Semantic Web

Carlo Meghini

Istituto di Scienza e Tecnologie della Informazione
Consiglio Nazionale delle Ricerche – Pisa

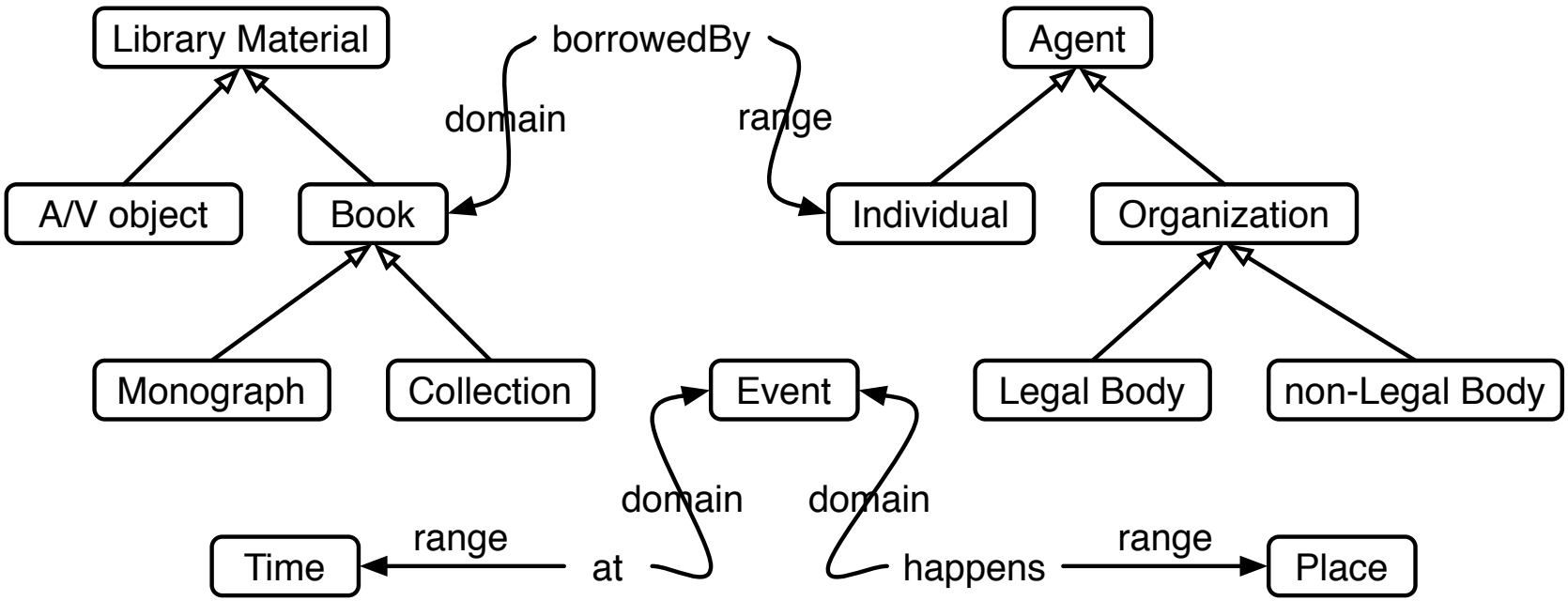
2017-18

Today's Lecture: OWL

Introduction

We have already encountered the notions of conceptualization and ontology, while discussing mechanisms to organize the knowledge in a semantic netowork.

We have come up with a simple form of ontology:



We have also seen an RDF vocabulary for expressing ontologies, the RDF Schema vocabulary.

RDF Schema introduces a few, very important IRIs for defining classes and taxonomies, and for typing properties.

The good news is that we have a sound, complete and efficient algorithm to compute the generalized RDF (RDFS) closure of a graph towards another graph, so we can compute all the triples implicit in a graph. And be smart.

Some of these inferences are not drammatically informative, *e.g.*:

- that a class is a sub-class of itself

Some other inferences may bring more useful knowledge, *e.g.*:

- that something which happens (somewhere) is an event
- that if Carlo is a Man and Man is a subclass of Person, then Carlo is a Person.

But we would like to have more inferences.

For instance:

- that if Carlo is a Man, then Carlo is not a Woman (class disjointness)
- that Carlo cannot have two fiscal codes (property functionality)
- that if Carlo is born in Macerata and Macerata is in Italy, then Carlo is an Italian citizen (property chain)
- that if Carlo is a child of Giuliana, then Giuliana is not a child of Carlo (property asymmetry)
- that if Alitalia flies from Pisa to Rome, it also flies from Rome to Pisa (property symmetry)
- that if I, J and K are time intervals, I is contained in J and J is contanied in K, then I is contained in K (property transitivity)
- that if an event E causes an event J, then J cannot precede E in time
- ...

All these inferences are possible if we can represent in an explicit way the meaning of the classes and of the properties that we use to model our domain.

In general, a more articulated representation of the semantics of classes and properties, would allow us to:

- better document our ontology
- introduce more automation in knowledge management, *i.e.*, more functionality with less code

For these reasons, we will now look into OWL, a powerful language of the Semantic Web family for expressing riche ontologies.

Types of ontologies

There exist different kinds of ontology, according to their level of generality.

- *Top-level* ontologies describe very general concepts like space, time, matter, object, event, action, *etc.*, which are independent of a particular problem or domain: it seems therefore reasonable to have unified top-level ontologies for large communities of users.
- *Domain* ontologies and *task* ontologies describe, respectively, the vocabulary related to a generic domain (like medicine, or automobiles) or a generic task or activity (like diagnosing or selling), by specializing the terms introduced in the top-level ontology.
- *Application* ontologies describe concepts depending both on a particular domain and task, which are often specializations of both the related ontologies. These concepts often correspond to roles played by domain entities while performing a certain activity, like a male Person being a spouse or a best-man in a wedding ceremony.

Ontology development

Since ontology was identified as a crucial tool for the development and the integration of information systems, many applied philosophers engaged themselves with the capturing in a formal way, **once for all**, the fundamental notions for knowledge representation, such as:

- part-whole
- time
- space
- quality
- agency
- events
- ...

Many of the questions that need to be answered in order to develop a formal theory of these notions are long-standing issues in philosophy:

- do universals exist? abstract entities? possible worlds? ...

or in science:

- what is time? how time and space relate? what is light? what is energy? how does it relate to matter?

The difference with the past is that now computerized information systems are very numerous and play a fundamental roles in human activities, and the formal capturing of these notions is *essential* for the *cost-effective development of sustainable information systems.*

The cooperation between philosophers and scientists in the development of ontologies has produced many ontologies that are built on top of the consolidated results in addressing the fundamental questions.

These ontologies are in use in information systems today:

- top ontologies: DOLCE, CIDOC CRM, CYC, . . . (order of dozens)
- domain ontologies: very many, possibly incompatible (order of hundreds)
- application ontologies: uncountable

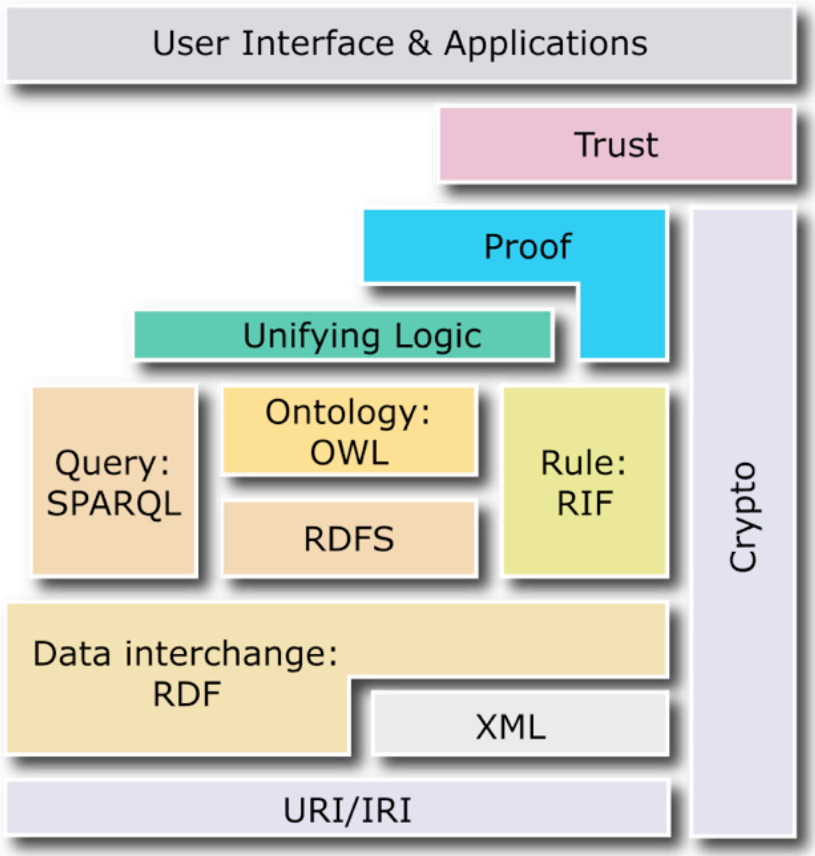
The engineering of an ontology is the result of the cooperation between a domain expert and an ontology language expert:

- the domain expert knows the terms of the domain, their definitions and their usage
- the ontology language expert knows how to translate the definitions into the expressions of an ontology language, and how to set up a system that uses such expressions to reach the final goals of the system.

For simple ontologies, the same person may cover both roles. For the successful engineering of realistic ontologies the cooperation is essential.

OWL

The language that we examine today is the Ontology Web Language (OWL), which in the Semantic Web stack is the language for defining ontologies.



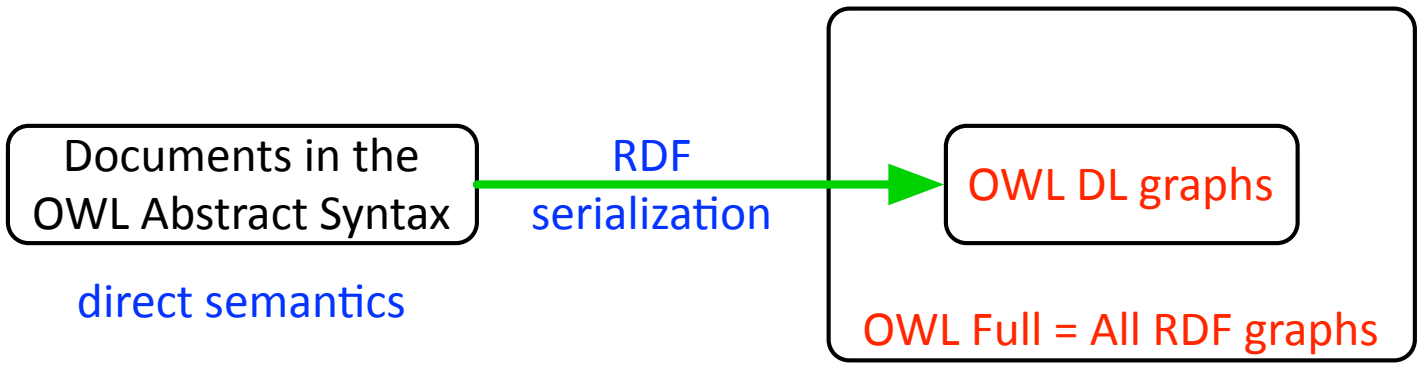
The OWL 2 Lineage and Family

OWL was developed within the World Wide Web Committe (W3C) in a series of subsequent activities carried out by different groups:

1. OWL 1, the first version of the language, became a W3C Recommendation in February 2004.
   - By the Web-Ontology (WebOnt) Working Group, which operated from November 2001 until May 2004
2. In December 2006, OWL 1.1, an extension of OWL 1, was proposed by some researchers for including the theoretical advances that had been in the meantime achieved in DLs, namely the DL SROIQ.
3. The OWL Working Group was created to turn OWL 1.1 into a new W3C recommendation for an updated OWL.
   - The Group operated from 2007 until December 2012, and
   - issued the first edition of OWL 2 in October 2009.
4. OWL 2, second edition, became a W3C Recommendation in December 2012.

As far as building OWL as a semantic extension of RDF, by far the hardest issue, two main versions of OWL were defined:

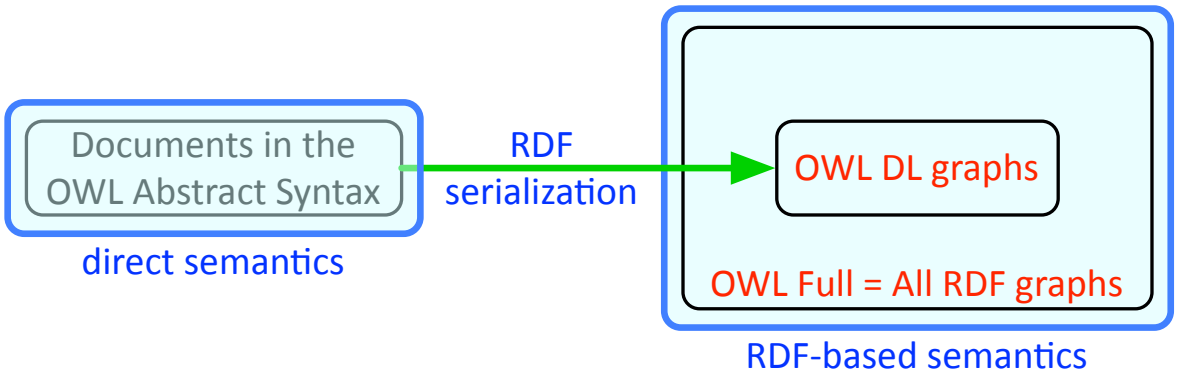
1. OWL DL, the language resulting from the encoding of the OWL abstract syntax into a concrete notation. OWL DL is **not** a semantic extension of RDF.
2. OWL Full, the language resulting from extending RDF Schema with the classes and properties needed for encoding the OWL abstract syntax in RDF. OWL Full is a semantic extension of RDF, and the language including all RDF graphs.



Two semantics, but equivalent on OWL DL

We have therefore two semantics for OWL:

- the direct semantics, defined on the OWL abstract syntax, and based on DLs model theory
- the RDF-based semantics, defined on the encoding of OWL Full in the RDF abstract syntax and based on the RDF model theory.



The direct semantics cannot be applied to OWL Full because the additional expressive power of OWL Full is meaningless in a DL.

Introduction

An OWL 2 ontology is a formal description of a domain of interest.

OWL 2 ontologies consist of three different syntactic categories:

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

An ontology is a resource identified by an IRI with an optional version.

Logically, an ontology consists of a set of axioms.

Physically, an ontology is associated with an ontology document, which physically contains the ontology. The ontology document should be accessible via the ontology IRI (if any), or via the ontology version (if any).

For modularization, an ontology can import other ontologies, specified via their document IRIs.

Datatypes

Datatypes are provided in OWL the same way they are provided in RDF, to allow using standardized values such as numbers and strings.

A set of datatypes supported by a reasoner is called a *datatype map.*

Most datatypes are taken from the set of XML Schema Datatypes, the RDF specification, or the specification for plain literals.

In addition, the OWL 2 datatype map adds:

- owl:real, whose value space is the set of real numbers, and whose lexical space is empty
- owl:rational, whose value space is the set of rational numbers, and whose lexical space is given by

  numerator '/' denominator

  where numerator is an xsd:integer and denominator is a positive, non-zero xsd:integer

The complete OWL 2 datatype map consists of the following datatypes:

owl:real
owl:rational
xsd:decimal
xsd:integer
xsd:nonNegativeInteger
xsd:nonPositiveInteger
xsd:positiveInteger
xsd:negativeInteger
xsd:long
xsd:int
xsd:short
xsd:byte
xsd:unsignedLong
xsd:unsignedInt
xsd:unsignedShort
xsd:unsignedByte

xsd:double
xsd:float

xsd:string
xsd:normalizedString
xsd:token
xsd:language
xsd:Name
xsd:NCName
xsd:NMTOKEN

xsd:boolean

xsd:hexBinary
xsd:base64Binary

xsd:anyURI

xsd:dateTime
xsd:dateTimeStamp

rdf:XMLLiteral

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
- Object Properties relate two individuals, and include:
  - owl:topObjectProperty, which connects all possible pairs of individuals
  - owl:bottomObjectProperty, which connects no pair of individuals
- 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

- Annotation Properties are used to make annotations for an ontology, axiom, or an IRI.
- Named Individuals, representing objects from the domain.
- Datatypes, each of which must be:
  - rdfs:Literal
  - a datatype in the OWL datatype map
  - a datattype defined by means of a datatype definition

In addition to entities, OWL 2 ontologies include:

- literals, similar to RDF literals, which can also be understood as individuals denoting data values
- 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.

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.

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 is "pairs of people such that the first is a parent of the second".

In some situations one may want to use childOf, in other situations the inverse may be more convenient.

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)

Class Expressions

Class expressions are provided in OWL 2 represent class intensions.

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

- Book $\rightarrow$ "resource that is an object and has textual content"
- Parent $\rightarrow$ "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.*,

- that a book has textual content
- that every parent has at least a son or a daughter

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 **that is a** Male **or** Female

From the semantical point of view, class expressions are like classes, *i.e.*, selections of fetures (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: "resource that can do something **and** is an individual"
- Union, as in: "Person of male gender **or** person of female gender"
- Complement, as in: "resource that is **not** a man"
- Enumeration, as in: "resource that is monday **or** tuesday **or** . . . **or** sunday"

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(Male Female)

Object property restrictions

Class expressions in OWL 2 can be formed by placing restrictions on object property expressions.

These include:

- Existential Quantification, as in: "resource that has **at least one** child who is a Male"
- Universal Quantification, as in: "resource that has **all** authors who are Italian"
- Individual Value Restriction, as in "resource that is born in **Italy**"
- Self Restriction, as in: "resource that is **self**-employed"

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	$\{x \mid \exists y : (x, y) \in (\text{OPE})^{OP} \text{ and } y \in (\text{CE})^{C}\}$
RDF Turtle Syntax	_:x rdf:type owl:Restriction . _:x owl:onProperty OPE . _:x owl:someValuesFrom CE .
Example	ObjectSomeValuesFrom(childOf Male)

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 differencies:

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: "resource that is at most 18 years old" (has age)
- Universal Quantification, as in: "resource that has **all** CAPs as integers"
- Literal Value Restriction, as in "resource that is **17** years old" (has age)

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: "resource that has **at least two** authors that are Italian"
- Maximal cardinality, as in: "resource that has **at most one** child that is Male"
- Exact cardinality, as in "resource that has **exactly two** members that are self-employed"

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	$\{x \mid \#\{y \mid (x,y) \in (\text{OPE})^{OP},\ y \in (\text{CE})^{C}\} = n\}$
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))

Note: the unqualified form: ObjectExactCardinality(n OPE) also exists, equivalent to ObjectExactCardinality(n OPE owl:Thing)

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 differencies:

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: "resource that has **at least two** phone numbers that are integers of seven digits"
- Maximal cardinality, as in: "resource that has **at most one** fiscal code which is a string of type CCC CCC NNCNN CNNNC"
- Exact cardinality, as in "resource that has **exactly one** birthdate that is a date"

Axioms

Axioms are the content of an OWL ontology.

Axioms express statements that are *true* in the domain that the ontology models.

There are several kinds of axioms:

- Declarations
- Axioms about classes
- Axioms about object or data properties
- Datatype definitions
- Keys
- Assertions, also called *facts*
- Axioms about annotations

Axioms about classes

Axioms about classes establish relationships between class expressions.

There are four kinds of class axioms:

- subclass axioms
- equivalent class axioms
- disjoint class axioms
- disjoint union axioms

Note: subclass and equivalent class axioms are present in RDF Schema too, however in OWL they connect *class expressions*, while in RDF Schema they connect simply classes.

In contrast, disjoint and disoint union axioms are not expressible at all in RDF Schema.

An example

Let us consider an ontology O with the following 4 axioms:

(1) A person that has a child has either at least one boy or a girl:

SubClassOf(PersonWithChild
ObjectSomeValuesFrom(hasChild ObjectUnionOf(Boy Girl)))

(2) Each boy is a child: SubClassOf(Boy Child)

(3) Each girl is a child: SubClassOf(Girl Child)

(4) If some resource has a child, then this resource is a parent:

SubClassOf(ObjectSomeValuesFrom(hasChild Child) Parent)

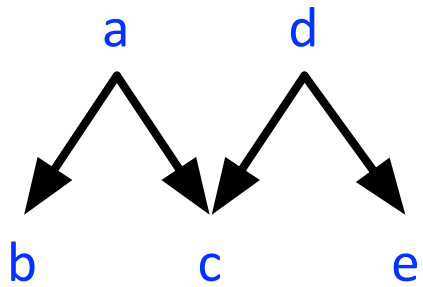
Then O entails: (5) SubClassOf(PersonWithChild Parent)

In every interpretation in which (1)-(4) are true, also (5) is true.

To verify the entailment, let us consider a "reasonable" interpretation I

$\Delta_I = \{a, b, c, d, e\}$
$(\text{Boy})^C = \{b, c\}$
$(\text{Girl})^C = \{e\}$
$(\text{Child})^C = \{b, c, e\}$
$(\text{PersonWithChild})^C = \{a, d\}$
$(\text{Parent})^C = \{a, d\}$
$(\text{ObjectUnionOf(Boy Girl))}^I = \{b, c, e\}$
$(\text{ObjectSomeValuesFrom(hasChild ObjectUnionOf(Boy Girl)))}^I = \{a, d\}$
$(\text{ObjectSomeValuesFrom(hasChild Child))}^I = \{a, d\}$

$(\text{hasChild})^{OP}$:



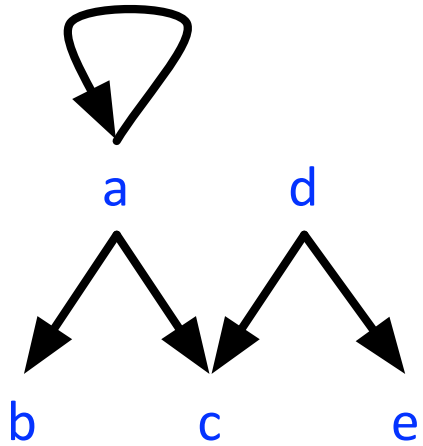
This interpretation is a model of the ontology O because it satisfies all axioms in O. And it satisfies also (5). So the entailment is verified by I.

It is reasonable because it also satisfies reasonable axioms that are not actually stated in O, namely "a Child is either a Boy or a Girl but not both", "a person with at least a child is a PersonWithChild", "a parent is a person with at least a child", "hasChild is irreflexive", ...

There could be less reasonable interpretations that are models of O anyway, such as J :

$\Delta_J = \{a, b, c, d, e, 1, 2\}$
$(\text{Boy})^C = \{b, c, 1\}$
$(\text{Girl})^C = \{e, 1\}$
$(\text{Child})^C = \{b, c, e, 1, 2\}$
$(\text{PersonWithChild})^C = \{d\}$
$(\text{Parent})^C = \{a, d, 2\}$
$(\text{ObjectUnionOf(Boy Girl))}^J = \{b, c, e, 1\}$
$(\text{ObjectSomeValuesFrom(hasChild ObjectUnionOf(Boy Girl)))}^J = \{a, d\}$
$(\text{ObjectSomeValuesFrom(hasChild Child))}^J = \{a, d\}$

$(\text{hasChild})^{OP}$:



Even though J is strange, it satisfies (1)-(4), so it is a model of O.

In addition, it satisfies (5), so it also verifies the entailment.

To rule out interpretations like J, we have to state more axioms!

Object Property Axioms

OWL 2 provides axioms that can be used to characterize and establish relationships between object property expressions.

These axioms are of the following kinds:

- object *sub-property:* isBestFriendOf is a sub-property of isFriendOf
- *complex role inclusion:* a mother's sister is an aunt SubObjectPropertyOf(ObjectPropertyChain(hasMother hasSister) hasAunt)
- *equivalent:* hasBrother is equivalent to hasMaleSibling
- *disjoint:* hasBrother is disjoint from hasSister
- *inverse:* isChildOf is the inverse of isParentOf
- *domain:* the domain of hasFather is Person
- *range:* the range of hasSister is Female

- *functional:* hasMother is functional
- *inverse functional:* the inverse of matherOf is functional
- *reflexive:* knows if reflexive (is it?)
- *irreflexive:* marriedTo is irreflexive
- *symmetric:* isRelativeOf is symmetric
- *asymmetric:* hasFather is asymmetric
- *transitive:* isDescendantOf is transitive

Data Property Axioms

OWL 2 also provides for data property axioms, which are similar to object property axioms.

These axioms are of the following kinds:

- data *sub-property:* hasLastName is a sub-property of hasName
- *equivalent:* hasName is equivalent to siChiama
- *disjoint:* hasName is disjoint from hasAddress
- *domain:* the domain of hasName is Person
- *range:* the range of hasName is xsd:string
- *functional:* hasFiscalCode is functional

Datatype Definitions

A datatype definition defines a new datatype DT as being semantically equivalent to (*i.e.*, a synonym of) a specified data range.

```
DatatypeDefinition(
    CodiceFiscale
    DatatypeRestriction(
        xsd:string
        xsd:pattern
        "[A-Z]{6}-[0-9]{2}-[A-Z]{1}-[0-9]{2}-[A-Z]{1}-[0-9]{3}-[A-Z]{1}" ) )
```

Keys

Similarly to keys in databases, a key in an OWL ontology is a set of object or data property expressions that uniquely identify the instances of a class expression.

More technically, a key axiom states that each named instance of a given class expression CE is uniquely identified by given object property expressions OPE_i and/or by given data property experssions DPE_j.

Example: HasKey(Person () (HasFiscalCode))

Note: if a key axiom is violated by two named individuals a and b, this does not per sé cause an inconsistency. The inconsistency arises only if the ontology states or entails that a and b are *different* individuals.

Assertions

Assertions are axioms about individuals, establishing relationships between individuals, or between individuals and class or property expressions. Kinds of asserions:

- same individual: Batman is the same individual as BruceWayne
- different individual: Superman is not the same individual as BruceWayne
- class assertion: carlo is a Male or a Female
- positive object property: the mother of carlo is Giuliana
- negative object property: the mother of carlo is Anna
- positive data property: Mozart was born on "1756-01-27"^^xsd:date
- negative data property: Mozart was born on "1956-01-27"^^xsd:date

Note: class assertions and positive property assertions are present in RDF Schema too; however, in OWL they connect individuals to class and property *expressions,* while in RDF they connect individuals to classes and properties only, respectively.

In contrast, the other kinds assertions are not expressible in RDF Schema.

Inference problems for an OWL DL Ontology

An OWL 2 ontology O is *satisfied* in an interpretation I if all axioms in the axiom closure of O are satisfied in I.

Ontology Consistency: O is consistent (or satisfiable) if a model of O exists.

Ontology Entailment: O entails O_1 if every model of O is also a model of O_1 w.r.t. D and V.

Class Expression Satisfiability: CE is satisfiable w.r.t. O if a model I of O exists such that $(CE)^C \neq \emptyset$.

Class Expression Subsumption: CE_1 is subsumed by a class expression CE_2 w.r.t. O if $(CE_1)^C \subseteq (CE_2)^C$ for each model I of O.

Instance Checking: a is an instance of CE w.r.t. O and if $(a)^I \in (CE)^C$ for each model I of O.

Global Restrictions on Axioms in OWL 2 DL

The axioms of each OWL 2 DL ontology O must satisfy global restrictions that are necessary in order to obtain a decidable language.

The formal definition of these conditions relies on the notion of a *property hierarchy* and of *simple object property expressions.*

These restrictions concern:

- the owl:topDataProperty
- datatypes
- simple roles
- property hierarchy
- anonymous individuals

Restriction on owl:topDataProperty

The owl:topDataProperty property occurs in Ax only as the second argument of SubDataPropertyOf axioms.

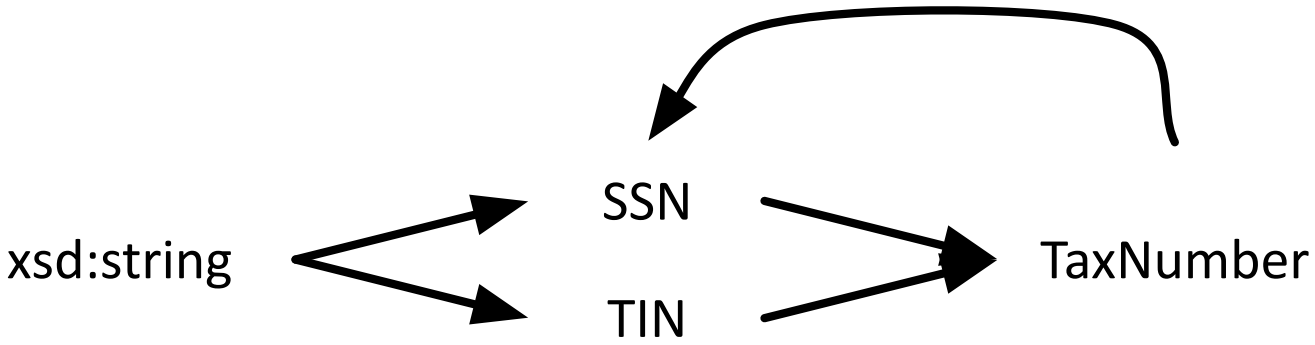
Without this restriction, owl:topDataProperty could be used to write axioms about datatypes, which would invalidate a fundamental Theorem.

Restriction on Datatypes

```
Declaration( Datatype( SSN ) )
Declaration( Datatype( TIN ) )
Declaration( Datatype( TaxNumber ) )
DatatypeDefinition(
   SSN
   DatatypeRestriction( xsd:string xsd:pattern "[0-9]3-[0-9]2-[0-9]4" ) )
DatatypeDefinition(
   TIN
   DatatypeRestriction( xsd:string xsd:pattern "[0-9]11" ) )
DatatypeDefinition( TaxNumber DataUnionOf( SSN TIN ) )
```

These datatype definitions are acyclic: SSN and TIN are defined in terms of xsd:string, and TaxNumber is defined in terms of SSN and TIN.

If we now add an axiom defining SSN in terms of TaxNumber, then datatypes SSN and TaxNumber could not be simply "unfolded", which is contrary to the intended meaning of these datatypes.



This situation, however, is disallowed by imposing that an ordering exists between the involved datatypes. Since no ordering exists, the extended axiom set is excluded.

Restriction on Simple Roles

An object property expression OPE is *simple in* Ax if it, or it inverse, does not have any sub-property that is the right-hand side of a complex role inclusion axiom.

Consider the ontology consisting of the following axioms:

SubObjectPropertyOf(ObjectPropertyChain(hasFather hasBrother) hasUncle)
The brother of someone's father is that person's uncle

SubObjectPropertyOf(hasUncle hasRelative)
Having an uncle implies having a relative

SubObjectPropertyOf(hasBiologicalFather hasFather)
Having a biological father implies having a father

- hasUncle is not simple, because it occurs in an complex role inclusion axiom
- hasRelative is not simple either, because it has a subproperty that is not simple
- hasBiologicalFather is simple, and so is hasFather

Each class expression and each axiom in Ax of type from the following two lists contains only simple object properties:

(1) ObjectMinCardinality, ObjectMaxCardinality, ObjectExactCardinality, and ObjectHasSelf.

(2) FunctionalObjectProperty, InverseFunctionalObjectProperty, IrreflexiveObjectProperty, AsymmetricObjectProperty, and DisjointObjectProperties.

This restriction is necessary in order to guarantee decidability of the basic reasoning problems for OWL 2 DL.

Restriction on the Property Hierarchy

The main goal of this restriction is to prevent cyclic definitions involving object subproperty axioms with property chains. Consider the following ontology:

SubObjectPropertyOf(ObjectPropertyChain(hasFather hasBrother) hasUncle)
The brother of someone's father is that person's uncle.

SubObjectPropertyOf(ObjectPropertyChain(hasUncle hasWife) hasAuntInLaw)
The wife of someone's uncle is that person's aunt-in-law.

The second axiom depends on the first one, but not vice versa; hence, these axioms are not cyclic and can occur together in the axioms of an OWL 2 DL ontology.

To verify this condition formally, it suffices to find one strict partial order $<$ on object properties such that each property is defined only in terms of the properties that are smaller w.r.t. $<$. For example:

hasFather	hasUncle
hasBrother	hasUncle
hasUncle	hasAuntInLaw
hasWife	hasAuntInLaw

In contrast to the previous example, the following axioms are cyclic and do not satisfy the restriction on the property hierarchy.

SubObjectPropertyOf(ObjectPropertyChain(hasFather hasBrother) hasUncle)
The brother of someone's father is that person's uncle.

SubObjectPropertyOf(ObjectPropertyChain(hasChild hasUncle) hasBrother)
The uncle of someone's child is that person's brother.

The second axiom depends on the first one and vice versa; hence, these axioms are cyclic and can**not** occur together in the axioms of an OWL 2 DL ontology.

To verify this condition formally, one must prove that no strict partial order $<$ on object properties can be found that satisfies the restriction on the property hierarchy.

Some cyclic definitions are known not to lead to decidability problems:

SubObjectPropertyOf(ObjectPropertyChain(hasChild hasSibling) hasChild)
The sibling of someone's child is that person's child.

Restrictions on Anonymous Individuals

These restrictions ensure that each OWL 2 DL ontology with anonymous individuals can be transformed to an equivalent ontology without anonymous individuals, thereby ensuring decidability of the basic reasoning problems.

Roughly speaking, this is possible if property assertions connect anonymous individuals in a tree-like way.

Consider the following ontology:

ObjectPropertyAssertion(hasChild Bob _:a) Bob has some unknown child.

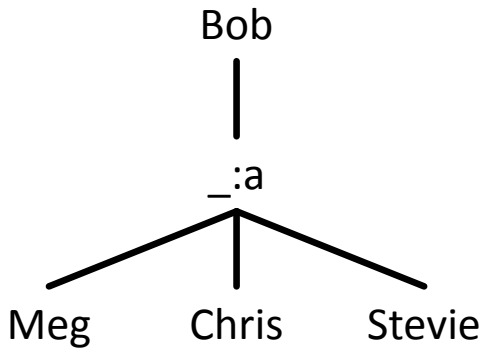
ObjectPropertyAssertion(hasChild _:a Meg) This unknown child has Meg . . .

ObjectPropertyAssertion(hasChild _:a Chris) . . . Chris . . .

ObjectPropertyAssertion(hasChild _:a Stewie) . . . and Stewie as children.

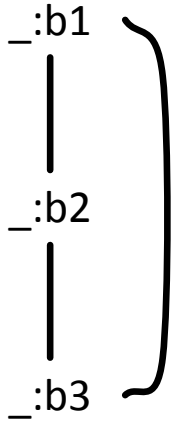
The connections between these can be understood as a tree that has _:a as its root, so they can be replaced by the equivalent assertion:

```
ClassAssertion(
    ObjectSomeValuesFrom(hasChild
        ObjectIntersectionOf(
            ObjectHasValue(hasChild Meg)
            ObjectHasValue(hasChild Chris)
            ObjectHasValue(hasChild Stewie)))
    Bob)
```

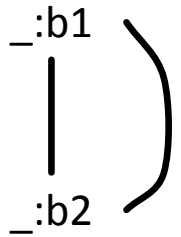


Unlike in the previous example, the following ontologies do not satisfy the restrictions on anonymous individuals:

ObjectPropertyAssertion(hasSibling _:b1 _:b2)
ObjectPropertyAssertion(hasSibling _:b2 _:b3)
ObjectPropertyAssertion(hasSibling _:b3 _:b1)



ObjectPropertyAssertion(hasChild _:b1 _:b2)
ObjectPropertyAssertion(hasDaughter _:b1 _:b2)



In both cases, the anonymous individuals are connected by property assertions in a non-tree-like way. These assertions can therefore not be replaced with class expressions, which can lead to the undecidability of the basic reasoning problems.

Conclusions

OWL 2 DL is the most sophisticated and recent language for knowledge representation on the web.

OWL DL 2 is based on the description logic SROIQ, which is the most expressive decidable description logic that offers a level of expressivity adequate to realistic applications.

In order to retain decidability, several restrictions are placed on an OWL2 DL ontology. These restrictions limit the exprssivity of the language but at the same time they guarantee full control of the systems built on top of OWL 2 DL.

The engineering of inference engines able to reason about OWL 2 DL ontologies is a current research topic in the engineering of theorem provers.

It is reasonable to expect that OWL 2 DL will stay with us for a long while.

Useful readings

- Boris Motik, Peter F. Patel-Schneider, Bijan Parsia, eds. OWL 2 Web Ontology Language: Structural Specification and Functional-Style Syntax (Second Edition) W3C Recommendation, 11 December 2012. http://www.w3.org/TR/owl2-syntax/.
- Boris Motik, Peter F. Patel-Schneider, Bernardo Cuenca Grau, eds. OWL 2 Web Ontology Language: Direct Semantics (Second Edition) W3C Recommendation, 11 December 2012. http://www.w3.org/TR/owl2-direct-semantics/.
- Michael Schneider, editor. OWL 2 Web Ontology Language: RDF-Based Semantics (Second Edition) W3C Recommendation, 11 December 2012. http://www.w3.org/TR/owl2-rdf-based-semantics/.