OWL DL

Second Part

Valentina Bartalesi Lenzi

Istituto di Scienza e Tecnologie dell'Informazione - Consiglio Nazionale delle Ricerche – Pisa

A.A. 2024-2025

OWL 2 DL Ontologies

OWL 2 DL ontologies consist of three different syntactic categories:

1. **Entities**: classes, properties, and individuals, identified by IRIs. They form the primitive terms and the basic elements of an ontology
2. **Expressions**: complex notions capturing the intensions of classes and properties
3. **Axioms**: statements that are asserted to be true

OWL 2 DL Ontologies

OWL 2 DL ontologies consist of three different syntactic categories:

1. **Entities**: classes, properties, and individuals, identified by IRIs. They form the primitive terms and the basic elements of an ontology
2. Expressions: complex notions capturing the intensions of classes and properties
3. Axioms: statements that are asserted to be true

Entities

Entities are the terms that are identified by IRIs. They are:

- **Classes**, including two built-in classes:
  - owl:Thing, the class of all individuals
  - owl:Nothing, the empty class

- **Properties**, which are divided into Object Properties, Data Properties and Annotation properties
  1. **Object Properties** relate two individuals, and include:
     - owl:topObjectProperty, which connects all possible pairs of individuals
     - owl:bottomObjectProperty, which connects no pair of individuals
  2. **Data Properties** connect individuals with literals, and include:
     - owl:topDataProperty, which connects all possible individuals with all literals
     - owl:bottomDataProperty, which connects no individual and no literal
  3. **Annotation Properties** are used to make annotations for an ontology, axiom, or an IRI

Entities

- **Named Individuals**, representing objects from the domain

- **Datatypes**, each of which must be:
  - rdfs:Literal
  - a datatype in the OWL datatype map
  - a datatype defined by means of a datatype definition

In addition to entities, OWL 2 ontologies include:

- literals
- anonymous individuals, analogous to blank nodes in RDF
- datatype facets, *i.e.*, pairs (F, lt), where F is a facet of a datatype, and lt is a literal of the appropriate datatype

Vocabulary

Entities, literals, anonymous individuals and datatype facets form the <u>vocabulary</u> of an OWL DL ontology

Formally, a vocabulary V over a datatype map D is a 7-tuple
$\mathbf{V = (V_C, V_{OP}, V_{DP}, V_I, V_{DT}, V_{LT}, V_{FA})}$, where:

- $\mathbf{V_C}$ is a **set of classes** containing at least the classes
  - owl:Thing, the class of all individuals
  - owl:Nothing, the empty class
- $\mathbf{V_{OP}}$ is a **set of object properties** containing at least
  - owl:topObjectProperty, which connects all pairs of individuals
  - owl:bottomObjectProperty, which connects no pair of individuals
- $\mathbf{V_{DP}}$ is a **set of data properties** containing at least
  - owl:topDataProperty, which connects all individuals with all literals
  - owl:bottomDataProperty, which connects no individual and no literal
- $\mathbf{V_I}$ is a set of individuals (named and anonymous)

Vocabulary

- $\mathbf{V_{DT}}$ is a set containing all **datatypes** of D, the datatype rdfs:Literal, and possibly other datatypes; that is, $N_{D'} \cup$ {rdfs:Literal} $\subseteq V_{DT}$ where N_{DT} is the set of datatypes that does not contain the datatype rdfs:Literal

- $\mathbf{V_{LT}}$ is a set of **literals** LV^^DT for each **datatype** DT $\in N_{DT}$ and each lexical form LV $\in N_{LS}$(DT) where N_{LS} is the lexical space

- $\mathbf{V_{FA}}$ is the set of **pairs (F, lt)** for each constraining facet F, datatype DT $\in N_{DT}$, and literal lt $\in V_{LT}$ such that (F, (LV, $DT_1)^{LS}$) $\in N_{FS}$(DT), where LV is the lexical form of lt and DT_1 is the datatype of lt and N_{FS} is the facet space

The direct semantics of OWL is based on the notion of **interpretation**

Similarly to RDF interpretations, **OWL interpretations are adaptations of standard first-order logic interpretations over the vocabulary of OWL**

Interpretations

An interpretation I for a datatype map D and a vocabulary V is a 10-tuple
$I = (\Delta_I , \Delta_D, \cdot^{C}, \cdot^{OP}, \cdot^{DP}, \cdot^{I}, \cdot^{DT}, \cdot^{LT}, \cdot^{FA}, NAMED)$,
where:

- Δ_I is a nonempty set called the object domain
  → *Δ_I contains the individuals (named and anonymous)*
- Δ_D, is a nonempty set disjoint with Δ_I called the data domain such that $(DT)^{DT} \subseteq \Delta_D$, for each datatype DT in V_{DT}
  → *Δ_D contains all possible values of the datatypes of I*
- ^{C} is the class interpretation function that assigns to each class C in V_C a subset $(C)^C \subseteq \Delta_I$ such that:
  $(owl{:}Thing)^C = \Delta_I$
  $(owl{:}Nothing)^C = \emptyset$
  → *this function assigns to each class the individuals that are instances of it*

- ^{OP} is the **object property interpretation function** that assigns to each object property OP in V_{OP} a subset $(OP)^{OP} \subseteq \Delta_I \times \Delta_I$ such that:
  $(owl{:}topObjectProperty)^{OP} = \Delta_I \times \Delta_I$
  $(owl{:}bottomObjectProperty)^{OP} = \emptyset$
  → *this function assigns to each object property the pairs of individuals related by it*
- ^{DP} is the **data property interpretation function** that assigns to each data property DP in V_{DP} a subset $(DP)^{DP} \subseteq \Delta_I \times \Delta_D$ such that: $(owl{:}topDataProperty)^{DP} = \Delta_I \times \Delta_D$
  $(owl{:}bottomDataProperty)^{DP} = \emptyset$
  → *this function assigns to each data property the pairs (individual, datatype value) related by it*
- ^{I} is the **individual interpretation function** that assigns to each individual a in V_I an element $(a)^I$ in Δ_I
  → *this function assigns to each vocabulary individual an individual of the world*

- ^{DT} is the **datatype interpretation function** that assigns to each datatype DT in V_{DT} a subset $(DT)^{DT} \subseteq \Delta_D$ such that:
  DT is the same as in D for each datatype DT in N_{DT}
  $(\text{rdfs:Literal})^{DT} = \Delta_D$
  → *for each datatype this function assigns all its possible values*
- ^{LT} is the **literal interpretation function** that is defined as $(lt)^{LT} = (LV, DT)^{LS}$ for each lt in V_{LT}, where LV is the lexical form of lt and DT is the datatype of lt
  → *this function assigns to the literals of the vocabulary their value starting from their lexical forms and their datatypes*
- ^{FA} is the **facet interpretation function** that is defined as $(F , lt)^{FA} = (F, (lt)^{LT})^{FS}$ for each $(F, lt) \in V_{FA}$.
  → *this function assigns to each pair (facet, literal) a value in the Facet space*
- NAMED is a subset of Δ_I such that $(a)^I$ in **NAMED for each named**
  → *this is the set of named individuals*

Example

Classes: Student, Exam, Session, CourseEdition
Properties: takes, isExamOf, isSessionOf, isRegisteredTo
Individuals: marco, maria, exam 6/12/2022, session 12/2022, course edition 2022

Axioms:
- A session is session of exactly one course edition

Vocabulary:
V_C = {:Student, :Exam, :Session, :CourseEdition}
V_{OP} = {:takes, :isExamOf, :isSessionOf, :isRegisteredTo}
$V_{DP} = \emptyset$
V_I = {:marco, :maria, :exam6_12_2022, :session12_2022, :course_edition_2022}
V_{DT} contains all datatypes of the datatype map D
V_{LT} contains all possible literals for each datatype of D
V_{FA} contains all facets defined on the datatypes of D

Interpretation:

Δ_I = {marco, maria, exam 6/12/2022, session 12/2022, course edition 2022}

Δ_D = {true, false, 1,0….} all possible values for all datatypes in the datatype map D

$Student^C$ = {marco, maria}
$Exam^C$ = {exam 6/12/2022}
$Session^C$ = {session 12/2022}
$CourseEdition^C$ = {course edition 2022}

$takes^{OP}$ = {(marco, exam 6/12/2022), (maria, exam 6/12/2022)}
$isExamOf^{OP}$ = {(exam 6/12/2022, session 12/2022)}
$isSessionOf^{OP}$ = {(session 12/2022, course edition 2022)}
$isRegisteredTo^{OP}$ = {(marco, course edition 2022), (maria, course edition 2022)}

$(:marco)^I$ = marco
$(:maria)^I$ = maria
$(:exam6\_12\_2022)^I$ = exam 6/12/2022
$(:session12\_2022)^I$ = session 12/2022
$(:course\_edition\_2022)^I$ = course edition 2022

SubClassOf(:Session ObjectExactCardinality(1 :isSessionOf :CourseEdition))
Session ⊆ 1 isSessionOf CourseEdition
{session 12/2022} ⊆ {session 12/2022}

OWL 2 DL Ontologies

OWL 2 DL ontologies consist of three different syntactic categories:

1. Entities: classes, properties, and individuals, identified by IRIs. They form the primitive terms and the basic elements of an ontology
2. **Expressions**: complex notions capturing the intensions of classes and properties
3. Axioms: statements that are asserted to be true

Property Expressions

Properties can be used in OWL 2 to form **property expressions**

- From a semantical point of view, **property expressions are like properties**, in that they are relationship types, used in statements to connect pairs of individuals

- From a syntactical point of view, property expressions are complex terms, formed by applying **constructors** to properties

In **OWL** there is just **one type of property expression**: **Inverse Property**, which applies **only** to **object properties**

Example:
- Suppose you have a property **childOf**, whose intension is “pairs of people such that the first is a direct descendant of the second”
- Then the intension of the **inverse property** of childOf (parentOf) is “pairs of people such that the first is parent of the second”
- In some situations, one may want to use childOf, in other situations the inverse may be more convenient

Inverse Object Property

Construct Name	object property inverse
Construct Type	object property expression
Functional Syntax	ObjectInverseOf (OP)
Description	the property having as extension the inverse of the extension of property OP
Semantics	$\{(y, x) \mid (x, y) \in (\mathrm{OP})^{OP}\}$
RDF Turtle Syntax	_:x owl:inverseOf OP .
Example	ObjectInverseOf(childOf)

Datatypes Expression

Datatypes can be used in OWL 2 to form **data ranges**, which are one kind of expressions constructed with set-theoretic operators

From the semantical point of view, **data ranges are like datatypes**, i.e. **sets of data values used as ranges of data properties**

From a syntactical point of view, data ranges are complex terms, formed by applying **set theoretic constructors** to datatypes

The simplest kind of data ranges are datatypes themselves

Complex data ranges are formed via:

- **intersection**
- **union**
- **complement**
- **enumeration**
- **restriction**

of datatypes

Data Range Intersection

Construct Name	data range intersection
Construct Type	datatype expression
Functional Syntax	DataIntersectionOf($DR_1 \ldots DR_n$) $n \geq 2$
Description	the intersection of $DR_1 \ldots DR_n$
Semantics	$(DR_1)^{DT} \cap \ldots \cap (DR_n)^{DT}$
RDF Turtle Syntax	_:x rdf:type rdfs:Datatype . _:x owl:intersectionOf ($DR_1 \ldots DR_n$) .
Example	DataIntersectionOf(xsd:nonNegativeInteger xsd:nonPositiveInteger)

Data Range Union

Construct Name	data range union
Construct Type	datatype expression
Functional Syntax	DataUnionOf($\mathrm{DR}_1 \ldots \mathrm{DR}_n$) $n \geq 2$
Description	the union of $\mathrm{DR}_1 \ldots \mathrm{DR}_n$
Semantics	$(\mathrm{DR}_1)^{DT} \cup \ldots \cup (\mathrm{DR}_n)^{DT}$
RDF Turtle Syntax	_:x rdf:type rdfs:Datatype . _:x owl:unionOf ($\mathrm{DR}_1 \ldots \mathrm{DR}_n$) .
Example	DataUnionOf(xsd:string xsd:integer)

Data Range Complement

Construct Name	data range complement
Construct Type	datatype expression
Functional Syntax	DataComplementOf(DR)
Description	the complement of DR
Semantics	$\Delta_D \setminus (\mathrm{DR})^{DT}$
RDF Turtle Syntax	_:x rdf:type rdfs:Datatype . _:x owl:datatypeComplementOf DR .
Example	DataComplementOf(xsd:positiveInteger)

Literal Enumeration

Construct Name	literal enumeration
Construct Type	datatype expression
Functional Syntax	DataOneOf($v_1 \ldots v_n$) $n \geq 1$
Description	the datatype consisting of the values denoted by $v_1 \ldots v_n$
Semantics	$\{ (v_1)^{LT}, \ldots, (v_n)^{LT} \}$
RDF Turtle Syntax	_:x rdf:type rdfs:Datatype . _:x owl:oneOf (D1 . . . Dn) .
Example	DataOneOf("tom" "jerry"^^xsd:string)

Data Range Restriction

Construct Name	data range restriction
Construct Type	datatype expression
Functional Syntax	DatatypeRestriction(DN f_1 v_1 $\ldots f_n$ v_n) $n \geq 1$
Description	the intersection of the value space of DN with the set denoted by (f_1 v_1) and ...and the set denoted by (f_n v_n)
Semantics	$(DN)^{DT} \cap (f_1\ v_1)^{FA} \cap \ldots \cap (f_n\ v_n)^{FA}$
RDF Turtle Syntax	_:x rdf:type rdfs:Datatype . _:x owl:onDatatype DN . _:x owl:withRestrictions (_:x1 ... _:xn) . _:xj fj vj . j=1 . . . n

Example

DatatypeRestriction(xsd:integer xsd:minInclusive "5"^^xsd:integer
xsd:maxExclusive "10"^^xsd:integer)

The above data range contains exactly the integers 5, 6, 7, 8, and 9

Class Expression

Class expressions are provided in OWL 2 to represent **class intensions**

We have already seen some class intensions, *e.g.*:

- Book →“resource that is an object, unique, textual”
- Parent → “resource that is a person and has at least one child”

In RDF Schema, all we can do to represent intensions is to identify them as classes (or properties), and indicate some aspect of the intension.
But in so doing, many **potentially interesting inferences are lost**,
e.g., a book has textual content or that every parent has at least a son or a daughter

Class Expression

In OWL we can use class expressions to give a **more accurate description of the intension of a class**, using various kinds of **constructors**, borrowed from logic or set theory, *e.g.*, (in some notation)

- for Book: Object **and at least one** hasContent **that is a** Text
- for Parent: Person **and at least one** hasChild **who is a** Person

Class Expression

From the semantical point of view, class expressions are like classes, *i.e.*, **selections of features (intension) that in every interpretation denote sets of individuals**

From a syntactical point of view, class expressions are **complex terms, formed by applying constructors to classes, properties or expressions**

We can group class expressions in:

- **set-theoretic expressions**
- **object or data property restrictions**
- **object or data property cardinality restrictions**

Set-Theoretic Class Expressions

These include:

- **Intersection**, as in: “a resource that can do something **and** is an individual”

  - ObjectIntersectionOf

- **Union**, as in: “adult person or young person”

  - ObjectUnionOf

- **Complement**, as in: “a resource that is **not** a human”

  - ObjectComplementOf

- **Enumeration**, as in: “a resource that is Monday **or** Tuesday **or** ...**or** Sunday”

  - ObjectOneOf

Intersection

Construct Name	Intersection of Class Expressions
Construct Type	class expression
Functional Syntax	ObjectIntersectionOf($\mathrm{CE}_1 \ldots \mathrm{CE}_n$) $n \geq 2$
Description	a class expression having as intension the logical conjunction of the intensions of class expressions $\mathrm{CE}_1 \ldots \mathrm{CE}_n$
Semantics	$(\mathrm{CE}_1)^C \cap \ldots \cap (\mathrm{CE}n)^C$
RDF Turtle Syntax	_:x rdf:type owl:Class . _:x owl:intersectionOf ($\mathrm{CE}_1 \ldots \mathrm{CE}_n$) .
Example	ObjectIntersectionOf(Agent Person)

Union

Construct Name	Union of Class Expressions
Construct Type	class expression
Functional Syntax	ObjectUnionOf($CE_1 \ldots CE_n$) $n \geq 2$
Description	a class expression having as intension the logical disjunction of the intensions of class expressions $CE_1 \ldots CE_n$
Semantics	$(CE_1)^C \cup \ldots \cup (CEn)^C$
RDF Turtle Syntax	_:x rdf:type owl:Class . _:x owl:unionOf ($CE_1 \ldots CE_n$) .
Example	ObjectUnionOf(Adult Young)

Complement

Construct Name	Complement of Class Expressions
Construct Type	class expression
Functional Syntax	ObjectComplementOf(CE)
Description	a class expression having as intension the logical negation of the intension of class expression CE
Semantics	$\Delta_I \setminus (CE)^C$
RDF Turtle Syntax	_:x rdf:type owl:Class . _:x owl:complementOf CE .
Example	ObjectComplementOf(Male)

Enumeration

Construct Name	Enumeration of Individuals
Construct Type	class expression
Functional Syntax	ObjectOneOf($a_1 \ldots a_n$) $n \geq 2$
Description	a class expression having as extension the resources named by the individuals $a_1 \ldots a_n$
Semantics	$\{(a_1)^I, \ldots, (a_n)^I\}$
RDF Turtle Syntax	_:x rdf:type owl:Class . _:x owl:oneOf ($a_1 \ldots a_n$) .
Example	ObjectOneOf(mon tue wed ... sun)

Object Property Restrictions

Class expressions in OWL 2 can be formed by placing **restrictions on object property expressions**. These include:

- **Existential Quantification**, as in: “a resource that has **at least one** child who is a male”

  - ObjectSomeValuesFrom
- **Universal Quantification**, as in: “a resource that has **all** authors who are Italian”

  - ObjectAllValuesFrom
- **Individual Value Restriction**, as in “a resource that is born in **Italy**”

  - ObjectHasValue
- **Self Restriction**, as in: “a resource that is **self**-employed”

  - ObjectHasSelf

Existential Quantification

Construct Name	Existential Quantification
Construct Type	class expression
Functional Syntax	ObjectSomeValuesFrom(OPE CE)
Description	a class expression having as extension the resources who are connected by object property expression OPE to resources that are instances of class expression CE
Semantics	$\{x \mid \exists y : (x, y) \in (\mathrm{OPE})^{OP} \text{ and } y \in (\mathrm{CE})^{C}\}$
RDF Turtle Syntax	_:x rdf:type owl:Restriction . _:x owl:onProperty OPE . _:x owl:someValuesFrom CE .
Example	ObjectSomeValuesFrom(child Male)

Universal Quantification

Construct Name	Universal Quantification
Construct Type	class expression
Functional Syntax	ObjectAllValuesFrom(OPE CE)
Description	a class expression having as extension the resources who are connected by object property expression OPE to no resources at all or only to resources that are instances of class expression CE
Semantics	$\{x \mid \forall y : (x, y) \in (\mathrm{OPE})^{OP} \text{ implies } y \in (\mathrm{CE})^{C}\}$
RDF Turtle Syntax	_:x rdf:type owl:Restriction . _:x owl:onProperty OPE . _:x owl:allValuesFrom CE .
Example	ObjectAllValuesFrom(hasAuthor Italian)

Individual Value Restriction

Construct Name	Individual Value Restriction
Construct Type	class expression
Functional Syntax	ObjectHasValue(OPE a)
Description	a class expression having as extension the resources who are connected by object property expression OPE to the individual a
Semantics	$\{x \mid (x, (a)^I) \in (\text{OPE})^{OP}\}$
RDF Turtle Syntax	_:x rdf:type owl:Restriction . _:x owl:onProperty OPE . _:x owl:hasValue a .
Example	ObjectHasValue(bornIn Italy)

Self-Restriction

Construct Name	Self-Restriction
Construct Type	class expression
Functional Syntax	ObjectHasSelf(OPE)
Description	a class expression having as extension the resources who are connected by object property expression OPE to themselves
Semantics	$\{x \mid (x, x) \in (\mathrm{OPE})^{OP}\}$
RDF Turtle Syntax	_:x rdf:type owl:Restriction . _:x owl:onProperty OPE . _:x owl:hasSelf "true"^^xsd:boolean .
Example	ObjectHasSelf(isEmployedBy)

Data Property Restrictions

Class expressions in OWL 2 can be formed by placing **restrictions on data property expressions**, similarly to the restrictions on object property expressions. But there are two **differences**:

1. the only data property expressions allowed in OWL are **data property themselves**, *i.e.*, no data property inverse, so the restrictions in this case are simpler
2. the restriction is defined over a **data range** instead of a class expression

Data property restrictions include:

- **Existential Quantification**, as in: “a resource that is **at most** 18 years old”
- **Universal Quantification**, as in: “a resource that has **all** IDs as integers”
- **Literal Value Restriction**, as in “a resource that is **17** years old”

Existential Quantification

Construct Name	Existential Quantification
Construct Type	class expression
Functional Syntax	DataSomeValuesFrom(DPE DR)
Description	a class expression having as extension the resources who are connected by data property expression DPE to literals that are instances of data range DR
Semantics	$\{x \mid \exists y : (x, y) \in (\mathrm{DPE})^{DP} \text{ and } y \in (\mathrm{DR})^{DT}\}$
RDF Turtle Syntax	_:x rdf:type owl:Restriction . _:x owl:onProperty DPE . _:x owl:someValuesFrom DR .
Example	

DataSomeValuesFrom(hasAge
DatatypeRestriction(xsd:integer xsd:maxInclusive "18"^^xsd:integer))

Universal Quantification

Construct Name	Universal Quantification
Construct Type	class expression
Functional Syntax	DataAllValuesFrom(DPE DR)
Description	a class expression having as extension the resources who are connected by data property expression DPE to no literals at all or only to literals that are instances of data range DR
Semantics	$\{x \mid \forall y : (x, y) \in (\mathrm{DPE})^{OP} \text{ implies } y \in (\mathrm{DR})^{DT}\}$
RDF Turtle Syntax	_:x rdf:type owl:Restriction . _:x owl:onProperty DPE . _:x owl:allValuesFrom DR .
Example	DataAllValuesFrom(hasID xsd:integer)

Literal Value Restriction

Construct Name	Literal Value Restriction
Construct Type	class expression
Functional Syntax	DataHasValue(DPE lt)
Description	a class expression having as extension the resources who are connected by data property expression DPE to the literal lt
Semantics	$\{x \mid (x, (\mathrm{lt})^{LT}) \in (\mathrm{DPE})^{DP}\}$
RDF Turtle Syntax	_:x rdf:type owl:Restriction . _:x owl:onProperty DPE . _:x owl:hasValue lt .
Example	DataHasValue(hasAge "17"^^xsd:integer)

Object Property Cardinality Restrictions

Class expressions in OWL 2 can be formed by **placing restrictions on the cardinality of object property expressions**, that is on the **number of relationships of the same type that an individual may have**

Object property cardinality restrictions include:

- **Minimal cardinality**, as in: “a resource that has **at least two** authors who are Italian”
- **Maximal cardinality**, as in: “a resource that has **at most one** child who is male”
- **Exact cardinality**, as in “a resource that has **exactly two** members who are self-employed”

Minimal Cardinality

Construct Name	Obj. Prop. Minimum Cardinality Restriction
Construct Type	class expression
Functional Syntax	ObjectMinCardinality(n OPE CE)
Description	a class expression having as extension the resources who have at least n connections of object property expression OPE to instances of class expression CE
Semantics	$\{x \mid \#\{y \mid (x, y) \in (\mathrm{OPE})^{OP},\ y \in (\mathrm{CE})^{C}\} \geq n\}$
RDF Turtle Syntax	_:x rdf:type owl:Restriction . _:x owl:onProperty OPE . _:x owl:minQualifiedCardinality n . _:x owl:onClass CE .
Example	ObjectMinCardinality(2 hasAuthor Italian)

Maximum Cardinality

Construct Name	Obj. Prop. Maximum Cardinality Restriction
Construct Type	class expression
Functional Syntax	ObjectMaxCardinality(n OPE CE)
Description	a class expression having as extension the resources who have at most n connections of object property expression OPE to instances of class expression CE
Semantics	$\{x \mid \#\{y \mid (x, y) \in (\mathrm{OPE})^{OP},\ y \in (\mathrm{CE})^{C}\} \leq n\}$
RDF Turtle Syntax	_:x rdf:type owl:Restriction . _:x owl:onProperty OPE . _:x owl:maxQualifiedCardinality n . _:x owl:onClass CE .
Example	ObjectMaxCardinality(1 Inverse(hasParent) Male)

Exact Cardinality

Construct Name	Obj. Prop. Exact Cardinality Restriction
Construct Type	class expression
Functional Syntax	ObjectExactCardinality(n OPE CE)
Description	a class expression having as extension the resources who have exactly n connections of object property expression OPE to instances of class expression CE
Semantics	$\{x \mid \#\{y \vert (x, y) \in (\mathrm{OPE})^{OP},\ y \in (\mathrm{CE})^{C}\} = n\}$
RDF Turtle Syntax	_:x rdf:type owl:Restriction . _:x owl:onProperty OPE . _:x owl:qualifiedCardinality n . _:x owl:onClass CE .
Example	ObjectExactCardinality(2 HasMember ObjectHasSelf(isEmployedBy))

Data Property Cardinality Restrictions

Class expressions in OWL 2 can be formed by placing **restrictions on the cardinality of data property expressions**, analogous to those on object property expressions, with two differences:

1. the only data property expressions allowed in OWL **are data properties**
2. the restriction is defined **over** a **data range** instead of a class expression

Data property cardinality restrictions include:

- **Minimal cardinality**, as in: “a resource that has **at least two** phone numbers that are integers”
- **Maximal cardinality**, as in: “a resource that has **at most one** fiscal code which is a string”
- **Exact cardinality**, as in “resource that has **exactly one** birthdate that is a date”

Data Property Minimal Cardinality

Construct Name	Data Prop. Minimum Cardinality Restriction
Construct Type	class expression
Functional Syntax	DataMinCardinality(n DPE DR)
Description	a class expression having as extension the resources who have at least n connections of data property expression DPE to instances of data range DR
Semantics	$\{x \mid \#\{y \mid (x,y) \in (\text{DPE})^{DP},\ y \in (\text{DR})^{DT}\} \geq n\}$
RDF Turtle Syntax	_:x rdf:type owl:Restriction . _:x owl:onProperty DPE . _:x owl:minQualifiedCardinality n . _:x owl:onDataRange DR .
Example	DataMinCardinality(2 hasPhoneNum xsd:Integer)

Data Property Maximal Cardinality

Construct Name	Data Prop. Maximum Cardinality Restriction
Construct Type	class expression
Functional Syntax	DataMaxCardinality(n DPE DR)
Description	a class expression having as extension the resources who have at most n connections of data property expression DPE to instances of data range DR
Semantics	$\{x \mid \#\{y \mid (x, y) \in (\mathrm{DPE})^{DP},\ y \in (\mathrm{DR})^{DT}\} \leq n\}$
RDF Turtle Syntax	_:x rdf:type owl:Restriction . _:x owl:onProperty DPE . _:x owl:maxQualifiedCardinality n . _:x owl:onDataRange DR .
Example	DataMaxCardinality(1 HasFiscalCode xsd:string)

Data Property Exact Cardinality

Construct Name	Data Prop. Exact Cardinality Restriction
Construct Type	class expression
Functional Syntax	DataExactCardinality(n DPE DR)
Description	a class expression having as extension the resources who have exactly n connections of data property expression DPE to instances of data range DR
Semantics	$\{x \mid \#\{y \mid (x,y) \in (\text{DPE})^{DP},\ y \in (\text{DR})^{DT}\} = n\}$
RDF Turtle Syntax	_:x rdf:type owl:Restriction . _:x owl:onProperty DPE . _:x owl:qualifiedCardinality n . _:x owl:onDataRange DR .
Example	DataExactCardinality (1 birthDate xsd:date)