



An Introduction to RDFS



V. CHRISTOPHIDES

Department of Computer Science
University of Crete
ICS - FORTH, Heraklion, Crete

1



RDF Schema

- The **RDF Vocabulary Description Language 1.0** (or **RDF Schema** or **RDFS**) is a language that can be used to define the **vocabulary** (i.e., the **terms**) to be used in an RDF graph
- **RDFS** is used to indicate that we are describing specific **kinds** of resources, and will use specific **properties** in describing those resources
- The **RDFS concepts** (classes, properties) are themselves provided in the form of an **RDFS vocabulary**
 - ◆ that is, as a specialized set of predefined RDF resources with their own special meanings
 - ◆ having URIs with the prefix `http://www.w3.org/2000/01/rdf-schema#` (associated with the QName prefix **rdfs:**)
- **RDFS** is a **simple ontology definition language** (compared to OWL) in which we can only define **taxonomies** and do some **basic inference** about them
- Vocabulary descriptions (schemas, ontologies) written in the RDF Schema language are **legal RDF graphs**
 - ◆ In other words, we use **RDF to represent RDFS information**

2



Defining Classes

- A basic step in any kind of description process is identifying the various **kinds of things** to be described
 - ◆ RDF Schema refers to these “kinds of things” as **classes**
- A **class** in RDF Schema corresponds to the generic concept of a **type** or **category**, somewhat like the notion of a class in object-oriented data models but with more flexible structural constraints
- In RDFS, a class **C** is defined by a triple of the form


```
C rdf:type rdfs:Class .
```

 using the predefined class **rdfs:Class** and the predefined property **rdf:type**
- **Example:** an organization `example.org` wants to use RDF Schema to provide information about motor vehicles, vans and trucks.
 - ◆ To define classes that represent these vehicle categories, we write the following statements (triples):


```
ex:MotorVehicle rdf:type rdfs:Class .
ex:Van rdf:type rdfs:Class .
ex:Truck rdf:type rdfs:Class .
```

3



Defining Instances

- Now suppose `example.org` wants to define an individual car (e.g., the company car) and say that it is a motor vehicle
- This can be done with the following RDF statement:


```
exthings:companyCar rdf:type ex:MotorVehicle .
```
- The predefined property **rdf:type** is used as a predicate in a statement


```
I rdf:type C .
```

 to declare that **individual I** is an **instance of class C**
- In statements of the form


```
C rdf:type rdfs:Class .
```

rdf:type is used to declare that **class C** (viewed as an individual object) is an **instance of the predefined class rdfs:Class**
- **Defining a class explicitly is optional**; if we write the triple


```
I rdf:type C .
```

 then **C** is **inferred** to be a class (an instance of **rdfs:Class**) in RDFS

4



Defining Subclasses

- Now suppose example.org wants to define that vans and trucks are **specialized kinds** of motor vehicle
- This can be done with the following RDF statements:

```
ex:Van    rdfs:subClassOf  ex:MotorVehicle .  
ex:Truck  rdfs:subClassOf  ex:MotorVehicle .
```
- The predefined property `rdfs:subClassOf` is used as a predicate in a statement to declare that **a class is a specialization of another more general class**
- The **meaning** of the predefined property `rdfs:subClassOf` in a statement of the form

```
C1 rdfs:subClassOf C2
```

is that any instance of class **C1** is also an instance of class **C2**
- **Example:** If we have the statements

```
ex:Van    rdfs:subClassOf  ex:MotorVehicle .  
exthings:myCar rdf:type ex:Van .
```

then RDFS allows us to **infer** the statement

```
exthings:myCar rdf:type ex:MotorVehicle .
```

5



Properties of `rdfs:subClassOf`

- The `rdfs:subClassOf` property is **reflexive** and **transitive**
- **Examples:**
 - ♦ If we have a class `ex:MotorVehicle` then RDFS allows us to **infer** the statement

```
ex:MotorVehicle rdfs:subClassOf ex:MotorVehicle .
```
 - ♦ If we have the statements

```
ex:Van    rdfs:subClassOf ex:MotorVehicle .  
ex:MiniVan rdfs:subClassOf ex:Van .
```

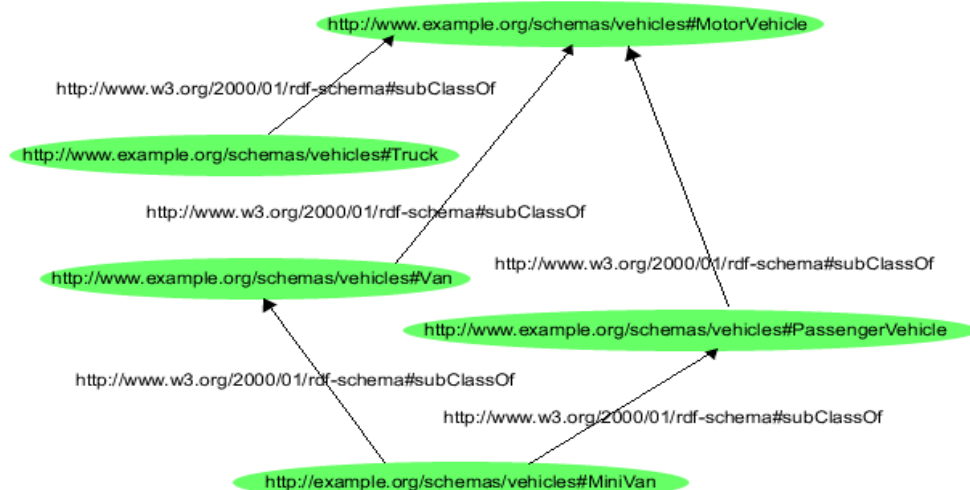
then RDFS allows us to **infer** the statement

```
ex:MiniVan rdfs:subClassOf ex:MotorVehicle .
```

6



Class and Instance Information as a Graph



A class can be a specialization of multiple superclasses (e.g., the graph defined by the `rdfs:subClassOf` property is a directed graph not a tree) ⁷



The Graph in Triple Notation

```
ex:MotorVehicle rdfs:type rdfs:Class .
ex:PassengerVehicle rdfs:type rdfs:Class .
ex:Van rdfs:type rdfs:Class .
ex:Truck rdfs:type rdfs:Class .
ex:MiniVan rdfs:type rdfs:Class .
```

```
ex:PassengerVehicle rdfs:subClassOf ex:MotorVehicle .
ex:Van rdfs:subClassOf ex:MotorVehicle .
ex:Truck rdfs:subClassOf ex:MotorVehicle .
```

```
ex:MiniVan rdfs:subClassOf ex:Van .
ex:MiniVan rdfs:subClassOf ex:PassengerVehicle .
```



Defining Properties

- In addition to defining the specific classes of things they want to describe, user communities also need to be able to define specific **properties** that characterize those classes of things
- A property can be defined by stating that it is an instance of the predefined class **rdf:Property**
- Example:

```
ex:author rdf:type rdf:Property .
```
- Then, property **ex:author** can be used as a **predicate in an RDF triple** such as the following:

```
ex:john ex:author ex:book123 .
```
- **Defining a property explicitly is optional**; if we write the RDF triple

```
S P O .
```

then **P** is **inferred** to be a property by RDFS

9



Properties (cont'd)

- **Properties are resources too** (this makes RDF and RDFS different than many other KR formalisms)
 - ◆ Therefore, **properties can appear as subjects or objects of triples**
- Example (provenance):

```
ex:author prov:definedBy ke:john  
ke:john prov:defined ex:author
```
- In RDFS property definitions are **independent** of class definitions
 - ◆ In other words, a property definition can be made without any reference to a class
- Optionally, properties can be declared to apply to certain instances of classes by defining their **domain** and **range**

10



Domain and Range

- RDFS provides vocabulary for describing **how properties and classes are intended to be used together** in RDF data
 - ◆ The **rdfs:domain** predicate can be used to indicate that a particular property applies to instances of a designated class (i.e., it defines the **domain** of the property)
 - ◆ The **rdfs:range** predicate is used to indicate that the values of a particular property are instances of a designated class (i.e., it defines the **range** of the property)
- For a property, we can have **zero, one, or more than one** domain or range statements
- **Example:**
 - ex:Book **rdf:type** **rdfs:Class** .
 - ex:Person **rdf:type** **rdfs:Class** .
 - ex:author **rdf:type** **rdf:Property** .
 - ex:author **rdfs:domain** ex:Book .
 - ex:author **rdfs:range** ex:Person .

11



Domain and Range (cont'd)

- **No domain or no range statement:** If no range statement has been made for property P, then **nothing has been said about the values** of this property (similarly for no domain statement)
- **Example:** If we have only the triple
exstaff:frank ex:hasMother exstaff:frances .
then nothing can be inferred from it regarding resources
exstaff:frank and exstaff:frances.
- **One domain statement:** If we have
P **rdfs:domain** D .
then we can **infer** that when P is applied to some resource, this resource is an instance of class D
- **One range statement:** If we have
P **rdfs:range** R .
then we can **infer** that when P is applied to some resource, the value of P is an instance of class R

12



Examples

- If we have

```
ex:hasMother rdfs:domain ex:Person .
exstaff:frank ex:hasMother exstaff:frances .
```

then we can infer:

```
exstaff:frank rdf:type ex:Person.
```
- If we have

```
ex:hasMother rdfs:range ex:Person .
exstaff:frank ex:hasMother exstaff:frances .
```

then we can infer

```
exstaff:frances rdf:type ex:Person.
```

13



Domain and Range (cont'd)

- **Two domain or range statements:** If we have

```
P rdfs:range C1 .
P rdfs:range C2 .
```

then we can **infer** that the values of property P are instances of both C1 and C2. Similarly, for two domain statements.
- **Example:** If we have

```
ex:hasMother rdfs:range ex:Female .
ex:hasMother rdfs:range ex:Person .
exstaff:frank ex:hasMother exstaff:frances .
```

then we can infer that exstaff:frances is an instance of both ex:Female and ex:Person

14



Another Example

```
ex:Human rdf:type rdfs:Class .
ex:hasParent rdf:type rdf:Property .
ex:hasParent rdfs:domain ex:Human .
ex:hasParent rdfs:range ex:Human .
ex:Tiger rdf:type rdfs:Class .
ex:hasParent rdfs:domain ex:Tiger .
ex:hasParent rdfs:range ex:Tiger .
ex:tina ex:hasParent ex:john .
```

- What new triples can we infer from the above? Is anything wrong?
 - ◆ Tina and John are inferred to be both humans and tigers. There is nothing wrong with this! We have not said anywhere that humans cannot be tigers and vice versa (nor can we say this in RDFS)
 - ◆ However, one might not want to do this kind of modeling in RDFS if Tina and John were meant to be humans

15



Datatypes for Ranges

- The **rdfs:range** property can also be used to indicate that the value of a property is given by a **typed literal**
- **Example:**

```
ex:age rdf:type rdf:Property .
ex:age rdfs:range xsd:integer .
```
- Optionally, we can also assert that **xsd:integer** is a **datatype** as follows:

```
xsd:integer rdf:type rdfs:Datatype .
```

16



Specializing Properties

- RDF Schema provides a way to **specialize properties** (similarly with classes)
 - ◆ This specialization relationship between two properties is described using the predefined property **`rdfs:subPropertyOf`**
- **Example:**

```
ex:driver rdfs:type rdfs:Property .
ex:primaryDriver rdfs:type rdfs:Property .
ex:primaryDriver rdfs:subPropertyOf ex:driver .
```
- If resources S and O are connected by the property P1 and P1 is a subproperty of property P2, then RDFS allows us to **infer** that S and O are also connected by the property P2
- **Example:** If we have the statements

```
ex:john ex:primaryDriver ex:companyCar
ex:primaryDriver rdfs:subPropertyOf ex:driver .
```

then we can infer

```
ex:john ex:driver ex:companyCar .
```

17



Specializing Properties (cont'd)

- **`rdfs:subPropertyOf`** is **reflexive** and **transitive**
- **Examples:**
 - ◆ If we have the property `ex:driver` then RDFS allows to infer the triple

```
ex:driver rdfs:subPropertyOf ex:driver .
```
 - ◆ If we have the triples

```
ex:primaryDriver rdfs:subPropertyOf ex:driver .
ex:driver rdfs:subPropertyOf ex:isResponsibleFor .
```

then RDFS allows us to **infer** the triple

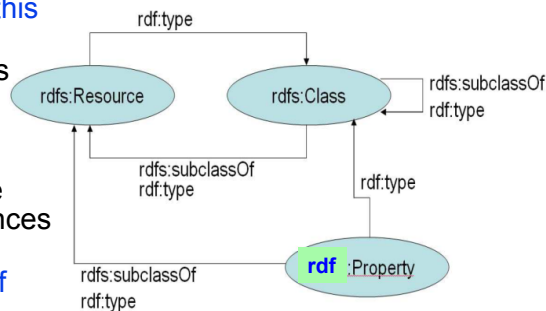
```
ex:primaryDriver rdfs:subPropertyOf
ex:isResponsibleFor .
```

18



RDF Schema Predefined Classes

- The group of resources that are RDF Schema classes is itself a class called `rdfs:Class`
 - ♦ All classes are instances of this class
- In the literature, classes such as `rdfs:Class` that have other classes as instances are called **meta-classes**
- All things described by RDF are called **resources**, and are instances of the class `rdfs:Resource`
- `rdfs:Resource` is the class of everything
 - ♦ All other classes are subclasses of this class. For example, `rdfs:Class` is a subclass of `rdfs:Resource`



19



Important (Tricky?) Details

- A class may be a member of its own class extension (i.e., an instance of itself)
Example: `rdfs:Class rdfs:type rdfs:Class` .
- A property may be applied to itself
Example: `rdfs:domain rdfs:domain rdfs:Property` .
- The semantics of RDF and RDFS are formalized appropriately so that we do not have problems with these features

20



Some Utility Properties

- `rdfs:label`
- `rdfs:comment`
- `rdfs:seeAlso`
- `rdfs:isDefinedBy`

21



The Property `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**
- The `rdfs:domain` of `rdfs:label` is `rdfs:Resource`
 - ◆ The `rdfs:range` of `rdfs:label` is `rdfs:Literal`
- Multilingual labels are supported using the **language tagging** facility of RDF literals

22



The Property `rdfs:comment`

- `rdfs:comment` is an instance of `rdf:Property` that may be used to provide a human-readable description of a resource
- The `rdfs:domain` of `rdfs:comment` is `rdfs:Resource`
 - ◆ The `rdfs:range` of `rdfs:comment` is `rdfs:Literal`
- Multilingual documentation is supported through use of the language tagging facility of RDF literals

23



The Property `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
- A triple of the form `S rdfs:seeAlso O` states that the resource `O` may provide additional information about `S`
 - ◆ It may be possible to retrieve representations of `O` from the Web, but this is not required. When such representations may be retrieved, no constraints are placed on the format of those representations
- The `rdfs:domain` of `rdfs:seeAlso` is `rdfs:Resource`
 - ◆ The `rdfs:range` of `rdfs:seeAlso` is `rdfs:Resource`

24



The Property `rdfs:isDefinedBy`

- `rdfs:isDefinedBy` is an instance of `rdf:Property` that is used to indicate a resource defining the subject resource
 - ◆ This property may be used to indicate an RDF vocabulary in which a resource is described
- A triple of the form `S rdfs:isDefinedBy O` states that the resource `O` defines `S`
 - ◆ It may be possible to retrieve representations of `O` from the Web, but this is not required. When such representations may be retrieved, no constraints are placed on the format of those representations.
 - ◆ `rdfs:isDefinedBy` is a subproperty of `rdfs:seeAlso`
- The `rdfs:domain` of `rdfs:isDefinedBy` is `rdfs:Resource`
 - ◆ The `rdfs:range` of `rdfs:isDefinedBy` is `rdfs:Resource`

25



RDFS vs. Types in OO Languages and Data Models

- The **scope** of an attribute description in most programming languages is **restricted to the class** or **type** in which it is defined
 - ◆ In RDFS, on the other hand, property definitions are, by default, **independent** of class definitions, and have, by default, **global** scope (although they may optionally be declared to apply only to certain classes using domain and range specifications)
- Since they are resources, properties are **first-class citizens** in RDF. But notice the following **asymmetry**:
 - ◆ The class `rdf:Property` has as instances all properties
 - ◆ There is no top class for the `rdfs:subPropertyOf` relationship e.g., `rdfs:topProperty` (unlike `rdfs:Resource` which is the top class for the `rdfs:subClassOf` relationship)
- Depending on how an application interprets the property descriptions, a description of an instance might be considered **valid** either **without** some of the schema-specified properties or with **additional** properties
 - ◆ There might be an instance of `ex:Book` without an `ex:author`, even if `ex:author` is described as having a domain of `ex:Book`
 - ◆ There might be an instance of `ex:Book` with an `ex:technicalEditor`, even though the schema describing class `ex:Book` does not describe such a property

26



RDFS vs. Types (cont'd)

- **Benefits of the RDF approach:** One can start with a property definition and then extend it to other uses that might not have been anticipated
- **Shortcoming:** In RDFS, it is not possible to say, for example, that if the property `ex:hasParent` is used to describe a resource of class `ex:Human`, then the range of the property is also a resource of class `ex:Human`, while if the property is used to describe a resource of class `ex:Tiger`, then the range of the property is also a resource of class `ex:Tiger`. **This can be done in ontology languages!**
- RDF Schema descriptions are not **prescriptive** in the way programming language type declarations typically are
- **Example:** If a programming language declares a class `Book` with an author attribute having values of type `Person`, this is usually interpreted as a group of **constraints**
 - ◆ The language will not allow the creation of an instance of `Book` without an author attribute
 - ◆ The language will not allow an instance of `Book` with an author attribute that does not have a `Person` as its value
 - ◆ If author is the **only** attribute defined for class `Book`, the language will not allow an instance of `Book` with some other attribute

27



RDFS vs. Types (cont'd)

- RDF Schema provides schema information as additional **descriptions** of resources, **but does not prescribe how these descriptions should be used by an application**
 - ◆ RDF Schema only allows us to **infer new triples** as we specified earlier
- This schema information might be used in different ways by an application:
 - ◆ As a **constraint** in the same way that a programming language might: it will ensure that any `ex:author` property has a value of the `ex:Person` class
 - But this functionality needs to be developed by the application itself!
 - ◆ As **additional information** about the data it receives: if it receives some RDF data that includes an `ex:author` property whose value is a resource of unspecified class, it can use the schema-provided statement to **infer** that the resource must be an instance of class `ex:Person`
 - In this case, the inference functionality is offered by any RDF Schema implementation

28



Richer Schema Languages

- RDF Schema provides **basic capabilities** for describing RDF vocabularies, but additional capabilities are also possible, and can be useful
- **Richer schema capabilities** that have been identified as useful and are provided in ontology languages such as OWL include:
 - ◆ **cardinality constraints** on properties, e.g., that a Person has **exactly one** father
 - ◆ specifying that a given property (such as `ex:hasAncestor`) is **transitive**, e.g., that if `A ex:hasAncestor B`, and `B ex:hasAncestor C`, then `A ex:hasAncestor C`
 - ◆ specifying that a given property is a unique **key** for instances of a particular class
 - ◆ specifying that two classes with different URIs actually represent the same class
 - ◆ specifying that two instances with different URIs actually represent the same individual
 - ◆ specifying **constraints on the range or cardinality of a property that depend on the class to which a property is applied**, e.g., being able to say that for a soccer team the `ex:hasPlayers` property has 11 values, while for a basketball team the same property should have only 5 values
 - ◆ the ability to describe **new classes in terms of combinations** (e.g., unions and intersections) of others, or to say that two classes are disjoint
 - ◆ The ability to specify **domain and range restrictions** for properties when they are used with a certain class

29



RDF(S) Semantics

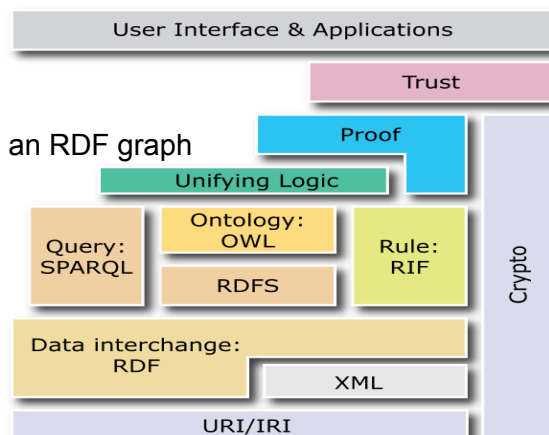
- RDF(S) vocabulary has built-in “meaning”

- **RDF(S) Semantics**

- ◆ Makes meaning explicit
- ◆ Defines what follows from an RDF graph

- **Semantic notions**

- ◆ Subgraph
- ◆ Instance
- ◆ Entailment



30



Subgraph

- E is a **subgraph** of S if and only if E predicates are a subset of S predicates
 - ◆ $\langle \#john, \#hasName, _ :johnsname \rangle$
 - ◆ $\langle _ :johnsname, \#firstName, "John"^{xsd:string} \rangle$
 - ◆ $\langle _ :johnsname, \#lastName, "Smith"^{xsd:string} \rangle$
- Subgraphs:
 - ◆ $\langle \#john, \#hasName, _ :johnsname \rangle$
 - ◆ $\langle _ :johnsname, \#firstName, "John"^{xsd:string} \rangle$
 - ◆ $\langle _ :johnsname, \#firstName, "John"^{xsd:string} \rangle$
 - ◆ $\langle _ :johnsname, \#lastName, "Smith"^{xsd:string} \rangle$
 - ◆ $\langle \#john, \#hasName, _ :johnsname \rangle$

31



Instance

- S' is an **instance** of S if and only if some blank nodes in S are replaced with blank nodes, literals or URIs
 - ◆ $\langle \#john, \#hasName, _ :johnsname \rangle$
 - ◆ $\langle _ :johnsname, \#firstName, "John"^{xsd:string} \rangle$
 - ◆ $\langle _ :johnsname, \#lastName, "Smith"^{xsd:string} \rangle$
- Instances:
 - ◆ $\langle \#john, \#hasName, \#abc \rangle$
 - ◆ $\langle \#abc, \#firstName, "John"^{xsd:string} \rangle$
 - ◆ $\langle \#abc, \#lastName, "Smith"^{xsd:string} \rangle$
 - ◆ $\langle \#john, \#hasName, _ :X \rangle$
 - ◆ $\langle _ :X, \#firstName, "John"^{xsd:string} \rangle$
 - ◆ $\langle _ :X, \#lastName, "Smith"^{xsd:string} \rangle$
 - ◆ $\langle \#john, \#hasName, _ :johnsname \rangle$
 - ◆ $\langle _ :johnsname, \#firstName, "John"^{xsd:string} \rangle$
 - ◆ $\langle _ :johnsname, \#lastName, "Smith"^{xsd:string} \rangle$
- Every graph is an instance of itself!

32



Entailment

- **Entailment** or logical implication is a relation between sentences of a formal language
- S entails E if E logically follows from S
 - ◆ Written: $S \models E$
- A graph entails all its subgraphs
 - ◆ If S' is a subgraph of S : $S \models S'$
- All instances of a graph S entail S
 - ◆ If S'' is an instance of S : $S'' \models S$

33



RDFS Entailment

```
<http://example.org/#john> rdf:type <http://example.org/#Student>
<http://example.org/#Student> rdfs:subClassOf <http://example.org/#Person>
```

entails

```
<http://example.org/#john> rdf:type <http://example.org/#Person>
```

```
<http://example.org/#hasName> rdfs:domain <http://example.org/#Student>
<http://example.org/#mary> <http://example.org/#hasName> "Mary"
```

entails

```
<http://example.org/#mary> rdf:type <http://example.org/#Student>
```

```
<http://example.org/#john> <http://example.org/#hasMother> <http://example.org/#mary>
<http://example.org/#hasMother> rdfs:subPropertyOf <http://example.org/#hasParent>
```

entails

```
<http://example.org/#john> <http://example.org/#hasParent> <http://example.org/#mary>
```

34



RDFS Entailment

```
<http://example.org/#john> rdf:type <http://example.org/#Student>  
<http://example.org/#Student> rdfs:subClassOf <http://example.org/#Person>
```

entails

```
<http://example.org/#john> rdf:type <http://example.org/#Person>
```

```
<http://example.org/#hasName> rdfs:domain <http://example.org/#Student>  
<http://example.org/#mary> <http://example.org/#hasName> "Mary"
```

entails

```
<http://example.org/#mary> rdf:type <http://example.org/#Student>
```

```
<http://example.org/#john> <http://example.org/#hasMother> <http://example.org/#mary>  
<http://example.org/#hasMother> rdfs:subPropertyOf <http://example.org/#hasParent>
```

entails

```
<http://example.org/#john> <http://example.org/#hasParent> <http://example.org/#mary>
```

35



RDFS Entailment

```
<http://example.org/#john> rdf:type <http://example.org/#Student>  
<http://example.org/#Student> rdfs:subClassOf <http://example.org/#Person>
```

entails

```
<http://example.org/#john> rdf:type <http://example.org/#Person>
```

```
<http://example.org/#hasName> rdfs:domain <http://example.org/#Student>  
<http://example.org/#mary> <http://example.org/#hasName> "Mary"
```

entails

```
<http://example.org/#mary> rdf:type <http://example.org/#Student>
```

```
<http://example.org/#john> <http://example.org/#hasMother> <http://example.org/#mary>  
<http://example.org/#hasMother> rdfs:subPropertyOf <http://example.org/#hasParent>
```

entails

```
<http://example.org/#john> <http://example.org/#hasParent> <http://example.org/#mary>
```

36



Entailment Rules

- Semantics defined through *entailment rules*:
 - ◆ If S contains `<triple pattern>` then add `<triple>`
- Executing all entailment rules yields *realization* of S
 - ◆ S entails E if E is a subgraph of the realization of S
- Axiomatic triple are always added
- if E contains `<A, B, C>` then add `<B, rdf:type, rdf:Property>`
- if E contains `<A, B, I>` (I is a valid XML literal) then add `<_:X, rdf:type, rdf:XMLLiteral>`
where `_:X` identifies to blank node allocated to I

37



RDFS Entailment

- everything in the subject is a resource
 - ◆ if E contains `<A, B, C>` then add `<A, rdf:type, rdfs:Resource>`
- every non-literal in the object is a resource
 - ◆ if E contains `<A, B, C>` (C is not a literal) then add `<C, rdf:type, rdfs:Resource>`
- every class is subclass of `rdfs:Resource`
 - ◆ if E contains `<A, rdf:type, rdfs:Class>` then add `<A, rdfs:subClassOf, rdfs:Resource>`
- inheritance:
 - ◆ if E contains `<A, rdf:type, B>`, `<B, rdfs:subClassOf, C>` then add `<A, rdf:type, C>`
- `rdfs:subClassOf` is transitive
 - ◆ if E contains `<A, rdfs:subClassOf, B>`, `<B, rdfs:subClassOf, C>` then add `<A, rdfs:subClassOf, C>`

38



RDFS Entailment (cont'd)

- **rdfs:subClassOf** is reflexive
 - ◆ if E contains $\langle A, \text{rdf:type}, \text{rdfs:Class} \rangle$ then add $\langle A, \text{rdfs:subClassOf}, A \rangle$
- **rdfs:subPropertyOf** is transitive
 - ◆ if E contains $\langle A, \text{rdfs:subPropertyOf}, B \rangle$, $\langle B, \text{rdfs:subPropertyOf}, C \rangle$ then add $\langle A, \text{rdfs:subPropertyOf}, C \rangle$
- **rdfs:subPropertyOf** is reflexive
 - ◆ if E contains $\langle P, \text{rdf:type}, \text{rdf:Property} \rangle$ then add $\langle P, \text{rdfs:subPropertyOf}, P \rangle$
- domain of properties
 - ◆ if E contains $\langle P, \text{rdfs:domain}, C \rangle$, $\langle A, P, B \rangle$ then add $\langle A, \text{rdf:type}, C \rangle$
- range of properties
 - ◆ if E contains $\langle P, \text{rdfs:range}, C \rangle$, $\langle A, P, B \rangle$ then add $\langle B, \text{rdf:type}, C \rangle$
- every literal is a member of **rdfs:Literal**
 - ◆ if E contains $\langle A, B, l \rangle$ (l is a plain literal) then add $\langle _ : X, \text{rdf:type}, \text{rdfs:Literal} \rangle$
- every datatype is subclass of **rdfs:Literal**
 - ◆ if E contains $\langle A, \text{rdf:type}, \text{rdfs:Datatype} \rangle$ then add $\langle A, \text{rdfs:subClassOf}, \text{rdfs:Literal} \rangle$

39



RDF and RDFS Vocabularies

RDF Vocabulary

Classes:

`rdf:Property`, `rdf:Statement`,
`rdf:XMLLiteral`
`rdf:Seq`, `rdf:Bag`, `rdf:Alt`, `rdf>List`

Properties:

`rdf:type`, `rdf:subject`, `rdf:predicate`,
`rdf:object`,
`rdf:first`, `rdf:rest`, `rdf:_n`
`rdf:value`

Resources:

`rdf:nil`

RDFS Vocabulary

RDFS Classes

`rdfs:Resource`
`rdfs:Class`
`rdfs:Literal`
`rdfs:Datatype`
`rdfs:Container`
`rdfs:ContainerMembershipProperty`

RDFS Vocabulary

RDFS Properties

`rdfs:domain`
`rdfs:range`
`rdfs:subPropertyOf`
`rdfs:subClassOf`
`rdfs:member`
`rdfs:seeAlso`
`rdfs:isDefinedBy`
`rdfs:comment`
`rdfs:label`

40



Readings

- Chapter 3 of the “Semantic Web Primer” and Chapter 7 of “Web Data Management” books
- The following material from the Semantic Web Activity Web page on RDF http://www.w3.org/standards/techs/rdf#w3c_all:
 - ◆ RDF Vocabulary Description Language 1.0: RDF Schema: <http://www.w3.org/TR/rdf-schema/>
 - ◆ RDF Semantics: <http://www.w3.org/TR/rdf-mt/>
- Check out the content published at the RDFS namespace URIs:
 - ◆ <http://www.w3.org/2000/01/rdf-schema#>
where you will find RDFS descriptions of the
RDF and RDFS vocabularies given in RDF/XML!

41



Acknowledgements

- Manolis Koubarakis “An Introduction to RDF Schema” lecture in course “Knowledge Technologies” <http://cgi.di.uoa.gr/~pms509/>
- Dieter Fensel and Federico Facca “Resource Description Framework (RDF)” lecture in course “Semantic Web” http://teaching-wiki.sti2.at/index.php/Semantic_Web

42