Knowledge Graphs: RDFS

ANNA QUERALT, OSCAR ROMERO

(FACULTAT D'INFORMÀTICA DE BARCELONA)

RDF Schema (RDFS)

Constraints on RDF statements must be stated in RDFS or OWL

RDFS allows to specify constraints on individual and relationships. More specifically:

- Declare objects and subjects as instances of certain classes
  - E.g., *Oscar isa lecturer*
- Inclusion statements between classes and between properties
  - E.g., lecturer *isa* human **TAXONOMIES!**
- Relate the domain and the range of a property to some classes
  - E.g., the predicate fatherOf relates a subject and an object that must be instances of the class human

will allow to say sth is instance , sth is class

sth is subset of another
stated in metadata level not in data level

constraint property of certain class
done at metadata level not data level

RDFS extends RDF not only to consider **instances** but **schema**

- The schema can be described with the same description
- rdfs: A namespace for RDFS
  - The URI is: http://www.w3.org/2000/01/rdf-schema#

RDFS Example

Some controversy here:
Some consider rdf:type to be part of RDF, some of RDFS

Schema

Taxonomies:

- (classes) :Lecturer rdfs:subclassOf :Human lecturer is subset of human
- (rels) :ResponsibleFor rdfs:subpropertyOf :Lectures

Domain and Range:

- :Lectures rdfs:domain :Human
- :Lectures rdfs:range :Course

here, it does not enforce constraint just infering knowledge

Instances

- :Oscar :Lectures :SDM

Based on the schema information and the instances created, RDFS allows us to **infer** new knowledge:
:Oscar rdf:type :Lecturer
:SDM rdf:type :Course

Reasoning (Inference)

Non-explicit knowledge inferred from explicitly asserted knowledge

A knowledge-base consists of schema and instances

- Schema:
  - :Lecturer rdfs:subclassOf :Human
  - :ResponsibleFor rdfs:subpropertyOf :Lectures
  - :Lectures rdfs:domain :Human
  - :Lectures rdfs:range :Course
- Instances (explicit facts asserted):
  - :Oscar rdf:type :Lecturer
  - :OpenData rdf:type :Course
  - :Oscar :ResponsibleFor :OpenData
- Inferred knowledge (instances that are a logical consequence of the schema + explicit instances):
  - :Oscar rdf:type :Human
  - :Oscar :Lectures :OpenData

Inference, in RDFS, is rule-based and is set by the pre-defined URIs of the RDFS namespace

RDFS

RDFS was created with two objectives:

- Express constraints on data (schema). For this purpose, RDFS defines a metamodeling level:
  - **RDFS Core classes**

- Enable inference on data (i.e., instances) based on the asserted (schema) constraints
  - In RDFS, inference is rule-based (*if the body holds then the header holds too*)
  - Rules are based on **RDFS core properties**
  - Two kinds of inference:
    - **Core type inference**: it infers the **type** (any of the RDFS core classes) of an asserted schema resource R (i.e., *R* rdf:type *coreClass*). This inference types the schema elements with RDFS core classes
    - **Domain-specific inference**:
      - **Inclusion dependencies** (taxonomies). This inference creates a inclusion relationship between two schema classes or two schema properties.
      - **Type inference**: it infers the type of an asserted instance *R* wrt a user-created schema class. This inference types an instance with a schema class.

RDFS Core Classes

rdfs:Resource, the class of all resources (**everything** is a resource)

rdfs:Class, the class of all classes

rdfs:Literal, the class of all literals

rdf:Property, the class of all properties

rdf:Statement, the class of all statements

These elements sit at the metamodeling level, and each element at the schema level must be typed (i.e., rdf:type) with one of these core classes

RDFS Core Properties (I)

rdf:type, which relates a resource to its class

`:oscar rdf:type :lecturer (constraint)`

- **Meaning**: this RDFS core property makes the constraint subject (i.e., :*oscar*) to be declared as an instance of the constraint object (i.e., *:lecturer*), which is automatically considered of type rdfs:Class
- **Inferred knowledge**:
  - Core type inference: the object is inferred as a Class.

  I.e.,: lecturer is a Class (*:lecturer rdf:type rdfs:Class*)（**inferred constraint**）

RDFS Core Properties (II)

rdfs:subClassOf, which relates a class to **one** of its superclasses

`:lecturer rdfs:subClassOf :human (schema constraint)`

`:oscar rdf:type :lecturer (instance)`

- **Meaning**: This RDFS core proprety makes the constraint subject and object to be declared as classes AND any instance of the subject is automatically declared as an instance of the object
- **Inferred knowledge**:
  - Core type inference: the subject and object are inferred as classes

  I.e.,: `:lecturer rdf:type rdfs:Class and`
  `:human rdf:type rdfs:Class (inferred constraints)`
  - Domain-specific inference (inclusion dependency): if *x* rdf:type *subject* then *x* rdf:type *object*

  I.e.,: `:oscar rdf:type :human (inferred instance)`

RDFS Core Properties (III)

rdfs:subPropertyOf, which relates a property to **one** of its superproperties

`:responsibleFor rdfs:subpropertyOf :lectures (schema constraint)`

`:oscar :responsibleFor :OD (instance)`

- **Meaning**: this RDFS core property makes the constraint subject and object to be declared as properties AND any two resources related by the subject predicate in a triple instance are automatically declared to be related by the object predicate
- **Inferred knowledge**:
  - Core type inference: the subject and object are inferred as properties

  I.e.,: `:responsibleFor rdf:type rdf:Property and`
  `:lectures rdf:type rdf:Property (inferred constraints)`
  - Additional inference (inclusion dependency): *x subject y* then *x object y*

  I.e.,: `:oscar :lectures :OD (inferred instance)`

RDFS Core Properties (IV)

rdfs:domain, which specifies the domain of a property

`:lectures rdfs:domain :lecturer (schema constraint)`

`:oscar :lectures :OD (instance)`

- **Meaning**: Via this RDFS core property we can express that whenever the constraint subject is used in an instance triple (as predicate), the subject of that triple is automatically considered an instance of the constraint object
- **Inferred knowledge**:
  - Core type inference: the subject is declared to be a property and the object is declared to be a class

I.e.,: `:lectures rdf:type rdf:Property and`
`:lecturer rdf:type rdfs:Class (inferred constraints)`

  - Domain-specific inference (type inference): if *x subject y* then *x rdf:type object*

I.e., `:oscar rdf:type :lecturer (inferred instance)`

RDFS Core Properties (V)

rdfs:range, which specifies the range of a property

`:lectures rdfs:range :course(schema constraint)`

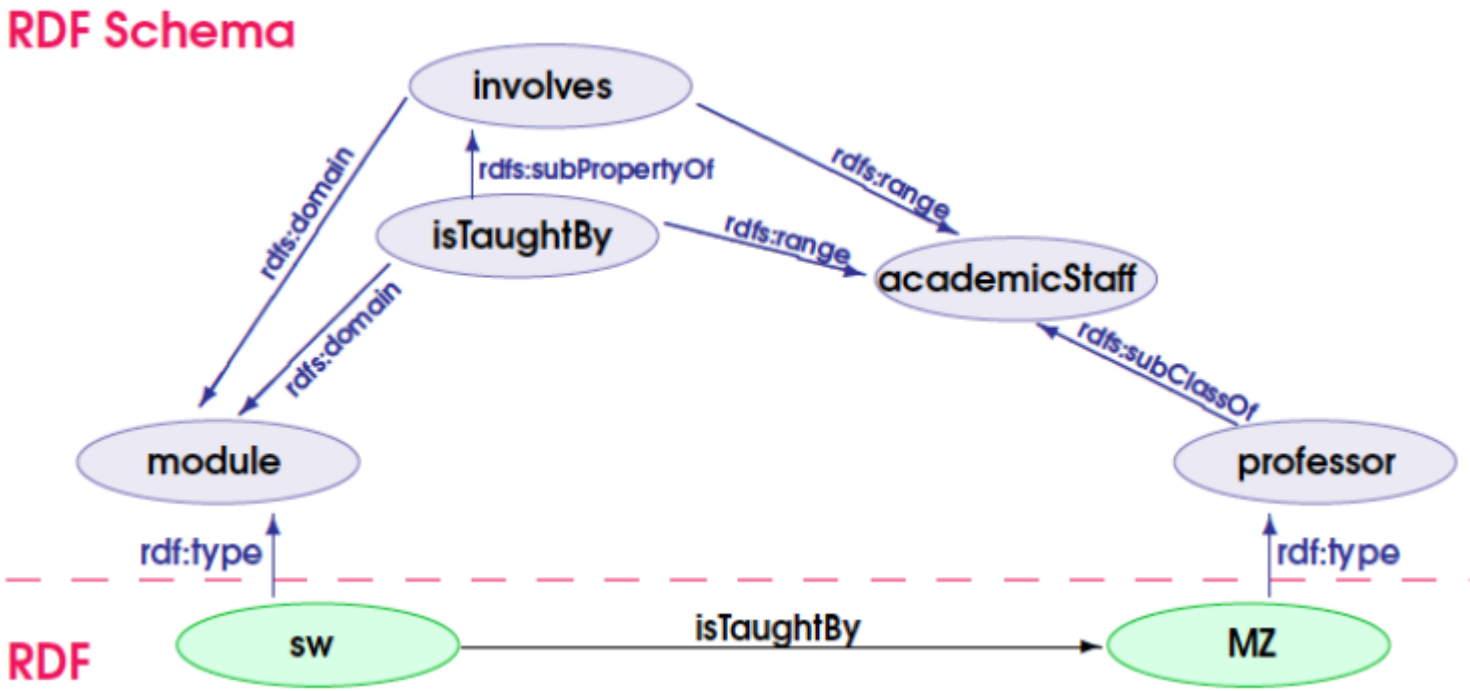
`:oscar :lectures :OD(instance)`

- **Meaning**: Via this RDFS core property we can express that whenever the constraint subject is used in an instance triple (as predicate), the object of that triple is automatically considered an instance of the constraint object
- **Inferred knowledge**:
  - Core type inference: the subject is declared to be a property and the object is declared to be a class

  I.e.,: `:lectures rdf:type rdf:Property and`
  `:course rdf:type rdfs:Class (inferred constraints)`
  - Domain-specific inference (type inference): if *x subject y* then *y rdf:type object*

  I.e., `:OD rdf:type :course (inferred instance)`

RDFS Semantics: Exercise



Generate all the inferred triples based on the semantics previously discussed

The RDFS Paradox

Russel's Paradox was ennounced as:

Consider the set $A = \{B \mid B \notin B\}$

set of element that do not belong to set

Popular version, the barber paradox

Suppose there is a town with just one male barber. According to law in this town, the barber shaves all and only those men in town who do not shave themselves. Who shaves the barber?

- if the barber does shave himself, then the barber (himself) must not shave himself
- if the barber does not shave himself, then the barber (himself) must shave himself

It is a set-theoretical paradox that arises within naïve set theory by considering the set of all sets that are not members of themselves.

- In the original definition of inference for RDFS the Russel's Paradox did hold (!)

http://plato.stanford.edu/entries/russell-paradox/

Beware of the RDFS Paradox!

Originally, the RDF metamodel was flawed

Problems in the original definition of RDFS

- *rdfs:Class* is an instance of itself
  - In RDFS all clases are instances of rdfs:Class, including rdfs:Class (Russel's paradox)
  - Thus, it presents a possibly infinite layer of classes
- *rdfs:Resource* is a superclass and an instance of *rdfs:Class*
- *rdfs:subClassOf, rdf:type, rdfs:Range* and *rdfs:Domain* are both used to define the other RDFS primitives and the ontology
  - Metamodel and model mixed
  - Is *rdfs:subClassOf* a relationship between sets or between sets of sets?!

both instance to schema level and schema level to metamodel level ma same concept use bhairako cha
so model and metamodel are mixed
confusion huncha

RDFS Flawed Reasoning

Due to the Russel's paradox, reasoning in RDFS (as originally defined) is flawed

As consequence, the SPARQL community rethought the RDFS metamodel to introduce **fix-point reasoning**

- Modified RDFS Entailment Regime

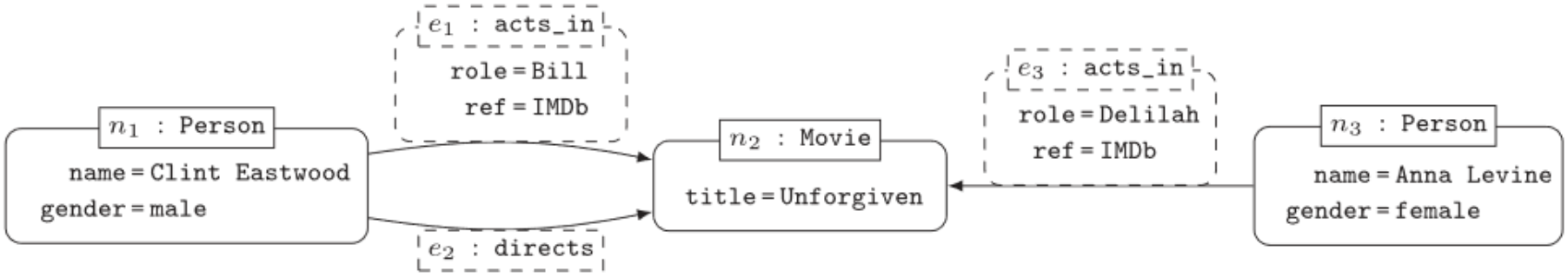
RDFS entailment patterns.

	If S contains:	then S RDFS entails recognizing D:
rdfs1	xxx aaa yyy.	aaa rdf:type rdf:Property .
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	aaa rdfs:subPropertyOf bbb . xxx aaa yyy .	xxx bbb yyy .
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 .

+ core class inference rules (check the RDF core properties slides)

RDFS Exercise

Create a correct RDFS graph capturing as much constraints as possible from the following graph:



What triples may you infer from the asserted RDFS graph?

Summary

RDFS

- RDFS Core Components
- RDFS Semantics
- RDFS Metamodel

Bibliography

RDFS. W3C Recommendation. Latest version at http://www.w3.org/TR/rdf-schema/

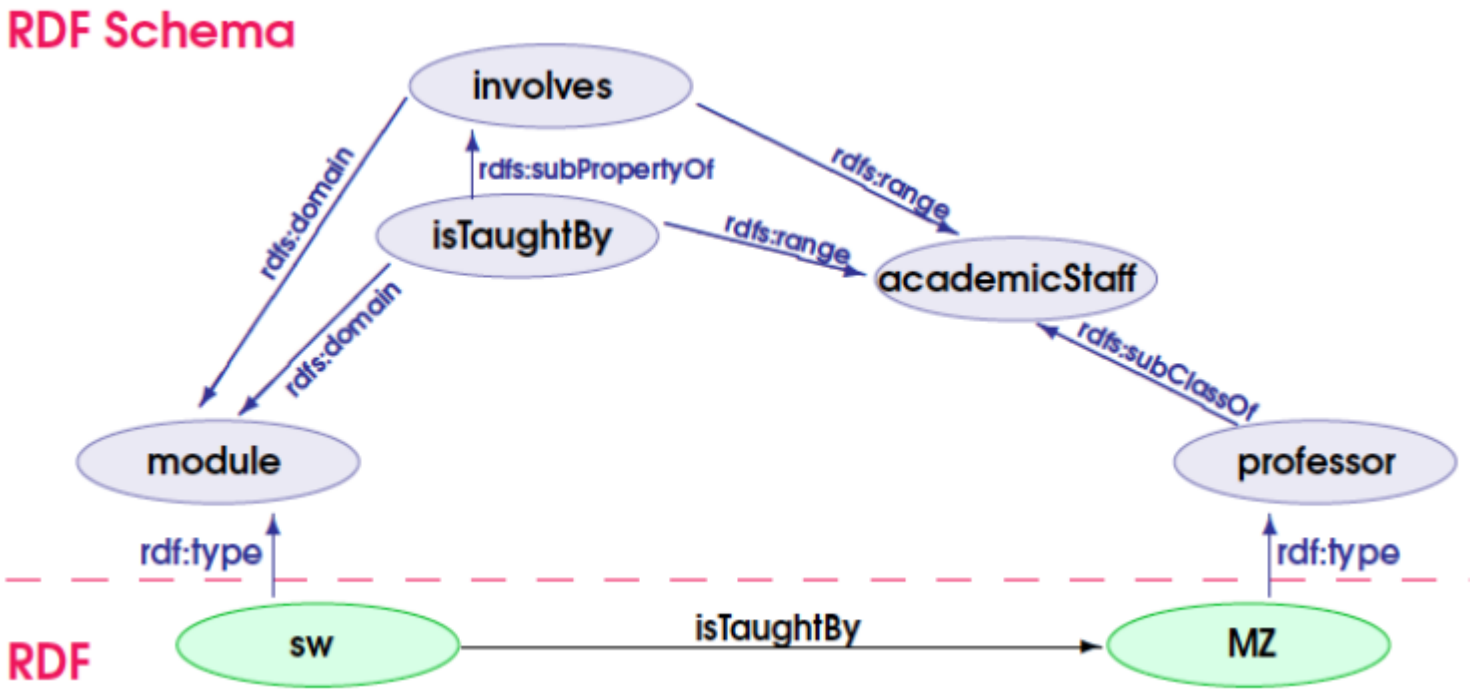
APPENDIX

Here, we showcase why the original RDFS reasoning is flawed. In this appendix, we will blindly follow the inference rules **originally** presented for RDFS, for a given toy example, and showcase the Russel's Paradox.

Considering the example in the next slide, the subsequent slides apply certain inference rules (the rule is stated in the title and the slides unfold the original RDFS inference).

Important note: if you follow the rules presented in this session you are safe! The Russell's paradox requires other initially defined rules that we omitted when discussing the RDFS inference to guarantee safe reasoning. This appendix is only to show, for those curious, why the Russel's Paradox held in the original definition

RDFS Semantics

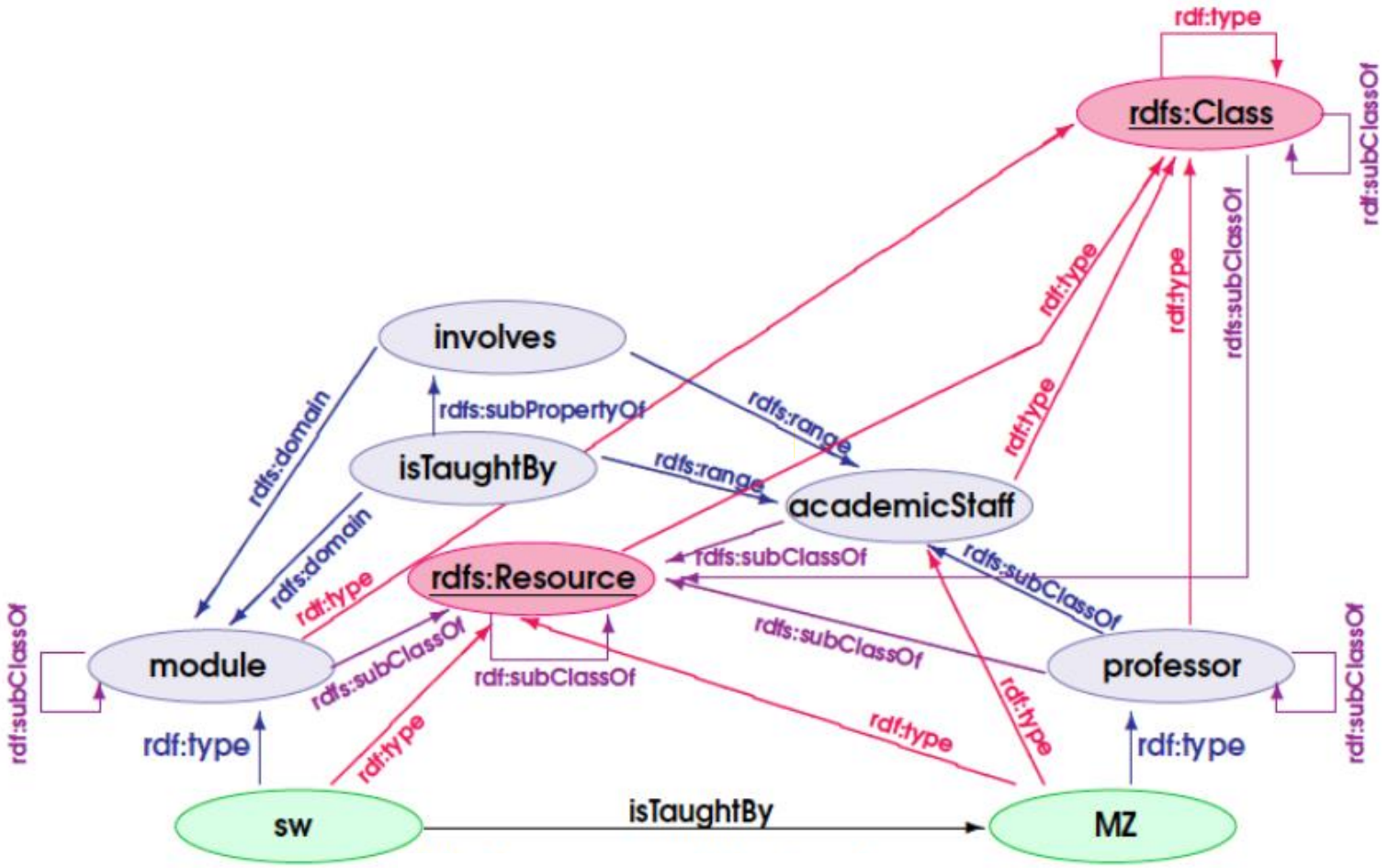


RDF Semantics: Class and Subclass

Consider the following inferred knowledge:

- The object of *rdf:type* is a class
- Every class is a subclass of *rdfs:Resource*
- If C_1 is *rdfs:subClassOf* of C_2 then, both C_1 and C_2 are classes. Also:
  - *rdfs:subClassOf* is reflexive on classes
  - *rdfs:subClassOf* is transitive

RDF Semantics: Class and Subclass

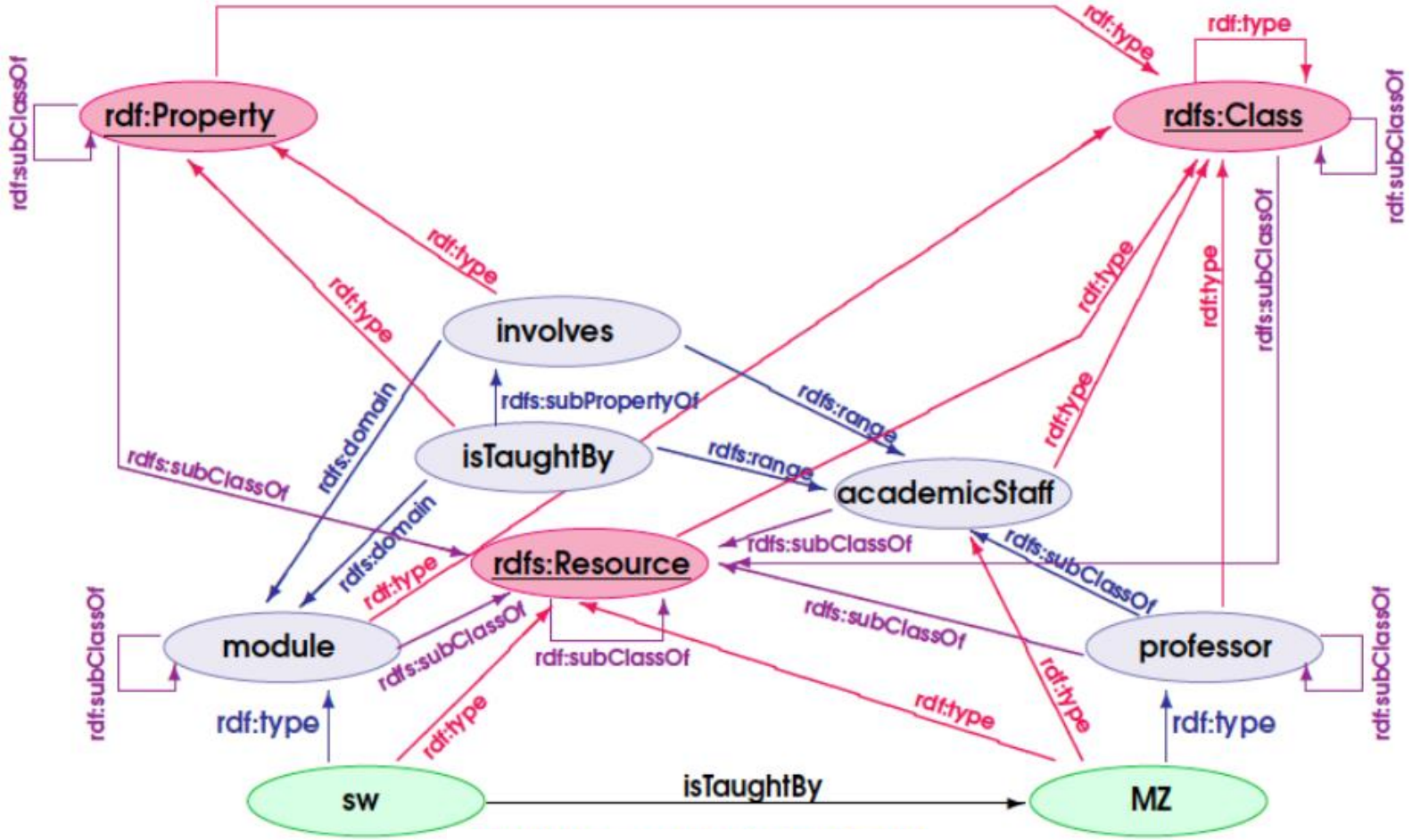


RDFS Semantics: Property and SubProperty

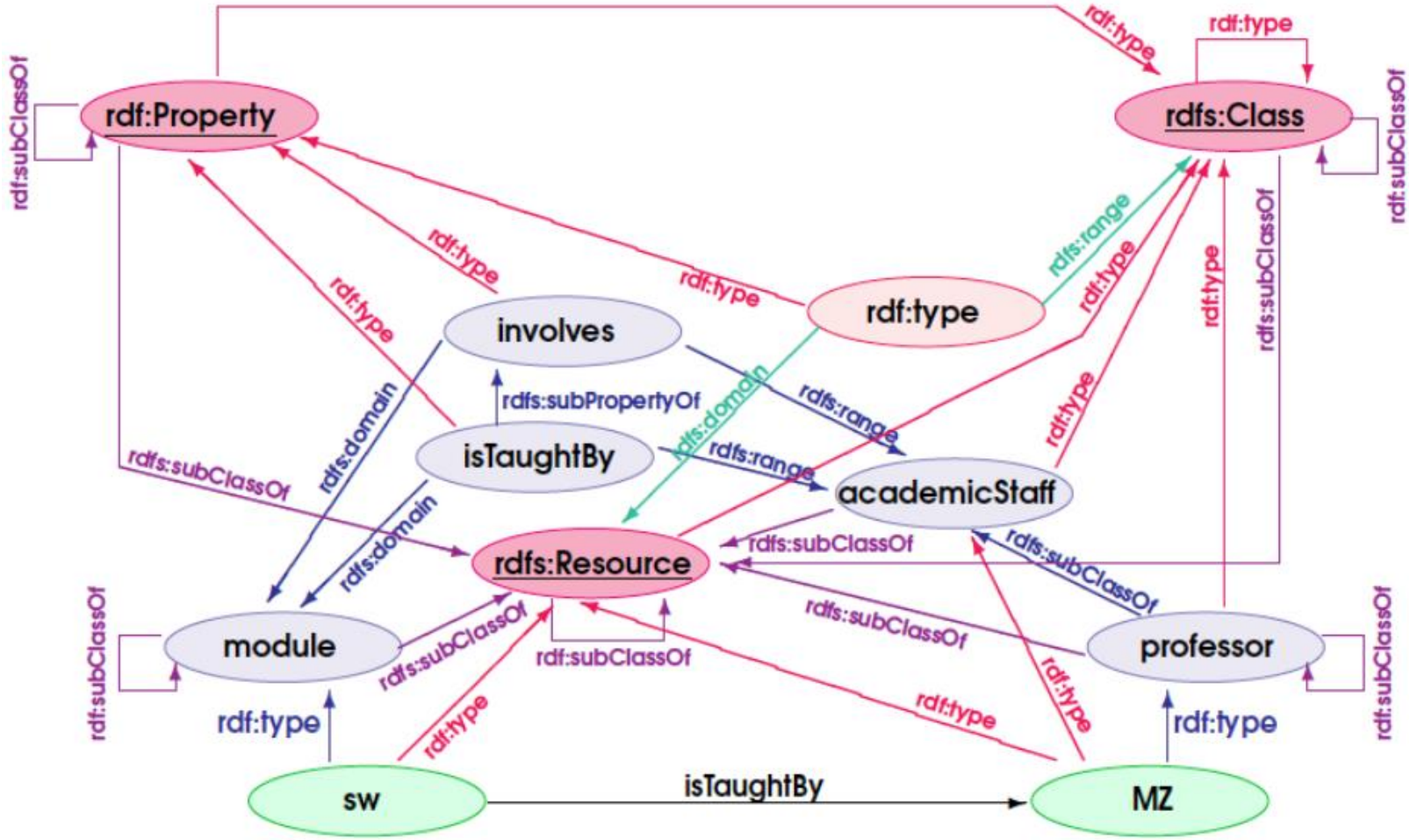
Consider the following inferred knowledge:

- If P_1 is *rdfs:subPropertyOf* of P_2 then, both P_1 and P_2 are properties. Also:
  - *rdfs:subPropertyOf* is reflexive on properties
  - *rdfs:subPropertyOf* is transitive

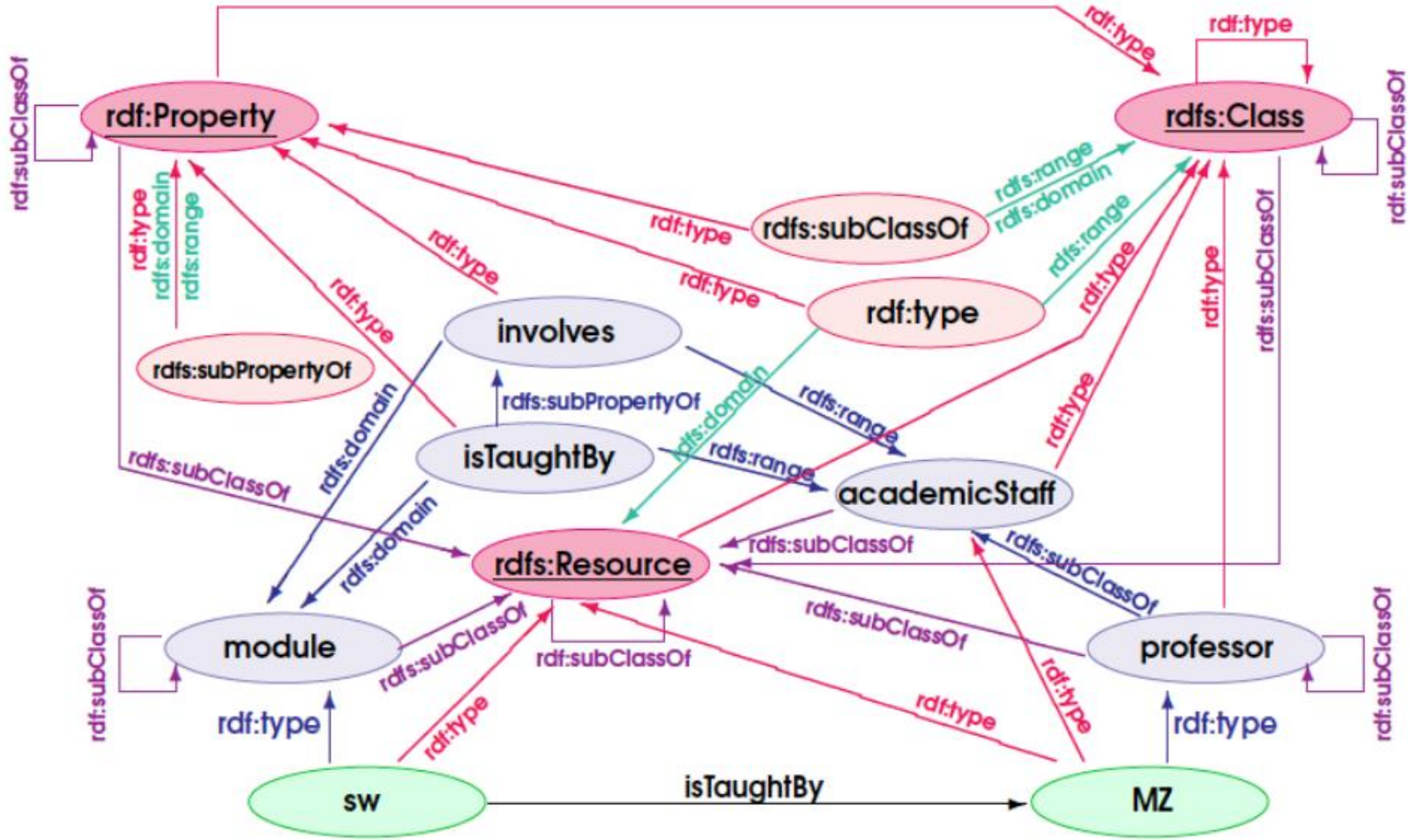
RDFS Semantics: Property and Subproperty



RDFS Semantics: Property and Subproperty



RDFS Semantics: Property and Subproperty

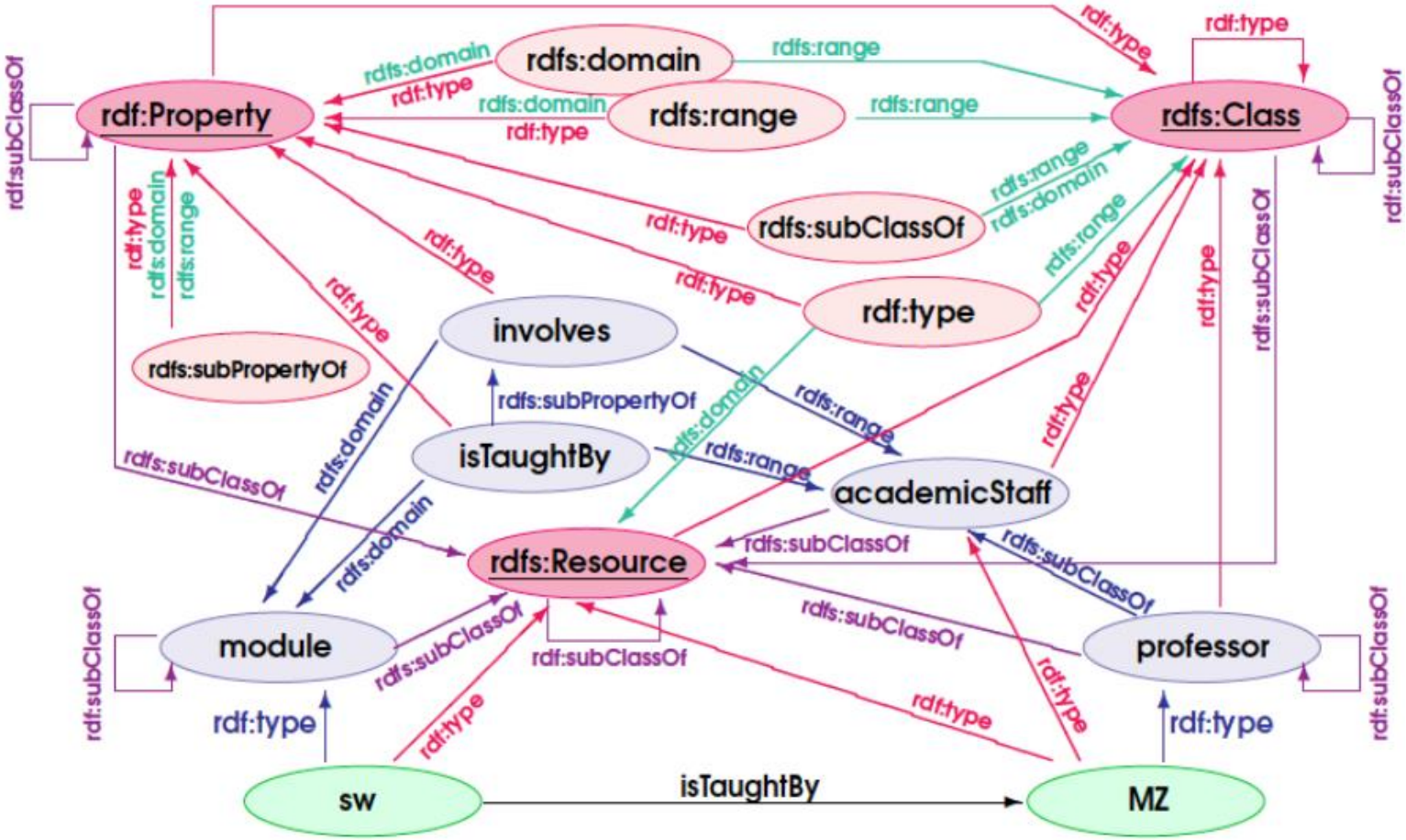


RDFS Semantics: Domain and Range

These are **axiomatic** triples

- Core knowledge to be assumed to be true for any for any profile (reasoning)
- If P *rdfs:domain* C then C is a class
- If P *rdfs:range* C then C is a class

RDFS Semantics: Domain and Range



RDFS Semantics: Conclusions

If you carefully check the graph from the previous slide, you will realize it does not make any sense and, indeed, it falls into the Russel's Paradox.