$\mathcal{F}$	Functional Properties	"Can only have ONE value"
$\mathcal{R}$	Role Inclusion	Complex role axioms

EXAMPLE

Decoding SHOIN (used by OWL-DL):

- $\mathcal{S}$ = Basic + Transitive properties
- $\mathcal{H}$ = Property hierarchies allowed
- $\mathcal{O}$ = Can list specific individuals
- $\mathcal{I}$ = Inverse properties allowed
- $\mathcal{N}$ = Can count (cardinality)

4 OWL Versions and Flavours

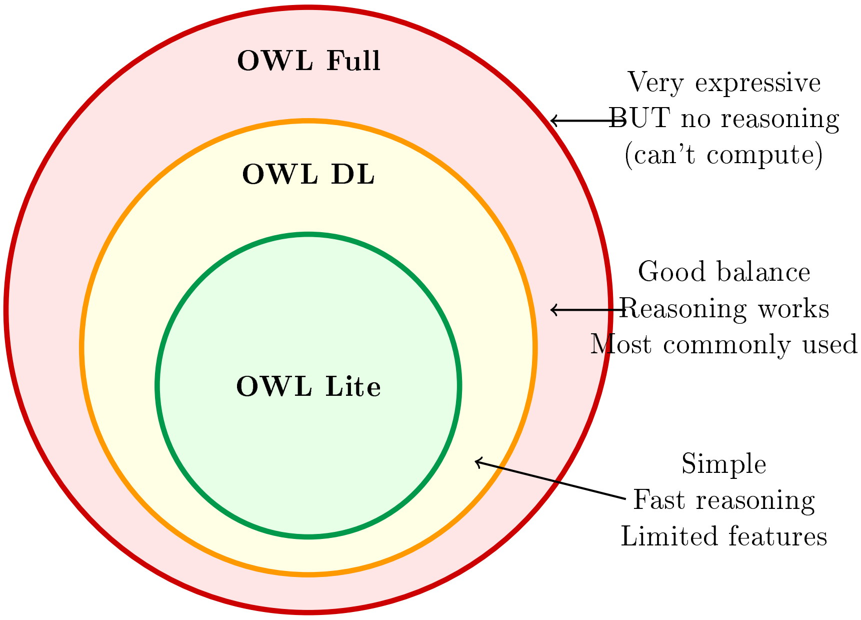


Figure 2: OWL Versions - Nested Relationship

Table 7: OWL Versions Comparison

lightblue **Version**	**DL Language**	**Expressiveness**	**Reasoning**
OWL Full	SROIQ(D)	Very High	NOT decidable
OWL-DL	**SHOIN**	**Good**	**Decidable - USE THIS!**
OWL 2	SHOIQ	High	Very complex
OWL Lite	SHIF	Low	Fast

EXAM TIP

Protégé supports SHOIN (OWL-DL). This is what you'll use in labs and likely what the exam focuses on.

5 OWL Class Restrictions - Combining Classes

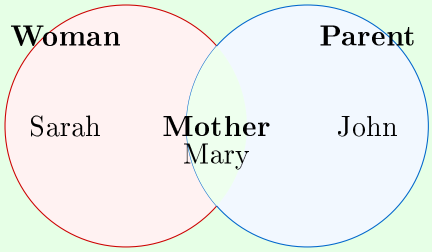
5.1 Class Intersection (AND)

DEFINITION

Intersection creates a class containing individuals that belong to **ALL** the listed classes.
Symbol: $\sqcap$ (in logic) or `and` (in Manchester syntax)
Formula: $\text{Class}(x) \Leftrightarrow \text{ClassA}(x)$ **and** $\text{ClassB}(x)$

EXAMPLE

Example: Defining "Mother"
A Mother is someone who is BOTH a Woman AND a Parent.



Manchester Syntax:

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

Turtle Syntax:

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

Result: If Mary is a Woman AND Mary is a Parent, the reasoner will automatically classify Mary as a Mother!

5.2 Class Union (OR)

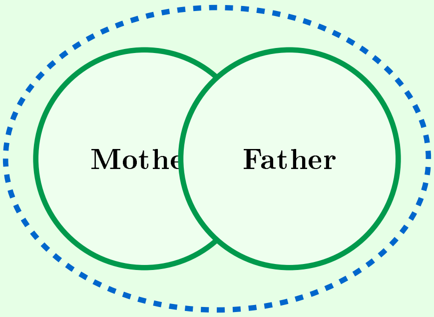
DEFINITION

Union creates a class containing individuals that belong to **AT LEAST ONE** of the listed classes.
Symbol: $\sqcup$ (in logic) or `or` (in Manchester syntax)

EXAMPLE

Example: Defining "Parent"
A Parent is someone who is EITHER a Mother OR a Father (or both).

Parent = Mother OR Father



Manchester Syntax:

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

Result: Anyone who is a Mother will be classified as a Parent. Anyone who is a Father will also be classified as a Parent.

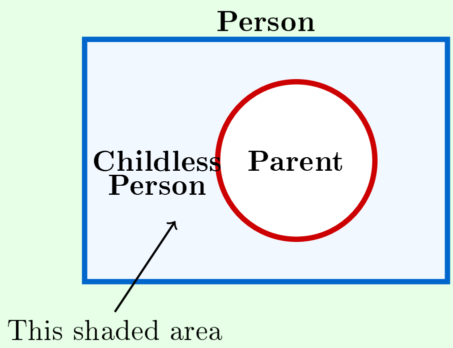
5.3 Class Complement (NOT)

DEFINITION

Complement creates a class containing individuals that are **NOT** members of another class.
Symbol: $\neg$ (in logic) or `not` (in Manchester syntax)

EXAMPLE

Example: Defining "ChildlessPerson"
A ChildlessPerson is a Person who is NOT a Parent.



Manchester Syntax:

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

Reading: "ChildlessPerson is equivalent to being a Person AND NOT being a Parent"

5.4 One-Way vs Two-Way Statements

WARNING / COMMON MISTAKE

CRITICAL EXAM CONCEPT! Understanding the difference between SubClassOf ($\sqsubseteq$) and EquivalentTo ($\equiv$) is essential!

Table 8: SubClassOf vs EquivalentTo

lightblue **Type**	**SubClassOf** ($\sqsubseteq$)	**EquivalentTo** ($\equiv$)
Direction	One-way only	Two-way (both directions)
Meaning	If A, then B (but not vice versa)	A if and only if B
Inference	A $\rightarrow$ B (can infer B from A)	A $\leftrightarrow$ B (can infer either way)
Classification	Cannot auto-classify into A	CAN auto-classify into A

EXAMPLE

Example: GrandFather
Option 1 - SubClassOf (Necessary conditions only):

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

This means:

- ✓ If someone is a GrandFather → they ARE a Man and a Parent
- × If someone is a Man and a Parent → they are NOT necessarily a GrandFather

Option 2 - EquivalentTo (Necessary AND Sufficient):

```
Class: GrandFather
    EquivalentTo: Man and (hasChild some Parent)
```

This means:

- ✓ If someone is a GrandFather → they are a Man with a child who is a Parent
- ✓ If someone is a Man with a child who is a Parent → they ARE a GrandFather

The reasoner can now automatically classify individuals as GrandFathers!

6 Enumerated Classes (owl:oneOf)

DEFINITION

Enumerated Class is a class defined by explicitly listing ALL its members. No other individuals can belong to this class.
Keyword: `owl:oneOf` or just curly braces { } in Manchester syntax

EXAMPLE

Example: Party Guests
You're having a party with exactly 3 guests: Bill, John, and Mary.

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

Turtle Syntax:

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

Result:

- Bill, John, and Mary are automatically PartyGuests
- Nobody else can be a PartyGuest (closed list!)
- If someone adds "Dave" later, Dave will NOT be a PartyGuest

7 Individual Assertions - Same and Different

7.1 The Unique Name Problem

WARNING / COMMON MISTAKE

OWL does NOT assume different names = different things!
This is called the "No Unique Name Assumption" (No-UNA).
Example: "Alex" and "Alexander" could be the SAME person unless you say otherwise!

7.2 owl:differentFrom

DEFINITION

owl:differentFrom explicitly states that two individuals are NOT the same entity.

EXAMPLE

Example: Three Different Students

```
Individual: Alex
    DifferentFrom: John, Bill

Individual: John
    DifferentFrom: Alex, Bill

Individual: Bill
    DifferentFrom: Alex, John
```

Or use `owl:AllDifferent` for multiple at once:

```
DifferentIndividuals: Alex, John, Bill
```

Turtle Syntax:

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

Why does this matter? For cardinality! If you say "John has at least 3 children: A, B, C" but don't say A, B, C are different, the reasoner might think they're all the same person!

7.3 owl:sameAs

DEFINITION

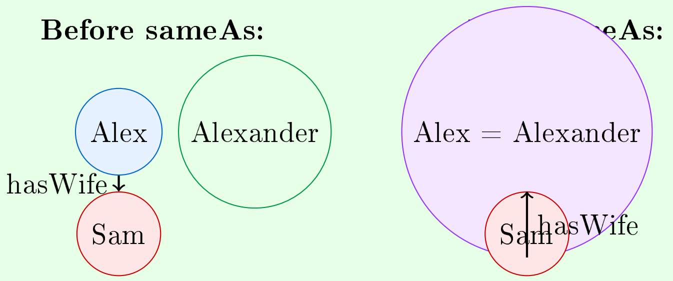
owl:sameAs states that two individuals are actually the SAME entity (just different names/URIs for the same thing).

EXAMPLE

Example: Linking Two Names

```
Individual: Alex
    SameAs: Alexander
    hasWife: Sam
```

Result: Alex and Alexander become completely identical! All properties of Alex apply to Alexander and vice versa.



Warning: This merges EVERYTHING! If Alex has properties you don't want Alexander to have, use `skos:exactMatch` or `skos:closeMatch` instead.

8 Property Restrictions - The Core of OWL

KEY CONCEPT

Property restrictions are the most powerful feature of OWL. They let you define classes based on what properties their members must have.

8.1 Existential Restriction (SOME / ∃)

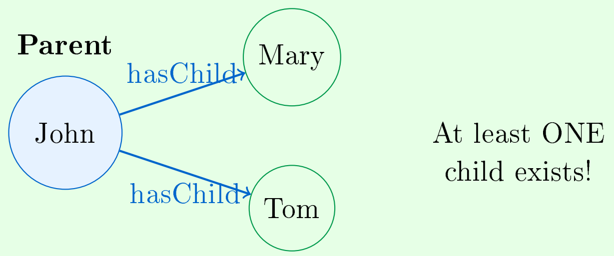
DEFINITION

Existential Restriction (some) means: "There must exist AT LEAST ONE value of property P that belongs to class C"
Symbol: ∃ or `some`
Formula: $C(x) \Rightarrow \exists y \cdot P(x, y)$

EXAMPLE

Example: Defining "Parent"
A Parent is someone who has AT LEAST ONE child who is a Person.

```
Class: Parent
    EquivalentTo: hasChild some Person
```



Key Point: `some` guarantees at least one relationship exists. John MUST have at least one child to be a Parent.

8.2 Universal Restriction (ONLY / ∀)

DEFINITION

Universal Restriction (only) means: "ALL values of property P (if any exist) must belong to class C"
Symbol: ∀ or `only`
Formula: $C(x) \Rightarrow \forall y \cdot (P(x, y) \rightarrow D(y))$

WARNING / COMMON MISTAKE

CRITICAL: `only` does NOT guarantee that any values exist! A person with NO children still satisfies "hasChild only HappyPerson" because they have no children to check!

EXAMPLE

Example: Defining "HappyPerson"
A HappyPerson is someone whose children (if any) are ALL happy.

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

Case 1: Has children **Case 2: No children**

✓ Valid HappyPerson ✓ Also valid!

Both John (with happy children) and Mary (with no children) satisfy the restriction!

8.3 Closure Axiom (SOME + ONLY)

KEY CONCEPT

Closure Axiom = Using `some` AND `only` together to fully define a class.

- `some` ensures at least one value exists
- `only` ensures no other types are allowed

EXAMPLE

Example: MeatLoversPizza
Wrong Definition (using only))

```
Class: MeatLoversPizza
    SubClassOf: hasTopping only MeatTopping
```

Problem: A pizza with NO toppings would count as a MeatLoversPizza! (Empty set satisfies "only")

Correct Definition (using some AND only):

```
Class: MeatLoversPizza
    EquivalentTo: Pizza and
                  (hasTopping some MeatTopping) and
                  (hasTopping only MeatTopping)
```

Breakdown:

- `hasTopping some MeatTopping` = Must have at least one meat topping
- `hasTopping only MeatTopping` = Cannot have any non-meat toppings
- Together = Has meat, and ONLY meat!

8.4 HasValue Restriction (value)

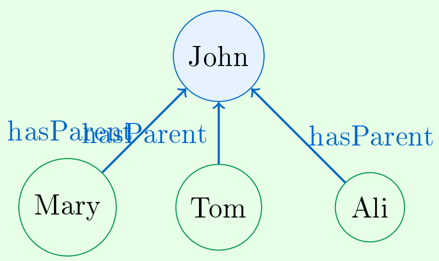
DEFINITION

HasValue Restriction means: "The property must point to a SPECIFIC individual"
Keyword: `value`

EXAMPLE

Example: JohnChildren
JohnChildren are people whose parent is specifically John (not just any parent).

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



All are automatically **JohnChildren**

Inference (with EquivalentTo):

- If Mary hasParent John → Mary is automatically a JohnChildren
- If Ali is a JohnChildren → Ali hasParent John

9 Cardinality Restrictions - Counting Relationships

KEY CONCEPT

Cardinality restrictions let you specify HOW MANY relationships an individual can have. There are three types: `min`, `max`, and `exactly`.

9.1 Minimum Cardinality (min)

DEFINITION

min n means: "Must have AT LEAST n different values"

EXAMPLE

Example: John has at least 2 children who are Parents

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

Turtle Syntax:

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

Meaning: John must have at least 2 different children who are also Parents (grandchildren exist!).

9.2 Maximum Cardinality (max)

DEFINITION

max n means: "Can have AT MOST n different values"

EXAMPLE

Example: A person can have at most 2 biological parents

```
Class: Person
    SubClassOf: hasBiologicalParent max 2 Person
```

Meaning: No person can have more than 2 biological parents.

9.3 Exact Cardinality (exactly)

DEFINITION

exactly n means: "Must have EXACTLY n different values (no more, no less)"
Equivalent to: (min n) AND (max n)

EXAMPLE

Example: A bicycle has exactly 2 wheels

```
Class: Bicycle
```

```
    SubClassOf: hasWheel exactly 2 Wheel
```

```
:Bicycle rdfs:subClassOf
    [ rdf:type owl:Restriction ;
      owl:onProperty :hasWheel ;
      owl:qualifiedCardinality "2"^^xsd:nonNegativeInteger ;
      owl:onClass :Wheel ] .
```

9.4 Qualified vs Unqualified Cardinality

Table 9: Qualified vs Unqualified Cardinality

lightblue **Type**	**Unqualified**	**Qualified**
Counts	Any values	Only values of a specific type
Example	hasChild min 2	hasChild min 2 Parent
Meaning	At least 2 children (any type)	At least 2 children who are Parents

EXAMPLE

Example: FullyMarriedMan (with exactly 4 wives)

```
Class: FullyMarriedMan
    EquivalentTo: hasWife exactly 4
    SubClassOf: Man
```

This is **unqualified** - we're just counting, not checking what type the wives are.
Qualified version would be:

```
Class: FullyMarriedMan
    EquivalentTo: hasWife exactly 4 Woman
```

WARNING / COMMON MISTAKE

Open World Assumption (OWA) Warning!
If John has 3 listed children (A, B, C) and we ask "Does John have 4 children?"
The answer is: **UNKNOWN** (not "No")!
Why? There might be a 4th child we don't know about. OWL doesn't assume we know everything!
To count correctly, you must:

1. List all children explicitly
2. Declare them as `DifferentIndividuals`

10 Property Characteristics

10.1 Inverse Properties (owl:inverseOf)

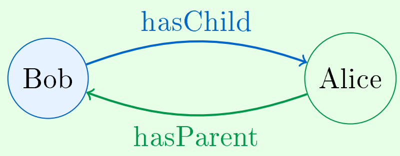
DEFINITION

Inverse Property: If P(x, y) then Q(y, x) - the relationship goes the opposite direction.

EXAMPLE

Example: hasParent and hasChild

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



If Bob hasChild Alice
Then Alice hasParent Bob (automatically!)

10.2 Symmetric and Asymmetric Properties

Table 10: Symmetric vs Asymmetric

lightblue **Type**	**Meaning**	**Example**
Symmetric	If P(x,y) then P(y,x)	hasSpouse, isSiblingOf
Asymmetric	If P(x,y) then NOT P(y,x)	hasChild, isOlderThan

EXAMPLE

Symmetric: hasSpouse

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

If John hasSpouse Mary → Mary hasSpouse John (automatically!)

Asymmetric: hasChild

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

If John hasChild Mary → Mary CANNOT hasChild John

10.3 Reflexive and Irreflexive Properties

Table 11: Reflexive vs Irreflexive

lightblue **Type**	**Meaning**	**Example**
Reflexive	P(x,x) is always true	knows (you know yourself)
Irreflexive	P(x,x) is never true	hasChild (can't be own child)

EXAMPLE

Example:

```
ObjectProperty: knows
    Characteristics: Reflexive

ObjectProperty: hasChild
    Characteristics: Irreflexive
```

- knows(John, John) = TRUE (John knows himself)
- hasChild(John, John) = IMPOSSIBLE (John can't be his own child)

10.4 Functional and Inverse Functional Properties

KEY CONCEPT

Functional = Each subject can have only ONE value for this property
Inverse Functional = Each value can have only ONE subject

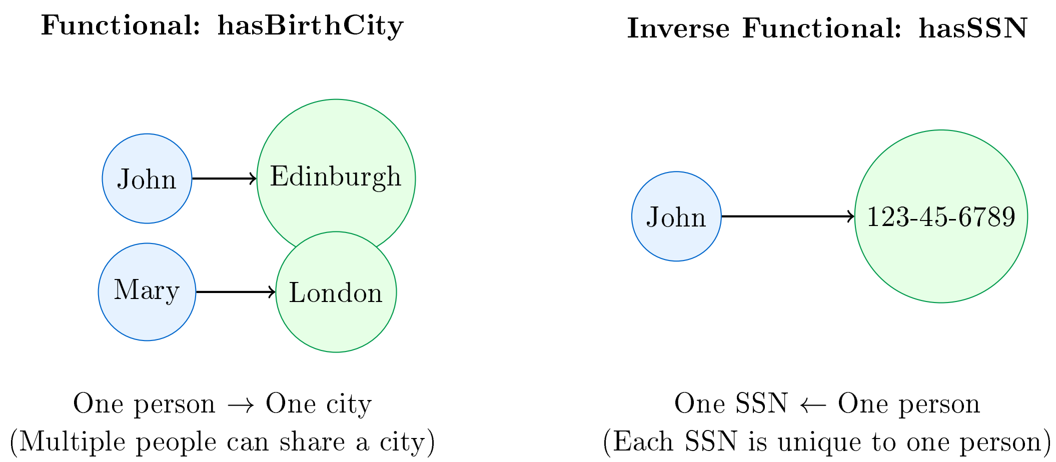


Figure 3: Functional vs Inverse Functional Properties

EXAMPLE

hasHusband Example:

```
ObjectProperty: hasHusband
    Characteristics: Functional
    InverseOf: hasWife

ObjectProperty: hasWife
    Characteristics: InverseFunctional
```

hasHusband is Functional:

- Each woman can have only ONE husband
- Mary hasHusband John **AND** Mary hasHusband Bob → VIOLATION!

hasWife is InverseFunctional:

- Each woman can be the wife of only ONE man
- John hasWife Mary **AND** Bob hasWife Mary → VIOLATION!

10.5 Transitive Properties

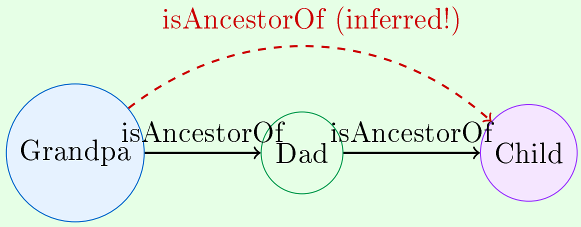
DEFINITION

Transitive: If P(x,y) and P(y,z) then P(x,z)
Think of it as "jumping through" intermediate nodes.

EXAMPLE

Example: isAncestorOf

```
ObjectProperty: isAncestorOf
    Characteristics: Transitive
```



Grandpa isAncestorOf Dad AND Dad isAncestorOf Child
$\Rightarrow$ Grandpa isAncestorOf Child (automatically inferred!)

10.6 Property Chains

DEFINITION

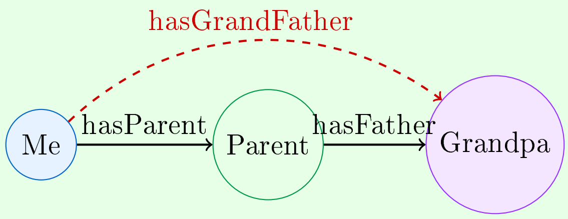
Property Chain: Defines a new property as a sequence of other properties.
If p(x,y) AND q(y,z) then r(x,z)

EXAMPLE

Example: hasGrandFather

```
ObjectProperty: hasGrandFather
    SubPropertyChain: hasParent o hasFather
```

Meaning: hasParent followed by hasFather = hasGrandFather



Note: This is different from `Transitive`! Transitive uses the SAME property. Chain uses DIFFERENT properties.

10.7 Disjoint Properties

DEFINITION

Disjoint Properties: Two properties P and Q can NEVER hold for the same pair of individuals.
NOT(P(x,y) AND Q(x,y))

EXAMPLE

Example: hasChild and hasSpouse

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

Meaning: You cannot be both someone's child AND their spouse!

- hasChild(John, Mary) AND hasSpouse(John, Mary) → INCONSISTENCY!
- hasChild(John, Mary) AND hasSpouse(John, Anna) → OK (different pairs)

11 Quick Reference: All OWL Features

Table 12: OWL Features Summary (EXAM CHEAT SHEET)

lightblue **Feature**	**Syntax**	**Meaning**
	Class Constructors	
Intersection	A and B	Both A AND B
Union	A or B	A OR B (or both)
Complement	not A	NOT A
Enumeration	{a, b, c}	Exactly these individuals
	Property Restrictions	
Existential	P some C	At least one P-value in C
Universal	P only C	All P-values (if any) in C
HasValue	P value x	P points to specific individual x
Min Cardinality	P min n C	At least n P-values in C
Max Cardinality	P max n C	At most n P-values in C
Exact Cardinality	P exactly n C	Exactly n P-values in C
	Property Characteristics	
Functional	Functional	Max 1 value per subject
InverseFunctional	InverseFunctional	Max 1 subject per value
Symmetric	Symmetric	P(x,y) ⇒ P(y,x)
Asymmetric	Asymmetric	P(x,y) ⇒ NOT P(y,x)
Transitive	Transitive	P(x,y) ∧ P(y,z) ⇒ P(x,z)
Reflexive	Reflexive	P(x,x) always true
Irreflexive	Irreflexive	P(x,x) never true
	Individual Assertions	
Same As	SameAs: x	Two names, one individual
Different From	DifferentFrom: x	Different individuals

12 Lab Practice Exercises

12.1 Exercise 1: Japanese Students

EXAMPLE

Task: Create a class for Japanese students.
Step 1: Create AsianCountry class

```
Class: AsianCountry
    SubClassOf: NonEUCountry
    DisjointWith: NorthAmericanCountry
```

Step 2: Add Japan as an instance

```
Individual: Japan
    Types: AsianCountry
```

Step 3: Define JapaneseStudent

```
Class: JapaneseStudent
    EquivalentTo: Student and (residentOf value Japan)
```

Step 4: Add a test individual

```
Individual: Mia
    Types: Person
    Facts: studies Programming,
           residentOf Japan
```

Result: After running the reasoner, Mia should be classified as a JapaneseStudent!

12.2 Exercise 2: Art Students (Closure Axiom)

EXAMPLE

Task: Create a class for students who ONLY study art courses.
Step 1: Create ArtCourse and some instances

```
Class: ArtCourse
    SubClassOf: Course

Individual: Dance
    Types: ArtCourse

Individual: EnglishLiterature
    Types: ArtCourse
```

Step 2: Define ArtStudent with closure axiom

```
Class: ArtStudent
    EquivalentTo: Student and
                  (studies some ArtCourse) and
                  (studies only ArtCourse)
```

Step 3: Create test individual Jim

```
Individual: Jim
    Types: Person
    Facts: studies Dance,
           studies EnglishLiterature,
```

```
            studies only {Dance, EnglishLiterature}
```

Important: The "studies only" line closes the world for Jim - he ONLY studies these courses.
Result: Jim should be classified as an ArtStudent.

12.3 Exercise 3: Staff Classification

EXAMPLE

Task: Create Academic and Administrator classes with specific jobs.
Step 1: Create Staff subclasses

```
Class: Academic
    SubClassOf: Staff
    DisjointWith: Administrator

Class: Administrator
    SubClassOf: Staff
    DisjointWith: Academic
```

Step 2: Create Job instances

```
Individual: Professor
    Types: Job

Individual: AssistantProfessor
    Types: Job

Individual: AdmissionsOfficer
    Types: Job
```

Step 3: Define job restrictions

```
Class: Academic
    EquivalentTo: Staff and
        (employedAs value {Professor, AssistantProfessor,
           AssociateProfessor})

Class: Administrator
    EquivalentTo: Staff and
        (employedAs value {AdmissionsOfficer, FinanceOfficer})
```

Step 4: Test with Jane

```
Individual: Jane
    Facts: employedAs Professor
```

Jane should be classified as an Academic!
Challenge: What happens if Bob is employedAs both Professor AND AdmissionsOfficer?
Answer: INCONSISTENCY! Because Academic and Administrator are disjoint. To fix: remove the disjoint axiom or create a new class for dual-role staff.

13 Exam Practice Questions

13.1 Question 1: Class Expressions

Write OWL Manchester syntax for each:

a) A VegetarianPizza has at least one vegetable topping and no meat toppings

b) A RichPerson has at least 1,000,000 in their bank account

c) A Grandparent is a Person who has a child who is also a Parent

Answers:

```
# a) VegetarianPizza
Class: VegetarianPizza
    EquivalentTo: Pizza and
                  (hasTopping some VegetableTopping) and
                  (hasTopping only (not MeatTopping))

# b) RichPerson
Class: RichPerson
    EquivalentTo: Person and (hasBankBalance some integer[>= 1000000])

# c) Grandparent
Class: Grandparent
    EquivalentTo: Person and (hasChild some Parent)
```

13.2 Question 2: Property Characteristics

For each property, identify the appropriate characteristics:

a) hasSocialSecurityNumber (each person has exactly one, each SSN belongs to exactly one person)

b) isFriendOf (if A is friend of B, then B is friend of A)

c) isPartOf (if A isPartOf B and B isPartOf C, then A isPartOf C)

d) hasChild (no one can be their own child)

Answers:

```
# a) hasSocialSecurityNumber
ObjectProperty: hasSSN
    Characteristics: Functional, InverseFunctional

# b) isFriendOf
ObjectProperty: isFriendOf
    Characteristics: Symmetric

# c) isPartOf
ObjectProperty: isPartOf
    Characteristics: Transitive

# d) hasChild
ObjectProperty: hasChild
    Characteristics: Asymmetric, Irreflexive
```

13.3 Question 3: Inference

Given the following ontology, what can the reasoner infer about John?

```
Class: Father
    EquivalentTo: Man and (hasChild some Person)

Class: Parent
    EquivalentTo: Father or Mother

Individual: John
    Types: Man
    Facts: hasChild Mary

Individual: Mary
    Types: Person
```

Answer:

1. John is a Man (stated)
2. John hasChild Mary who is a Person (stated)
3. Therefore: John is a Father (by EquivalentTo definition)
4. Therefore: John is a Parent (because Father or Mother = Parent)

14 Common Exam Mistakes to Avoid

WARNING / COMMON MISTAKE

Mistake 1: Confusing `some` and `only`

- `some` = "at least one exists"
- `only` = "if any exist, they must be of this type"

Wrong: "eatsOnlyVegetables" $\neq$ `eats only Vegetable`
The `only` version allows someone who eats NOTHING!
Correct: `eats some Vegetable` AND `eats only Vegetable`

WARNING / COMMON MISTAKE

Mistake 2: Forgetting the Open World Assumption
OWL assumes we don't know everything. Just because something isn't stated doesn't mean it's false!
To close the world for an individual, use closure axioms or enumerate all values.

WARNING / COMMON MISTAKE

Mistake 3: Not declaring individuals as different
For cardinality to work correctly, you must declare individuals as `DifferentIndividuals`.
Otherwise, the reasoner might think "Mary" and "Tom" are the same person!

WARNING / COMMON MISTAKE

Mistake 4: Using SubClassOf when EquivalentTo is needed
SubClassOf gives necessary conditions only (one-way inference).
EquivalentTo gives necessary AND sufficient conditions (two-way inference = automatic classification).

15 Final Exam Checklist

- ☐ I understand the difference between Propositional and First Order Logic
- ☐ I know why Description Logic is used instead of full FOL (decidability!)
- ☐ I can decode DL language names (SHOIN, SHOIQ, etc.)
- ☐ I understand OWL versions (Full, DL, Lite) and their trade-offs
- ☐ I can write class intersections, unions, and complements
- ☐ I know the difference between SubClassOf and EquivalentTo
- ☐ I understand `some` vs `only` and when to use closure axioms
- ☐ I can write cardinality restrictions (min, max, exactly) - qualified and unqualified
- ☐ I know all property characteristics (functional, symmetric, transitive, etc.)
- ☐ I understand owl:sameAs and owl:differentFrom
- ☐ I remember the Open World Assumption and how to close it
- ☐ I can trace through what a reasoner would infer

Good luck on your exam!
Remember: Practice with Protégé to reinforce these concepts.