Semantic Web

Carlo Meghini

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

2017-18

Today's Lecture: Abstraction Mechanisms and RDF Schema

SPARQL allows manipulating in a declarative way large quantities of data, disregarding all technical details of formats, storage and access.

There is still a practical problem that SPARQL alone cannot solve.

A semantic network of realistic size comprises thousands of nodes and thousands of arcs and can easily go to millions these days: how do we keep it under control? *i.e.*, how do we make sure

- that the same resource is always represented by the same IRI
- that properties are used correctly and consistently
- ...

Moreover, how to modularize the body of knowledge represented in the network, so that that knowledge can be communicated to or explored by someone who is not familiar with it?

These are difficult problems ...

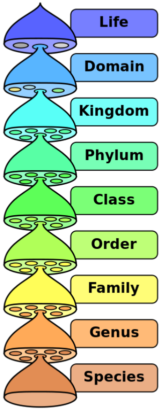
. . . but fortunately they have been studied and solved long before computers were invented!

For instance, in natural science.

Natural science is a branch of science concerned with the description, prediction, and understanding of natural phenomena, based on empirical evidence from observation and experimentation (wikipedia).

Natural scientists are very busy producing knowledge. They collect a myriad of facts (observations) and describe them for the purpose of understanding and communicating the underlying laws.

To do that in an effective way, natural sciences have defined *knowledge organization mechanisms.* For instance, biology has created *groups* (of many kinds) and has organized these groups in a *taxonomy.*



Abstraction mechanisms

Computer Science has built on top of the techniques that have been introduced in natural science to organize large bodies of knowledge about natural or social phenomena.

The result are *abstraction mechanisms*, which are employed in AI, databases and programming languaes to modularize large bodies of knowledge, data and code.

Abstraction results in the suppression of details, and abstraction mechanisms use it to introduce new entities in that help organizing knowledge in a semantic network.

These entities form a new network, **distinct** from the one that we have considered so far, but **related** to it in a way that, as we will discover, is very well known to us.

This other network clearly adds to the complexity of the system, but it is an helpful addition because it highlights relevant features of the original network.

Classification

The first abstraction mechanism that we consider is borrowed from natural sciences and is based on *grouping.*

The idea is to divide the individuals in the domain of discourse in groups, each group including those individuals that share some features, and as such are *identical* one another with respect to those features.

A group defined for this purpose is called a *class.*

A class is an abstract entity that represents a set of features of the individuals in the domain of discourse.

The features that a class represents give the meaning of the class, and are called the *intension* of the class.

When modelling a certain domain, classes are introduced by:

1. *considering* the features of the individuals in the domain of discourse
2. *selecting* the features that are considered as relevant for a certain purpose
3. *assigning* a name at those features, thereby creating a *class*
4. *grouping* into the same class the things that share the features that define the class.

Selecting features

For example, let us consider the knowledge domain, which comprises individuals that are knowledge carriers, such as monographs, periodicals, web resources, courses, seminars, digital ontologies, knowledge bases, encyclopedias, . . .

Each of these objects possess a specific set of features that distinguish that object from the other objects:

- nature: event, object, . . .
- periodicity: cyclic, unique, . . .
- medium: speech, text, formal knowledge, . . .
- language: which language is used in the resource
- creation: when and where the resource was created
- creator: who created it
- cost: how mach do you need to have it
- . . .

Suppose that for our application it is useful to have the group that includes the resources that in English are called “books”.

In order to define the class of books, we make an *abstraction:*

- we *select* some features: object, unique, textual, ...
- we *suppress* all other features: language, creator, birthdate, birthplace, cost ...

Two resources that share the selected features are *undistinguishable:* they belong to the same class, whatever suppressed features (creation, creator, cost, language, ...) they have.

- The Bible and The Weaverley belong to the same class, because they share the features that constitute the intension of the class, *i.e.*, they are objects (and not events), unique, textual, ...

The intension of the class of books is “resource that is an object, unique, textual, ...”

Every class that we create becomes part of our *conceptualization,* that is of the abstract, simplified view of the world that is of interest for us–in the example, the world of knowledge resources.

Examples of classes:

- in a library domain: book, people, topic, . . .
- university: student, teacher, course, department, assignment, . . .
- business: customer, salesman, item, invoice, . . .

Classes are *universals:* In metaphysics, a universal is “what particular things have in common, namely characteristics or qualities” (wikipedia).

Do classes really exist? This is an ancient problem in metaphysics about whether universals exists.

From an engineering point of view, classes are very useful abstractions, so we will use them to organize our semantic networks.

Assigning a name (and a meaning)

Once we have introduced a class in our conceptualization, we have to select a *symbol*, or a name, for identifying it.

Naturally enough we select a word from some natural language that *best approximates* the features that we have in mind for the class, so we call our class "Book".

This choice is arbitrary: we could have chosen a name from another language "Libro", "βιβλίο", or a neutral name, such as "C07123".

What really matters is the *usage* of the class in the representations of the world that we build.

Classification generates a number of classes, each representing an intension via a symbol that is the name of the class.

What is the meaning of a class?

As we said, the meaning of a class is given by the class intension, *i.e.*, that features that the class represents.

How to express in a mathematical way the intension of a class?

We will follow the same method that we followed for the meaning of sentences. Recall: *to understand a sentence, or to know what the sentence means, is to know precisely in which possible worlds the sentence is true and in which possible worlds the sentence is false.*

To understand a class, or to know what a class means, is to know precisely which individuals are members of the class in each possible world.

Or, put in a different way, to know the intension of the class is to be able to say, for each individual and each possible world, whether that individual is or is not a member of that class in that world.

The set of members of a class in a possible world is called the *extension* of the class in that world.

In sum, the intension of a class is the extension of the class in every possible world. Two classes are the same if they have the same extension in every possible world, that is, they have the same intension.

Grouping resources in a class

Classes are introduced to group resources into them. To use classification in a language we need a property for linking resources to the classes where they belong.

Such property is expressed in natural language by the predicate “is”.

In semantic modelling, the relationship type is called *instance of.*

In the RDF Vocabulary, we have the property rdf:type

Relationship:	“The Waverley” is a book
Resources:	“The Waverley”, book
Type of relationship:	rdf:type
Rank of the type:	2

Relationship:	Carlo is a person
Resources:	Carlo, person
Type of relationship:	rdf:type
Rank of the type:	2

However, in the RDF vocabulary we have no means to state that a certain resource is a class.

We would need some IRI X that we can use as follows:

Book rdf:type X .

to mean that Book is a class.

But the RDF vocabulary does not provide such an X.

As we will see, the RDF Schema vocabulary will remedy to this situation, by providing IRIs to properly represent the concepts that we are examining for organizing knowledge.

The semantics of these new IRIs will capture the notions of meaning that we are discussing now.

Multiplicity of class membership

Every object in a semantic net must be an instance of a class, and may be instance of two or more classes, to capture orthogonal dimensions of modeling:

- the diachronic dimension
  - an object representing a person will be an instance of class Person for all its lifetime, as Person is a natural kind
  - the time-dependent aspects of the object will be represented by making the object an instance of other classes, *e.g.*, Boy, Student, Employee, Father, Retired, etc.
- the modularity of the model, which may capture different aspects of the same reality by relying on separated classes, *e.g.*:
  - descriptive aspect, *e.g.*, single, married, divorced, *etc.*.
  - administrative aspect, *e.g.*, public-servant, free-lancer, *etc.*.
  - preservation aspect
  - ...

Grouping relationships in properties

The same way we group objects in classes, we group relationships in properties. In fact we have defined a property as a *type of relationships.*

In defining a property, we select the features that distinguish relationships of the same kind and disregard all other features. These features are the intension of the property that we define. For instance,

- the intension of the property IsBorrowedBy is "person and library resource such that that person is the physical custodian of the library resource for a fixed amount of time". An instance of this property is the relationship "The Waverley IsBorrowedBy Carlo".
- the intentio of the property HasChild is "pairs of persons such that the latter is a child of the former", and an instance can be "Carlo HasChild Giulia".

A property is a universal, features that particulars have in common.

Examples of properties:

- library: IsBorrowedBy (book, individual), isAbout (book, concept), . . .
- social: happens (event, time), at (event, place), . . .
- university: teaches (teacher, course), attends (student, course), . . .

What is the semantics of a property?

To understand a property, or to know what a property means, is to know precisely which pairs of individuals are members of the property in each possible world.

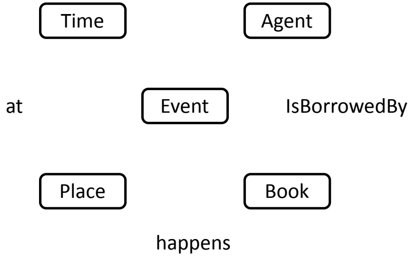
The set of pairs of individuals that are members of a property in a possible world is called the *extension* of the property in that world.

In mathematics, a set of pairs is called a binary relation.

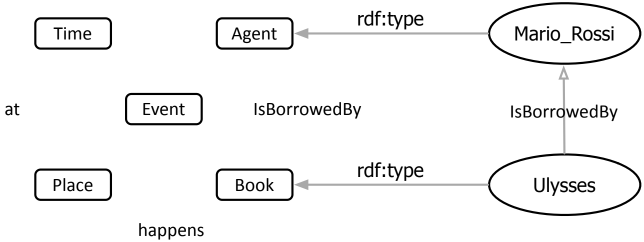
So, the extension of a property in a world is a binary relation.

In sum, the intension of a property is the extension of the property in every possible world. Two properties are the same if they have the same extension in every possible world, that is, they have the same intension.

If we apply classification to our example knowledge domain, we may end up with a bunch of classes and properties as follows:



The new network that we are building is connected to the semantic network we started from:



Specialization

The next abstraction mechanism that we introduce is also borrowed from natural sciences, and it is based on the notion of *taxonomy.*

In fact, the mechanism is not really different from classification, it just results from the systematic application of classification.

In this mechanism, we focus on one class, which we call the *pivot class,* and consider the intension of that class.

- For example, we focus on class Agent, whose intension is “resource that can do something”.

In order to derive the Agent class, we have considered the whole domain, focussed on some features and suppressed all the others. As a result, the members of the class share all those features.

- all agents can do something, whether they are persons or companies, legal bodies or non-legal bodies, Italian or foreign, *etc.*

As a first approximation, this suppression is very useful to organize our domain in main categories.

Now suppose that some of the differencies that we suppressed in defining the pivot class, become relevant for our application.

- The difference between agents that are organizations and individual agents is important, and we want to introduce it in our conceptualization of the world.

So we need to introduce these features by defining new, apposite classes that capture them.

But this is an exercise that we have already done when we have defined the pivot class. The only difference is that now we do not consider the whole domain of discourse, but the intension of the pivot class.

To introduce the classes that capture the relevant features, we *recursively* apply grouping to the intension of the pivot class, as we did in the first place for the whole domain, namely:

- we consider the intension of the pivot class
- we select those features that we consider relevant *within the intension of the pivot class*
- we assign a name to these features to define a new class
- we group into the new class the things that share these features

In our example, we consider the intension of the pivot class "resource that can do something", and two new features within it "resource that is an individual" and "resource that is an organization" and we define two new classes as follows:

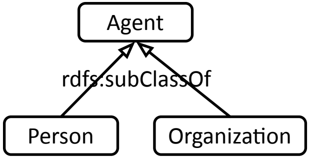
- "resource that can do something **and** *is an individual*" $\rightarrow$ class Person
- "resource that can do something **and** *is an organization*" $\rightarrow$ class Organization

This operation is called "specialization" (of the pivot class) and the type of relationship that it establishes between the involved classes is a property known in semantic modelling as "IsA".

In the semantic web, such property is part of the RDF Schema vocabulary, and it is the rdfs:subClassOf property.

Each class generated by specialization is a *sub-class of* the pivot class, or the pivot class is a *super-class of* each of the new classes.

The relationships of the rdfs:subClassOf type form the so-called *class hierarchy* or *class taxonomy*.



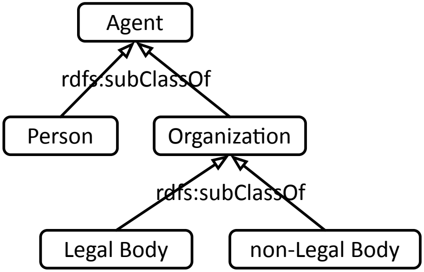
We can now repeat the exercise to introduce the difference between organizations that are legal bodies and organizations that are not.

Organization is now our pivot class, the intension of which is "resource that can do something and is an organization". The new classes are:

- "resource that can do something and is an organization and is a legal body" → LegalBody
- "resource that can do something and is an organization and is not a legal body" → NonLegalBody

LegalBody and NonLegalBody are sub-classes of Organization, as is reflected in the structure of the class hierarchy.

In this way a taxonomy is generated, similar to the taxonomies in science.



Generalization

We have created the class hierarchy *from the top down,* starting from a pivot class and *refining* its intension to generate *specializations.*

Likewise, we could develop the class hierarchy *from the bottom up:*

- we consider two or more classes C_1, C_2, $\ldots C_k$, $k \geq 2$ as pivot classes
- we unite their intensions and assign it to a new class C
- consequently, C_1, C_2, $\ldots C_k$, are all sub-classes of C
- we say that C is a *generalization of* (or a *super-class of*) C_1, C_2, $\ldots C_k$,

Generalizations are useful to "tie things up", defining classes that close a hierarchy at the top.

Let's see an example.

Suppose we have created classes Male ("Person of male gender") and Female ("Person of female gender"), and we need a class of "resources who can have children".

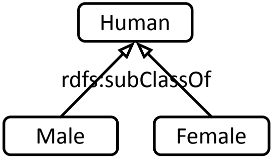
We consider Male and Female as pivot classes and generalize them by creating a new class Human whose intension unites the intension of both of them:

- "Person of male gender **or** person of female gender" $\rightarrow$ Human

Since male and female are the only genders from the reproduction point of view, the intension of Human is equivalent to "Person".

Human is a generalization of Male and Female.

Human is a super-class of Male and is a super-class of Female.



The semantics of rdfs:subClassOf

The semantics (intension) of a property is a binary relation in every possible world.

Which binary relation is the meaning of rdfs:subClassOf?

Since rdfs:subClassOf is a property combining classes, in every possible world the extension of rdfs:subClassOf will be pairs of class extensions.

Now, what mathematical relation should hold between these sets?

Intuitively,

A rdfs:subClassOf B .

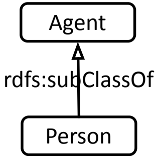
means that every instance of A is also an instance of B in every possible world, which is to say that the extension of class A is a subset of the extension of class B in every possible world.

So, rdfs:subClassOf captures the notion of subset between class extensions, as expected.

From a logical point of view, rdfs:subClassOf establishes an implication relation:

- the intension of a class *implies* the intension of any super-class of it
- the intension of a class *is implied by* the intension of any sub-class of it.

"resource that can do something and is an individual" evidently implies "resource that can do something"



This conforms to our intuition when we say:

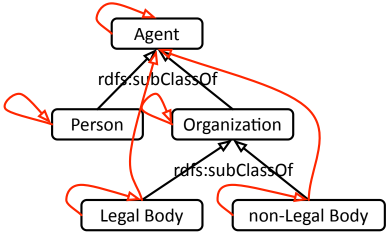
- every Male is a Human
- if a resource is a Male then it is also a Human

Logical consequences

The subset relation is reflexive and transitive:

(R) $A \subseteq A$
(T) $A \subseteq B,\ B \subseteq C \rightarrow A \subseteq C$

in every possible world.



If the black arrows

non-LegalBody rdfs:subClassOf Organization .
LegalBody rdfs:subClassOf Organization .
Organization rdfs:subClassOf Agent .
Person rdfs:subClassOf Agent .

are in the graph, then they are true in every possible world that satisfies the graph, therefore the red arrows are true in the same worlds.

The red arrows are logical consequences of the black arrows.

Logical consequences

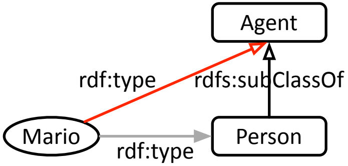
Moreover:

if a $\in$ A and A $\subseteq$ B then a $\in$ B

in every possible world.

If the black arrows

Mario rdf:type Person .
Person rdfs:subClassOf Agent .



are in the graph, then they are true in every possible world that satisfies the graph, therefore the red arrow is true in the same worlds.

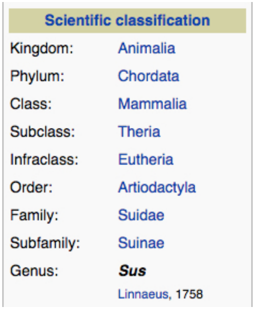
The red arrow is a logical consequence of the black arrows.

What IsA is not

The sub/super-class relation captures a very precise notion, that is scientific classification.

This notion **must not** be confused with other, similar but different notions, found in various branches of knowledge:

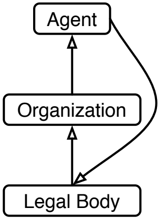
- lexical similarity: broader/narrower concept
- temporal, historical or geographical inclusion
- ...

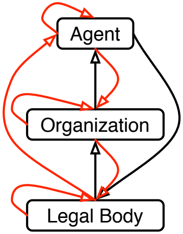


Scientific classification	
Kingdom:	Animalia
Phylum:	Chordata
Class:	Mammalia
Subclass:	Theria
Infraclass:	Eutheria
Order:	Artiodactyla
Family:	Suidae
Subfamily:	Suinae
Genus:	***Sus***
	Linnaeus, 1758

Cycles in taxonomies

What happens if there is a cycle in the class taxonomy?





For example, I may later discover that every Agent is also a Legal Body

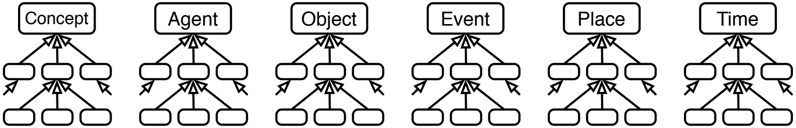
Let's compute the consequences. Result: every class is a sub-class of every class.

Every class in a loop is equivalent (or synonymous) to every other class. We may well replace all the classes in the loop by a single class and leave the rest of the taxonomy unchanged.

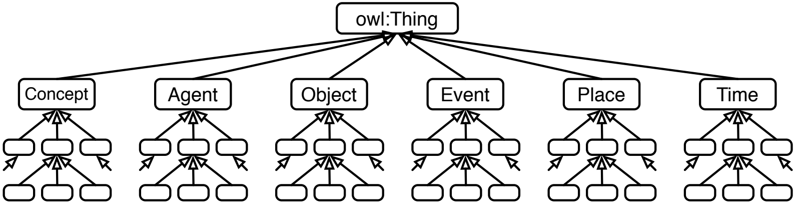
Is this really what we wanted?

Closing the taxonomy up

In general, a class taxonomy consists of one or more unconnected branches, each articulating one of the main independent classes.



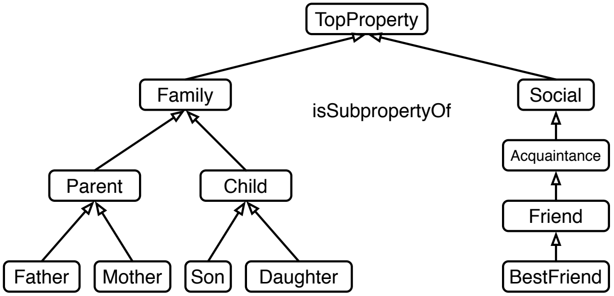
It is a good practice to close the class taxonomy up, by defining a super-class of the top classes, so that it is possible to programmatically explore the whole class space starting from this super-class. For example:



Property taxonomy

Also properties can be organized in a taxonomy, following the same principles as for classes. The property for expressing IsA between properties is: rdfs:subPropertyOf.

A property taxonomy helps organizing the space of the relationship types, by specializing the most general types.



The intension of the sub-property is a special case, or *refinement*, of the intension of the super-property. So the former implies the latter.

- the intension of Son is “male direct descendant”, while that of Child is “direct descendant”

The semantics of rdfs:subPropertyOf is the subset relation between property extensions:

- reflexive
- transitive
- **if** a P b . **and** P rdfs:subPropertyOf Q . **then** a Q b .

Cycles in property taxonomy behave exactly as cycles in the class taxonomy.

Property typing

The last abstraction mechanism that we consider is borrowed from Mathematics, and is called *property typing.*

The idea is that more some properties *always* combines instances of a certain class with instances of another, or the same class.

e.g.

- in a library, “is borrowed by” always combines a book and a person
- “happens” always combines an event and a time
- “at” always combines an event and a place

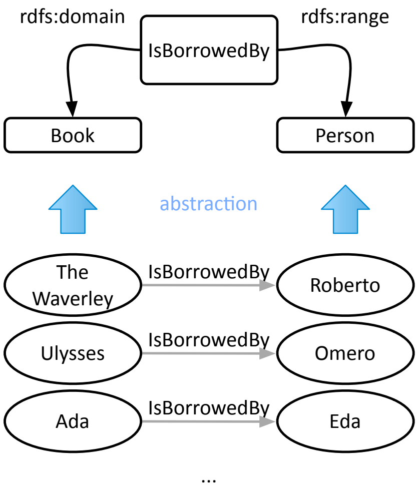
We can use these regularities to introduce another abstraction mechanism into our semantic net, aimed at preventing misuse of the property vocabulary.

This time we consider all relationships that are instances of the same property and suppress the difference between the combined objects, to consider only the classes they are instances of.

To do that, we borrow from mathematics the notions of *domain* and *range* and say:

- the *domain* of IsBorrowedBy is the class Book and its *range* is the class Person
- the *domain* of happens is the class Event and its *range* is the class Time
- the *domain* of at is the class Event and its *range* is the class Place

Notice that these sentences capture some part of the *meaning* of a property.



The RDF Schema vocabulary provides two IRIs for stating this:

IsBorrowedBy rdfs:domain Book .
IsBorrowedBy rdfs:range Person .

Semantics of rdfs:domain and rdfs:range

rdfs:domain and rdfs:range are properties: in every possible world, their extensions are binary relations. Which ones?

rdfs:domain relates a property with the class whose instances are the first kind of resources combined by the property. In every possible world,

- the extension of a property is a binary relation
- the extension of a class is a set

so every pair (R,S) in the extension of rdfs:domain is a binary relation R and a set S, such that the first column of R is a subset of S.

In other words,

P rdfs:domain C .

is true in a world w if and only if the domain (*i.e.*, the first column) of the extension of P is a subset of the extension of C.

Practically speaking, asserting the triple

IsBorrowedBy rdfs:domain Book .

in a graph, means to *state* that in every triple

b IsBorrowedBy p .

b is an instance of Book .

According to the semantics of rdfs:domain, if in a world w it is true that

IsBorrowedBy rdfs:domain Book .
b IsBorrowedBy p .

it is also true that

b rdf:type Book .

Therefore in every interpretation that “understands”rdfs:domain, the first two triples imply the last one.

Analogously, rdfs:domain relates a property with the class whose instances are the second kind of resources combined by the property. In every possible world,

- the extension of a property is a binary relation
- the extension of a class is a set

so every pair (R,S) in the extension of rdfs:domain is a binary relation R and a set S, such that the second column of R is a subset of S.

In other words,

P rdfs:range C .

is true in a world w if and only if the range (*i.e.*, the second column) of the extension of P is a subset of the extension of C.

Practically speaking, asserting the triple

IsBorrowedBy rdfs:range Person .

in a graph, means to *state* that in every triple

b IsBorrowedBy p .

p is an instance of Person .

According to the semantics of rdfs:range, if in a world w it is true that

IsBorrowedBy rdfs:range Person .
b IsBorrowedBy p .

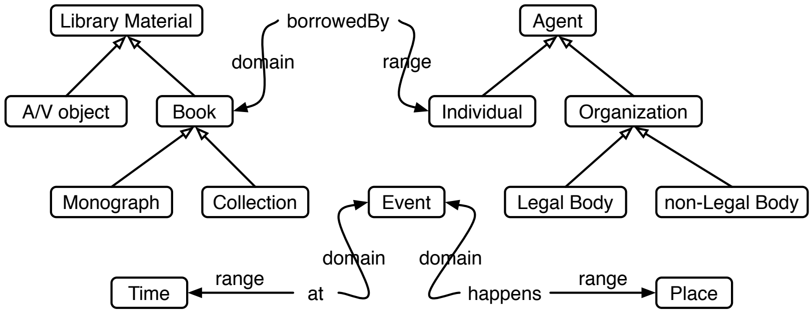
it is also true that

p rdf:type Person .

Therefore in every interpretation that "understands"rdfs:range, the first two triples imply the last one.

A look at our abstraction

Our abstraction has grown into a rather complex net:



We find in it:

1. classes and properties that represents in an abstract form the objects and the relationships in our discourse, and
2. a possibly large set of relationships that combine classes and properties and give important features of them:
    1. relationships of the isSubclassOf type, forming the class taxonomy
    2. relationships of the isSubcpropertyOf type, forming the property taxonomy
    3. relationships of the domain type
    4. relationships of the range type

From a linguistic point of view such a thing would be called a *dictionary:*

> *a book or electronic resource that lists the words of a language (typically in alphabetical order) and gives their meaning*

In computer science, this thing is called an *ontology:*

An ontology is a formal, explicit specification of a shared conceptualization.

- conceptualization: the <u>objects, concepts, and other entities</u> that are assumed to exist in some area of interest and <u>the relationships that hold among them</u>. A conceptualization is an abstract, simplified view of the world that we wish to represent for some purpose.
- explicit specification: a specification in a language providing:
    - names for representing *the objects, concepts, and other entities*
    - syntax for representing *the relationships that hold among them*
    - semantics for defining implicit knowledge
    - (possibly) an efficient algorithm for computing implicit knowledge

The ontology is a separated network from the one we started from.

An ontology captures *some of the intensional aspects of classes and properties,* its definitions are independent of a particular world:

- a Man will always be a Person in our conceptualization, even though in a particular state of the world there may be no (instances of class) Man.

On the other hand, the original semantic network is a model a particular reality, telling us what is true in this reality.

- Carlo is a Man, Pisa is in Tuscany, pizza is good, . . .

The ontology is said to hold *terminological* (or *lexical*) *knowledge.*

The semantic network is said to hold *assertional* (or *factual*) *knowledge.*

In Knowledge Representation, a KB is defined as a system with two components:

1. the TBox, or Terminological Box, holding the ontology
2. the ABox, or Assertional Box, holding the semantic net proper

This is just a convention, depending on the particular context of work.

What is important, is to understand the interaction between ontology and semantic net:

a semantic net holds the relationships that are instances of the properties in the ontology, and that are formed by using the objects that are instances of the classes in the ontology.

Conversely:

an ontology gives the definitions of the classes and the properties whose instances occur in the semantic net.

The RDF Schema Vocabulary

The RDF Schema Vocabulary (RDFSVoc) consists of a collection of IRIs of RDF resources that can be used to introduce abstraction mechanisms in RDF graphs.

The core vocabulary is defined in a namespace identified by the IRI

`http://www.w3.org/2000/01/rdf-schema#`

and is conventionally associated with the prefix `rdfs:`.

The IRIs in the RDFSVoc are of two kinds:

- classes and
- properties

Classes in RDFSVoc

`rdfs:Class`

This is the class of resources that are RDF classes, so it must be used to declare the classes in a graph. For instance

Book rdf:type rdfs:Class .

asserts that Book is a class.

`rdfs:Resource`

All things described by RDF are called resources, and rdfs:Resource is the class of everything.

- Everything is an instance of the class rdfs:Resource
- All other classes are subclasses of this class.

rdfs:Resource is an instance of rdfs:Class. Therefore:

rdfs:Resource rdf:type rdfs:Class .

is valid.

`rdfs:Literal`

The class rdfs:Literal is the class of literal values. Property values such as textual strings are examples of RDF literals.

`rdfs:Datatype`

rdfs:Datatype is the class of datatypes. It is somewhat similar to rdfs:Class, in that its instances are classes, so it is both an instance of and a subclass of rdfs:Class.

```
rdfs:Datatype rdf:type rdfs:Class .
rdfs:Datatype rdfs:subClassOf rdfs:Class .
```

Each instance of rdfs:Datatype is a subclass of rdfs:Literal.

Properties in RDFSVoc

`rdfs:subClassOf`

The property rdfs:subClassOf is an instance of rdf:Property that is used to state that one class is a sub-class of another class.

```
rdfs:subClassOf rdf:type rdf:Property .
rdfs:subClassOf rdfs:domain rdfs:Class .
rdfs:subClassOf rdfs:range rdfs:Class .
```

`rdfs:subPropertyOf`

The property rdfs:subPropertyOf is an instance of rdf:Property that is used to state that one property is a sub-property of another property.

```
rdfs:subPropertyOf rdf:type rdf:Property .
rdfs:subPropertyOf rdfs:domain rdf:Property .
rdfs:subPropertyOf rdfs:range rdf:Property .
```

rdfs:domain

rdfs:domain is an instance of rdf:Property that is used to state that a class is a domain of a property.

```
rdfs:domain rdf:type rdf:Property .
rdfs:domain rdfs:domain rdf:Property .
rdfs:domain rdfs:range rdfs:Class .
```

Notice:

```
P rdfs:domain Person .
P rdfs:domain MusicLover .
carlo P r .
```

RDFS entail that the resource carlo is an instance of **both** Person and MusicLover, *i.e.*

```
carlo rdf:type Person .
carlo rdf:type MusicLover .
```

rdfs:range

rdfs:range is an instance of rdf:Property that is used to state that a class is a range of a property.

```
rdfs:range rdf:type rdf:Property .
rdfs:range rdfs:domain rdf:Property .
rdfs:range rdfs:range rdfs:Class .
```

Notice:

```
P rdfs:range Person .
P rdfs:range MusicLover .
r P carlo .
```

RDFS entail that the resource carlo is an instance of **both** Person and MusicLover, *i.e.*

```
carlo rdf:type Person .
carlo rdf:type MusicLover .
```

rdfs:label

rdfs:label is an instance of rdf:Property that may be used to provide a human-readable version of a resource's name.

```
rdfs:label rdf:type rdf:Property .
rdfs:label rdfs:domain rdfs:Resource .
rdfs:label rdfs:range rdfs:Literal .
```

For instance,

```
crm:E1_CRM_Entity rdfs:label “CRM Entity”@en .
crm:E1_CRM_Entity rdfs:label “Entité CRM”@fr .
crm:E1_CRM_Entity rdfs:label “οντότιτα”@el .
```

Multilingual labels are supported using the language tagging facility of RDF literals.

rdfs:comment

rdfs:comment is an instance of rdf:Property that may be used to provide a human-readable description of a resource.

```
rdfs:comment rdf:type rdf:Property .
rdfs:comment rdfs:domain rdfs:Resource .
rdfs:comment rdfs:range rdfs:Literal .
```

```
crm:E1_CRM_Entity rdfs:comment "This class comprises all things in the
universe of discourse of the CIDOC Conceptual Reference Model."@en .
```

Multilingual documentation is supported through use of the language tagging facility of RDF literals.

Utility properties

`rdfs:seeAlso`

rdfs:seeAlso is an instance of rdf:Property that is used to indicate a resource that might provide additional information about the subject resource.

```
rdfs:seeAlso rdf:type rdf:Property .
rdfs:seeAlso rdfs:domain rdfs:Resource .
rdfs:seeAlso rdfs:range rdfs:Resource .
```

`rdfs:isDefinedBy`

rdfs:isDefinedBy is an instance of rdf:Property that is used to indicate a resource defining the subject resource.

```
rdfs:isDefinedBy rdf:type rdf:Property .
rdfs:isDefinedBy rdfs:domain rdfs:Resource .
rdfs:isDefinedBy rdfs:range rdfs:Resource .
rdfs:isDefinedBy rdfs:subPropertyOf rdfs:seeAlso .
```

RDFS entailment patterns

How to decide RDF or RDFS entailment between two graphs G and E (recognizing D)?

An obvious idea, then could be the following:

1. to exapnd the graph G by adding the implicit triples, thus obtaining a graph $G^\star$
2. to test whether $G^\star$ simply entails E.

This idea of the expansion is not new: it comes from logic and it is applied in databases.

It is also doable, and we now see how it can be applied to RDFS entailment.

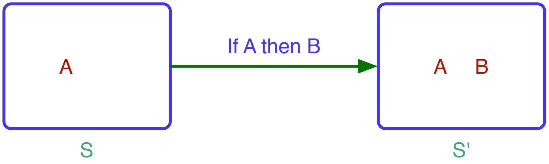
The basic tools are *entailment patterns.*

These entailment patterns can be viewed as rules of the form

if A then B

where A is the premise of the rule, and B is the conclusion.

To apply a rule to a graph means: if the graph includes the premise of the rule (A), then add the conclusion (B) to the graph.



The rules correspond closely to the RDFS semantic conditions:

	If S contains:	**then S RDFS entails recognizing D:**
rdfs1	any IRI aaa in D	aaa rdf:type rdfs:Datatype .
rdfs2	aaa rdfs:domain xxx . yyy aaa zzz .	yyy rdf:type xxx .
rdfs3	aaa rdfs:range xxx . yyy aaa zzz .	zzz rdf:type xxx .
rdfs4a	xxx aaa yyy .	xxx rdf:type rdfs:Resource .
rdfs4b	xxx aaa yyy .	yyy rdf:type rdfs:Resource .
rdfs5	xxx rdfs:subPropertyOf yyy . yyy rdfs:subPropertyOf zzz .	xxx rdfs:subPropertyOf zzz .
rdfs6	xxx rdf:type rdf:Property .	xxx rdfs:subPropertyOf xxx .
rdfs7	xxx aaa yyy . aaa rdfs:subPropertyOf bbb .	xxx bbb yyy .

	If S contains:	**then S RDFS entails recognizing D:**
rdfs8	xxx rdf:type rdfs:Class .	xxx rdfs:subClassOf rdfs:Resource .
rdfs9	xxx rdfs:subClassOf yyy . zzz rdf:type xxx .	zzz rdf:type yyy .
rdfs10	xxx rdf:type rdfs:Class .	xxx rdfs:subClassOf xxx .
rdfs11	xxx rdfs:subClassOf yyy . yyy rdfs:subClassOf zzz .	xxx rdfs:subClassOf zzz .
rdfs12	xxx rdf:type rdfs:ContainerMembershipProperty .	xxx rdfs:subPropertyOf rdfs:member .
rdfs13	xxx rdf:type rdfs:Datatype .	xxx rdfs:subClassOf rdfs:Literal .

The general algorithm to check RDF (or RDFS) entailment between graphs S and E, consists of the following sequence of steps:

1. Add to S all the RDF (or RDF and RDFS) axiomatic triples except those containing the container membership property IRIs rdf:_1, rdf:_2, ...
2. For every container membership property IRI which occurs in E, add the RDF (or RDF and RDFS) axiomatic triples which contain that IRI.
3. Apply the RDF (or RDF and RDFS) inference patterns as rules, adding each conclusion to the graph, to exhaustion; that is, until they generate no new triples.
4. If there is some instance of E which is a subset of the so obtained graph, then "yes". Otherwise, "no".

This process is clearly correct, in that if it gives a positive result then indeed S does RDF (RDFS) entail E.

It is also complete: it discovers entailment between two graphs whenever it exists.

To obtain this result, the syntax of RDF must be generalized, allowing blank nodes as predicates of triples.

The closures are finite. The generation process is decidable and of polynomial complexity. Detecting simple entailment is NP-complete in general, but of low polynomial order when E contains no blank nodes.

Every RDF(S) closure, even starting with the empty graph, will contain all RDF(S) tautologies which can be expressed using the vocabulary of the original graph plus the RDF and RDFS vocabularies. In practice there is little utility in re-deriving these, and a subset of the rules can be used to establish most entailments of practical interest.

Detecting datatype entailment for larger sets of datatype IRIs requires attention to idiosyncratic properties of the particular datatypes.

Conclusions

RDF Schema provides a vocabulary for expressing simple ontologies in RDF.

The semantics of this vocabulary captures the meaning of abstraction mechanisms, well-known knowledge organization tools in science.

RDFS entailment relies on the semantics to infer knowledge implicit in RDFS graphs.

By generalizing the syntax of RDF (allowing blank nodes as predicates and literals as subjects) a sound and complete calculus can be used to expand a graph into its closure, including all implicit knowledge involving the RDF and the RDFS vocabularies.

The graph can be generated efficiently.

Detecting simple entailment is NP-complete but of low polynomial order when E contains no blank nodes.

RDF Schema is largely employed in practical applications.

Most available technologies implement its semantics and its entailment relation.

It has severe limitations in capturing the intension of classes and properties:

- no negation
- no disjunction
- no numerical restrictions
- no quantifiers
- . . .

To overcome these limitations, we must look into OWL.

However, efficiency will be lost (almost) forever.

Useful readings

- Dan Brickley, R. V. Guha. RDF Schema 1.1. W3C Recommendation, 25 February 2014. URL: http://www.w3.org/TR/rdf-schema/
- Patrick J. Hayes, Peter F. Patel-Schneider. RDF 1.1 Semantics. W3C Recommendation, 25 February 2014. URL:http://www.w3.org/TR/rdf11-mt/
- Pascal Hitzler, Markus Krtzsch, Sebastian Rudolph. Foundations of Semantic Web Technologies. CRC Press 2010 [ch. 2 & 3].
- Herman J. ter Horst. Completeness, Decidability and Complexity of Entailment for RDF Schema and a Semantic Extension Involving the OWL Vocabulary. Journal of Web Semantics 3 (2005) 79-115.