Knowledge Representation: Ontologies in the Semantic Web

Wojciech Jaworski

Institute of Informatics
University of Warsaw

Semantic Web

- An initiative for creating and propagating common standards for web content description in a way that allows for the machines and programs to process information accordingly to its meaning.
- One of the Semantic Web standards is OWL for ontology description.
- OWL is an extension of RDF which in turn makes use of XML.
- Moreover, Semantic Web provides:
  - Rule-based languages of higher expressive power than OWL and RDF;
  - Descriptive logic (subset of FOL) for defining the semantics of RDF, OWL and rule-based languages as well as inferences.
  - Resources - ontologies and tools to process them.

Resource Description Framework (RDF)

- Language used to represent information about web resources, particularly to describe metadata.
  - A resource is anything with identity, both a physical entity and a downloadable digital entity; the counterpart of an individual
  - Metadata are data about data, eg. author, text title, etc.

```
<?xml version= 1.0"?>
<rdf:RDF
    xmlns:rdf='http://www.w3.org/1999/02/22-rdf-syntax-ns#'
    xmlns:dc='http://purl.org/dc/elements/1.1/'>
  <rdf:Description rdf:about='http://www.cat.com/docs#R20301'>
    <dc:creator rdf:resource='http://www.cat.com/auth#R051156'/>
    <dc:title>Karin Homepage</dc:title>
    <dc:date>2021-01-20</dc:date>
  </rdf:Description>

</rdf:RDF>
```

- RDF expresses a conjunction of binary relations.
- The first argument is called a subject and the second - an object.
- The domain of the first argument is the *domain*, the domain of the second - the *range*

Syntax of symbols in RDF

```
<?xml version='1.0'?>
<rdf:RDF
    xmlns:rdf='http://www.w3.org/1999/02/22-rdf-syntax-ns#'
    xmlns:dc='http://purl.org/dc/elements/1.1/'>
  <rdf:Description rdf:about='http://www.cat.com/docs#R20301'>
    <dc:creator rdf:resource='http://www.cat.com/auth#R051156'/>
    <dc:title>Karin Homepage</dc:title>
    <dc:date>2021-01-20</dc:date>
  </rdf:Description>
</rdf:RDF>
```

- The constants and names of relations are identifiers of URI resources (=URL+URN).
- Uniform Resource Identifier (URI): identifies a resource.
- URI Reference (URIref): URI with an optional fragment of an identifier.
- Namespace: A space of names identified by URIref.
- Qualified Name (QName): identifies a name in the namespace, has the form *Namespace*:*Name*, expands to an URIref identifying a *Namespace* concatenated with a *Name*.
- The definitions of the namespace occur at the beginning of the document and have the form *Namespace URI*

Main namespaces

```
rdf   http://www.w3.org/1999/02/22-rdf-syntax-ns#
rdfs  http://www.w3.org/2000/01/rdf-schema#
daml  http://www.daml.org/2001/03/daml+oil#
owl   http://www.w3.org/2002/07/owl#
dc    http://purl.org/dc/elements/1.1/
skos  http://www.w3.org/2009/08/skos-reference/skos.r
```

Designation in RDF

- For the use in World Wide Web the following semantics for symbols has been introduced:
  - The model for constants interpretation is the Internet;
  - There is a fixed canonical interpretation of constants in the model (an algorithm of finding the resource for the given address). In other words the name determines the designation.
- This convention enables RDF to define an object through indicating it on the web.
- Natural languages have a similar capability: They allow to indicate particular objects in the world.
- Logics whose semantics is based on the model theory don't have this property, although surrogates are a way to simulate it.
- RDF can refer to itself - one can write a logical formula treating about itself.
- RDF is its own metalanguage, so some definitions are covert.

The syntax of RDF/XML

```
<rdf:RDF>
  <rdf:Description rdf:about=subject1>
    <predicate1 rdf:resource=object1>
    <predicate2>object2</predicate2>
  </rdf:Description>
  <class1 rdf:ID=subject2>
    <predicate3 rdf:datatype=datatype1>object3</predicate3>
  </class1>
</rdf:RDF>
```

- $subject_1$, $object_1$, $subject_2$, $datatype_1$ are URIrefs and surrogates for particular resources.
- $predicate_1$, $predicate_2$, $class_1$, $predicate_3$ are QNames and surrogates for predicates and types.
- $object_2$ is a (text) resource defined by its content.
- $object_3$ is a resource of the type $datatype_1$ defined by its content.

The semantics of RDF/XML

```
<rdf:RDF>
  <rdf:Description rdf:about=subject1>
    <predicate1 rdf:resource=object1>
    <predicate2>object2</predicate2>
  </rdf:Description>
  <class1 rdf:ID=subject2>
    <predicate3 rdf:datatype=datatype1>object3</predicate3>
  </class1>
</rdf:RDF>
```

$$predicate_1(subject_1, object_1) \wedge predicate_2(subject_1, object_2) \wedge$$

$$\texttt{rdf:type}(subject_2, class_1) \wedge (\exists x) predicate_3(subject_2, x) \wedge$$

$$\texttt{rdf:datatype}(x, datatype_1) \wedge \texttt{rdf:value}(x, object_3)$$

Types - elements of `rdfs:Class`

- `rdfs:Resource` the most general type.
  - `rdfs:Class` the type containing all the types.
    - ★ `rdfs:Datatype` the type containing all types of data.
  - `rdfs:Literal` strings of letters or digits.
    - ★ `rdf:PlainLiteral` a literal of no fixed type, an element of `rdfs:Datatype`.
    - ★ `rdf:XMLLiteral` a literal whose value is XML, an element of `rdfs:Datatype`.
  - `rdfs:Container` a collection
    - ★ `rdf:Bag` unordered collections.
    - ★ `rdf:Seq` ordered collections.
    - ★ `rdf:Alt` collections of alternative resources.
  - `rdf:List` lists.
  - `rdf:Property` relations (properties, attributes, features).
    - ★ `rdf:ContainerMembershipProperty` the relations of being a member of a collection.
  - `rdf:Statement` the type that contains all the formulas of the RDF language.

Attributes - elements of `rdf:Property`

- `rdf:about` indicates the subject in a RDF formula by giving its URIref.
- `rdf:ID` indicates the subject in a RDF formula, it is the URIref created from the base URI, symbol „#” and the value `rdf:ID`.
- `rdf:resource` indicates the object in a RDF formula by giving its URIref.
- `rdf:datatype` has the values of an URIref defining the type of data, eg. a string that is a value of an attribute.

These attributes are not defined in the namespace `rdf`.

Attributes - elements of `rdf:Property`

- `rdf:type` $\subseteq$ `rdfs:Resource`$\times$`rdfs:Class`
  determines the type of the subject.
- `rdfs:subClassOf` $\subseteq$ `rdfs:Class`$\times$`rdfs:Class`
  the relation of being a subclass.
- `rdfs:subPropertyOf` $\subseteq$ `rdf:Property`$\times$`rdf:Property`
  the relation of being a subrelation.
- `rdfs:comment` $\subseteq$ `rdfs:Resource`$\times$`rdfs:Literal`
  a description.
- `rdfs:label` $\subseteq$ `rdfs:Resource`$\times$`rdfs:Literal`
  a human-readable name.
- `rdfs:domain` $\subseteq$ `rdf:Property`$\times$`rdfs:Class`
  the domain of the first argument of the relation.
- `rdfs:range` $\subseteq$ `rdf:Property`$\times$`rdfs:Class`
  the domain of the second argument of the relation.
- `rdfs:seeAlso` $\subseteq$ `rdfs:Resource`$\times$`rdfs:Resource`
  additional information about the resource.

Types in OWL

- `owl:Class` the type containing all the ontological categories; it is both an element and a subset of `rdfs:Class`
- `owl:Thing` the type containing all individuals, elt `owl:Class`.
- `owl:Nothing` the empty type, subtype of `owl:Thing`, elt `owl:Class`.
- relations on types taken as sets

`owl:equivalentClass`(A, B)	$A = B$
`rdfs:subClassOf`(A, B)	$A \subseteq B$
`rdf:type`(v, A)	$v \in B$
`owl:disjointWith`(A, B)	$A \cap B = \emptyset$
`owl:intersectionOf`$(A, \{B_1, \ldots, B_n\})$	$A = B_1 \cap \cdots \cap B_n$
`owl:unionOf`$(A, \{B_1, \ldots, B_n\})$	$A = B_1 \cup \cdots \cup B_n$
`owl:complementOf`(A, B)	$A = \texttt{owl:Thing} \setminus B$
`owl:oneOf`$(A, \{v_1, \ldots, v_n\})$	$A = \{v_1, \ldots, v_n\}$

- relations on individuals - elements of `owl:Thing`.

`owl:sameAs`(a, b)	$a = b$
`owl:differentFrom`(a, b)	$a \neq b$

Extension and intension

- Concepts (types, classes) make up an abstract specification, they are not sets of objects.
- Each type is associated with a set *of designates* — of objects that belong to the type.
- The set is called the *denotation* or *extension*.
- Eg. the denotation of the name “cat” is a set of all (past, present and future) cats.
- *Intension* (*meaning*) is a relation between an observable object and a concept.
- Intension is algorithmic in character: for a given object it generates a set of concepts whose extensions the object belongs to.

Extensionality

- Extensionality is an axiom that states that the identity of two extensions implies the identity of the intensions.
- Set theory is extensional: If two sets have the same elements, they are identical.
- The set-theoretical equality $A = B$ says that A and B are the same (or two names for the same).
- Lambda calculus is not extensional unless we add η-reduction

$$(\lambda x)M(x) \xrightarrow{\eta} M$$

- Eg. the definition $A(x) \stackrel{\text{def}}{=} B(x)$, ie. $A = (\lambda x)B(x)$ says that A and the procedure for B are the same entity.
- The addition of η-reduction allows also for $A = (\lambda x)B(x)$ to entail $A = B$.

Non-extensional equivalence

- $(\forall x)A(x) \Leftrightarrow B(x)$ means that A and B are two procedures returning the same output for a given input.
- Eg, let

$$A(n) = \begin{cases} \sum_{k=0}^{n} k & \text{if } n \in \mathbb{N} \\ 0 & \text{O/W} \end{cases}, \text{ and } B(n) = \begin{cases} \frac{n(n+1)}{2} & \text{if } n \in \mathbb{N} \\ 0 & \text{O/W} \end{cases}$$

- A and B are equivalent but not equal because they compute the outcome in different ways.
- Equivalence is represented as $A \approx B$.

Co-extensionality

$$A \approx B \overset{def}{\Longleftrightarrow} (\forall x)A(x) \Leftrightarrow B(x)$$

- In the definition of equivalence there occurs the quantifier $(\forall x)$.
- It symbolizes quantification over all possible entities.
- If we define

$$A(n) = \sum_{k=0}^{n} k \quad \text{and} \quad B(n) = \frac{n(n+1)}{2},$$

  then A and B will not be equivalent.
- But when restricted to a world consisting of only natural numbers, they will have the same extension, which is represented as $A \sim B$.

Theseus' paradox

- The identity of physical entities depends on which properties do we take as relevant.
- By a legend written down by Plutarch:

  The ship wherein Theseus and the youth of Athens returned from Crete had thirty oars, and was preserved by the Athenians down even to the time of Demetrius Phalereus, for they took away the old planks as they decayed, putting in new and stronger timber in their places, in so much that this ship became a standing example among the philosophers, for the logical question of things that grow
- If we replace all the components of some compound object and all the object will be replaced in its totality will it remain the same object?
- The positive answer entails that an exact copy of an object (eg. of a picture) is the very same object.
- The negative answer entails that no objects whose components are being replaced after a time (people in particular) keep their

Theseus' paradox in US Army

- US Army records the distribution of riffles but does not record the distribution of their parts.
- A soldier with a damaged riffle is entitled to collect a spare part with minimum formalities.
- But an unrestricted replacing of parts would make riffle larceny possible through collecting all the parts for a new riffle.
- In order to prevent this misuse a rule has been introduced saying that the identity of a riffle lies in its butt and the butt may not be replaced.

Identity of physical entities

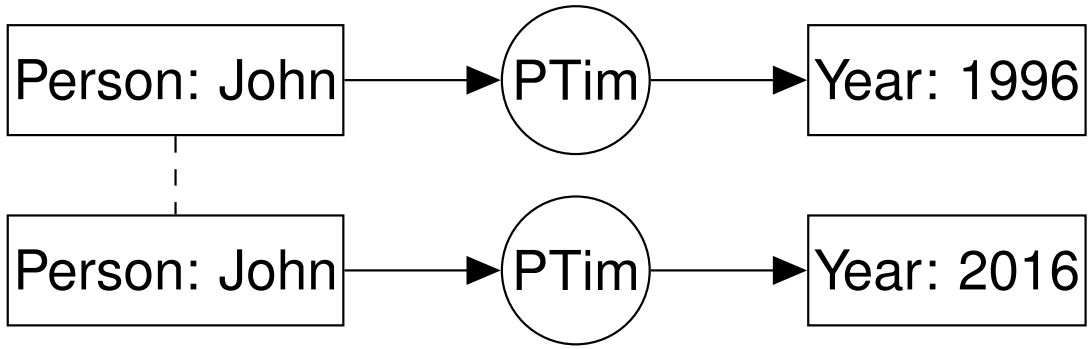
- All physical entities undergo changes in time.
- Is Vistula the same river today as it was 100 year ago?
- Is the contemporary Vistula and the past Vistula from 100 years ago the same object?
- It is made up of different atoms, has a different shape, but it flows in more less the same area and has the same properties.
- From our point of view the change was gradual.
- Is a shattered plate the same object as before?
- It is made up of the same atoms but the configuration of its parts changed and the functionality of the object.
- From our point of view the change was sudden.

Identity of physical entities

- Whether an entity is constant or changing in time depends more on the intension of the observer than the entity alone.
- For a geographer Vistula today and Vistula 100 years ago is the same object, for a chemist they make two separate entities.
- Alike, a plate before and after shuttering is the same from the point of view of waste utilization but not the same as a kitchen equipment.

Complementarity

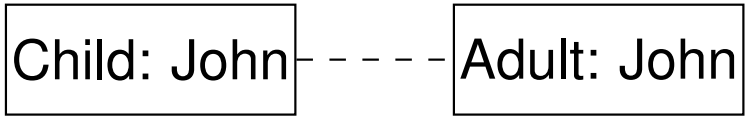
- Identity (equality) is defined in logic as a relation between objects whose all properties are identical.
- 20 years ago John was a child, now he is an adult.
- Is John the same person as 20 year ago?
- From the legal point of view he simultaneously is: it is assumed that no properties have changed (eg. what belonged to him 20 years ago is still his by assumption)



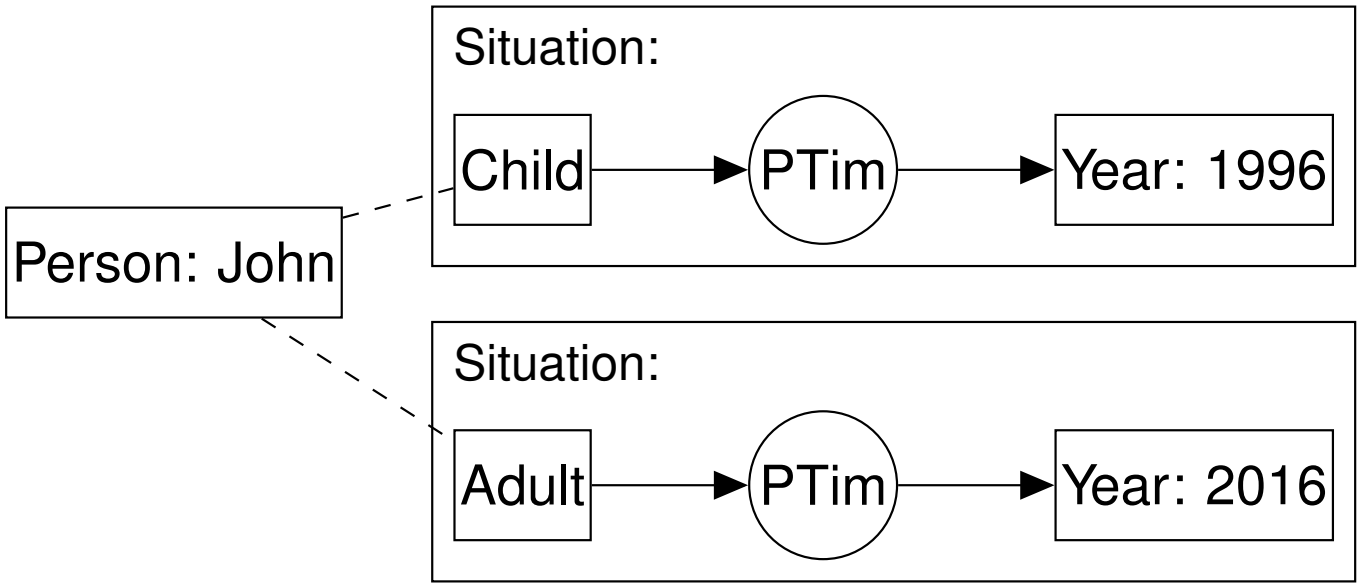
- and also is not: certain properties have changed (eg. he used to be juvenile and how he is of age)
- In particular cases each of these points of view apply.

Complementarity

- But those cases must not be mixed as it leads to a contradiction.
- From the graphs above we might have inferred:



- Which is contradictory to the fact that a child cannot be an adult.
- Complementary ways of describing the world can be mixed if we indicate the context overtly.



Hierarchy of concepts

- Co-extensionality doesn't mean that the concepts are identical, eg. "man" and "two-legged featherless animal" have the same extension but a different meaning (intension), similarly "the morning star" and "the evening star".
- On the other hand, "a unicorn" has an empty extension but we don't want for it to be a subtype of "a tree".
- A hierarchy of concepts is built based on intension inclusion.
- Types will be understood as algorithms that return true or false for an input object.
- Quantification will be executed over all the possible entities (observations).
- Eg. in order to say that a unicorn has a horn we will consider (among others) a unicorn existing in the context of some novel.

equivalence, equality and inclusion in OWL

- `rdfs:subClassOf`(A, B) is interpreted as

$$(\forall x)A(x) \Rightarrow B(x) \text{ and symbolized as } A \leq B.$$

- `owl:equivalentClass`(A, B) is interpreted as

$$(\forall x)A(x) \Longleftrightarrow B(x) \text{ and symbolized as } A \approx B.$$

- Equivalent concepts can differ in the way of determining their extension.
- `owl:sameAs`(A, B) is understood as two constants referring to the same entity and we symbolize it as

$$A = B.$$

- Synonyms are words or phrases that have the same meaning (intension); Two names of the same concepts, eg. doctor and physician.

Conjunctions defining types

- `owl:intersectionOf`$(A, \{B_1, \ldots, B_n\})$ is the definition of types intersection

$$A(x) \iff (B_1 \cap \cdots \cap B_n)(x) \iff B_1(x) \wedge \cdots \wedge B_n(x).$$

- `owl:unionOf`$(A, \{B_1, \ldots, B_n\})$ is the definition of types sum

$$A(x) \iff (B_1 \cup \cdots \cup B_n)(x) \iff B_1(x) \vee \cdots \vee B_n(x).$$

- `owl:complementOf`(A, B) is the definition of a type complementary to a given type

$$A(x) \iff (\texttt{owl:Thing} \setminus B)(x) \iff \neg B(x).$$

Disjointness and defining through enumeration

- `owl:disjointWith`(A, B) says that A and B are disjoint.

$$A \cap B \approx \texttt{owl:Nothing}, \text{ so } \neg(\exists x)A(x) \wedge B(x)$$

- `owl:oneOf`$(A, \{v_1, \ldots, v_n\})$ defines a type through enumerating all the individuals belonging to it, ie. by extension

$$A(x) \iff x \in \{v_1, \ldots, v_n\} \iff x = v_1 \vee \cdots \vee x = v_n$$

Relations

- `owl:ObjectProperty` a type containing binary relations on individuals, a subtype of `rdf:Property`
- subtypes `owl:ObjectProperty`:
  `owl:AsymmetricProperty`, `owl:SymmetricProperty`,
  `owl:IrreflexiveProperty`, `owl:ReflexiveProperty`,
  `owl:FunctionalProperty`,
  `owl:InverseFunctionalProperty`,
  `owl:TransitiveProperty`
- relations on relations (elements of `owl:ObjectProperty`)
  - `owl:equivalentProperty` equivalence
  - `rdfs:subPropertyOf` being a subrelation
  - `owl:inverseOf` being an inverse
  - `owl:propertyDisjointWith` being disjoint
- full relation: `owl:topObjectProperty`,
- empty relation: `owl:bottomObjectProperty`.

Meaning: assigning concepts to entities

- Methods of ascribing types to entities:
  - indicate an existing object and ascribe a type-label to it (it requires sensors, world perception and a language capable of referring to particulars in the world)
  - define a type — describe it with more basic concepts, eg. a symmetric relation is a relation that satisfies the axiom

$$(\lambda R)(\forall x)(\forall y)xRy \Longrightarrow yRx$$

- Methods of checking whether an entity belongs to a type
  - classifiers, eg. an algorithm for recognizing faces is the meaning of the concept “face”
  - inference in logic (with a proof system) when the entity is defined axiomatically.
  - analysing whether the defining formula is satisfied.
- The methods can be combined:
  - The description of a concept can consist of both a formal definition and a number of instances.
  - Face recognition can be based on the structure of a face so that the classifiers recognize elements of the face and a definition that puts them together.

The world outside the ontology

- When analysing texts or pictures we can come upon data that doesn't suit the ontology assumed.
- Entities with no suitable concepts in the ontology cannot be placed in the data structures and they are invisible for logical reasoning.
- For certain data we may end up at different conclusions depending on particular algorithms for determining the meaning, the extension of the concepts and the relations between them (their axiomatization).
- Eg. a space ontology based on pixels or on a hexagonal grid.
- Making the properties discreet or continuous alters the inference outcomes.
- The ontology allows for logical inference but restricts its extend as well.
- The same applies to people but they do not and cannot notice it.

Restrictions

`owl:Restriction` types obtained through choosing elements of other types that satisfy a condition, a subclass of `owl:Class`

```
<owl:Class rdf:about=C >
  ...
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource=P>
      ...  declaration of restriction R...
    </owl:Restriction>
  </rdfs:subClassOf>
  ...
</owl:Class>
```

$$C \leq (\lambda x) R(x)$$

Existentially quantified restriction

```
<owl:Class rdf:about='HeldInEuropeConf'>
  <rdfs:subClassOf rdf:resource='Conf'/>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource='heldIn'/>
      <owl:someValuesFrom
          rdf:resource='EuropeanCity'/>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

$$\text{HeldInEuropeConf}(x) \Longrightarrow \text{Conf}(x) \wedge (\exists y)\text{heldIn}(x, y) \wedge \text{EuropeanCity}(y)$$

For a given binary relation f and a concept A we define the inverse image of A under f as $(f^{-1}(A))(x) \Longleftrightarrow (\exists y) f(x, y) \wedge A(y)$ and obtain

$$\text{HeldInEuropeConf} \leq \text{Conf} \cap \text{heldIn}^{-1}(\text{EuropeanCity})$$

Universally quantified restriction

```
<owl:Class rdf:about='EuropeanConf'>
  <rdfs:subClassOf rdf:resource='Conf'/>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource='heldIn'/>
      <owl:allValuesFrom
          rdf:resource='EuropeanCity'/>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

$$\mathrm{EuropeanConf}(x) \Longrightarrow \mathrm{Conf}(x) \wedge (\forall y)\mathrm{heldIn}(x,y) \Longrightarrow \mathrm{EuropeanCity}(y)$$

For a given binary relation f and a concept A we define the image of A under f as $(f(A))(y) \Longleftrightarrow (\exists x) f(x,y) \wedge A(x)$. And since the formula $(\forall x)(A(x) \Rightarrow (\forall y) f(x,y) \Rightarrow B(y))$ is equivalent to the formula $(\forall y)((f(A))(y) \Rightarrow B(y))$ we obtain

$$\mathrm{EuropeanConf} \leq \mathrm{Conf} \qquad \mathrm{heldIn}(\mathrm{EuropeanConf}) \leq \mathrm{EuropeanCity}$$

Restriction on the cardinality of the relation

```
<owl:Class rdf:about='Publication'>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource='noPages'/>
      <owl:cardinality
          rdf:datatype= 'xsd:nonNegativeInteger'>
          7
      </owl:cardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

$$\text{Publication}(x) \implies \text{card}(\{y : \text{noPages}(x, y)\}) = 7$$

Similarly: `owl:maxCardinality` and `owl:minCardinality`.

Restriction on the value of a relation

```
<owl:Class rdf:about='ConfInBerlin'>
  <rdfs:subClassOf rdf:resource='Conf'/>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource='heldIn'/>
      <owl:hasValue rdf:resource='Berlin'/>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

$$\text{ConfInBerlin}(x) \Longrightarrow \text{Conf}(x) \wedge \text{heldIn}(x, '\text{Berlin}')$$

$$\begin{aligned} \text{ConfInBerlin} &\leq \text{Conf} \\ \text{ConfInBerlin} &\leq \text{heldIn}^{-1}(\{'\text{Berlin}'\}) \end{aligned}$$

An example use of types intersection

```
<owl:Class rdf:about='TrulyEuropeanConf'>
  <owl:intersectionOf rdf:parseType='Collection'>
    <owl:Class rdf:about='Conf'/>
    <owl:Restriction>
      <owl:onProperty rdf:resource='heldIn'/>
      <owl:allValuesFrom
          rdf:resource='EuropeanCity'/>
    </owl:Restriction>
  </owl:intersectionOf>
</owl:Class>
```

$$\text{TrulyEuropeanConf}(x) \Longrightarrow (\text{Conf} \cap (\forall y)\text{heldIn}(x, y) \Rightarrow \text{EuropeanCity}(y))(x)$$

$$\begin{aligned} \text{TrulyEuropeanConf} &\leq \text{Conf} \\ \text{heldIn}(\text{TrulyEuropeanConf}) &\leq \text{EuropeanCity} \end{aligned}$$

An example use of type complement

```
<owl:Class rdf:about='NonEuropeanConf'>
  <owl:intersectionOf rdf:parseType='Collection'>
    <owl:Class rdf:about='Conf'/>
    <owl:Class>
      <owl:complementOf>
        <owl:Restriction>
          <owl:onProperty rdf:resource='heldIn'/>
          <owl:allValuesFrom
            rdf:resource='EuropeanCity'/>
        </owl:Restriction>
      </owl:complementOf>
    </owl:Class>
  </owl:intersectionOf>
</owl:Class>
```

$$\text{NonEuropeanConf}(x) \Longleftrightarrow (\text{Conf} \cap$$

$$\cap \neg(\forall y)\text{heldIn}(x, y) \Rightarrow \text{EuropeanCity}(y))(x)$$