

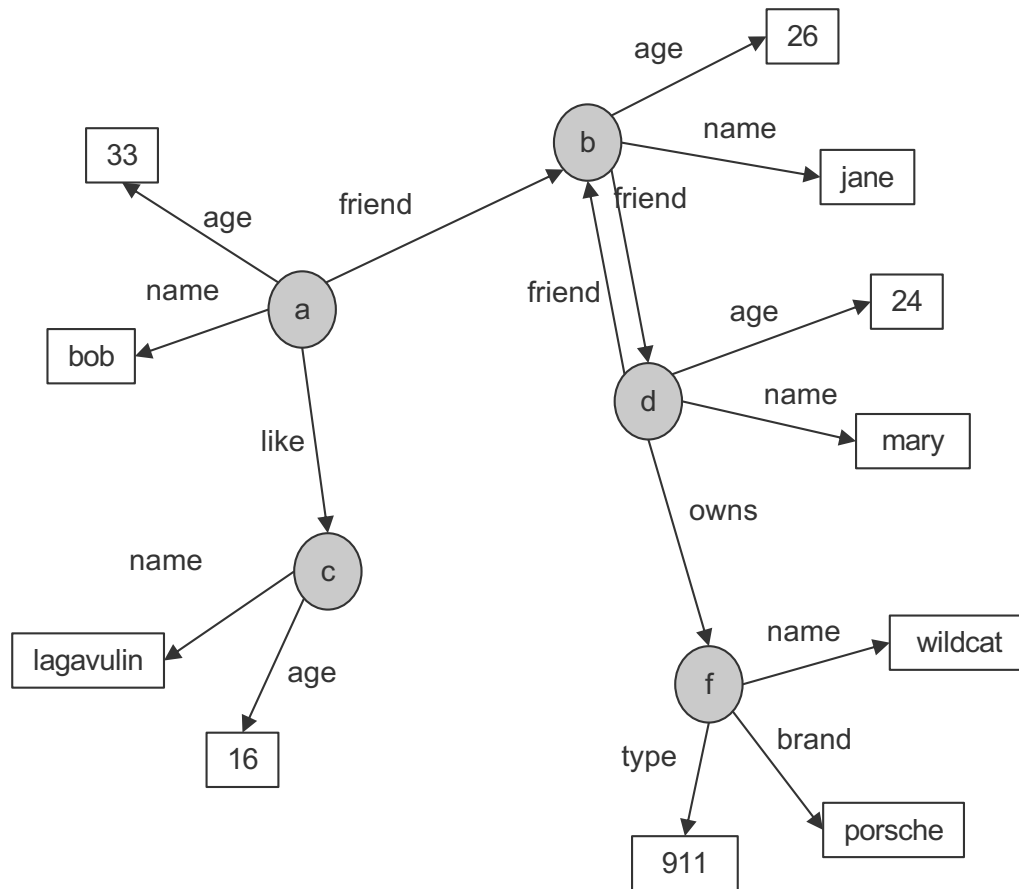


# RDF and Knowledge Nets

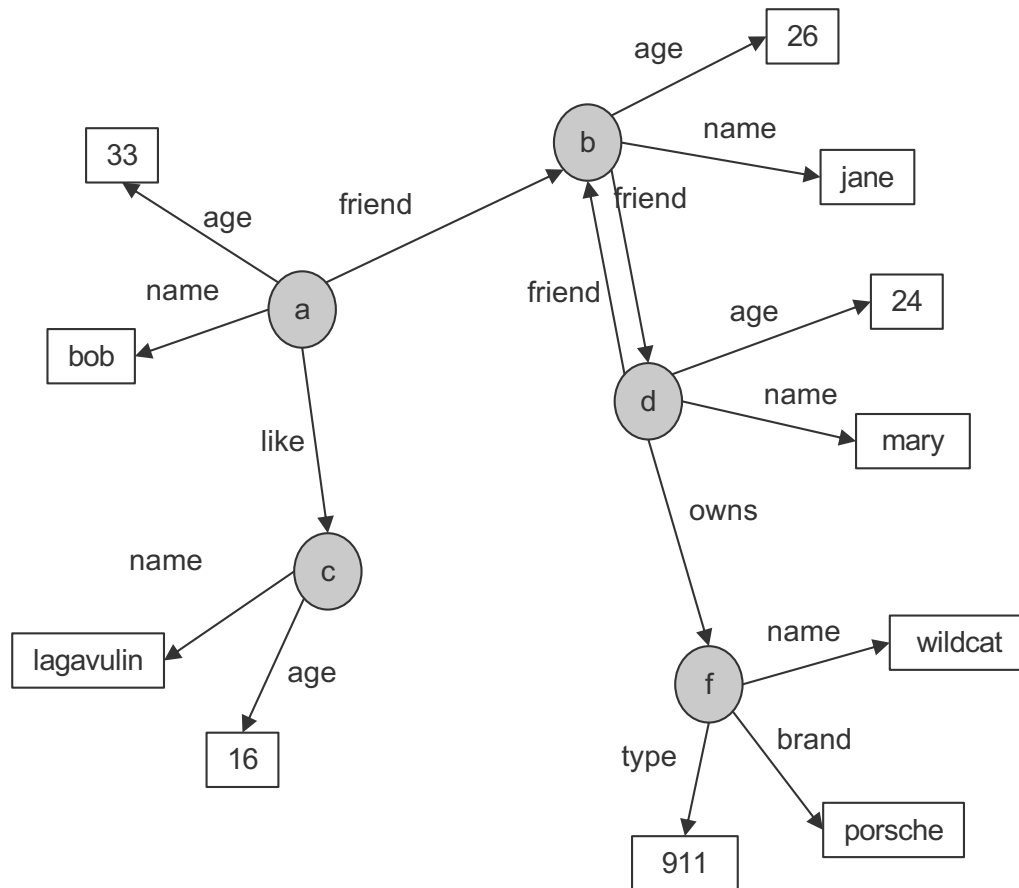


# AN EXAMPLE

# Example: KnowledgeNets ...



# Example: KnowledgeNets and Facts



```

name(a,bob).
age(a,33).
friend(a,b).
like(a,c).

name(b,jane).
age(b,26).
friend(b,d).

name(c,lagavulin).
age(c,16).

name(d,tom).
age(d,24).
friend(d,b).
owns(d,f).

name(f,wildcat).
brand(f,porsche).
type(f,911).
  
```

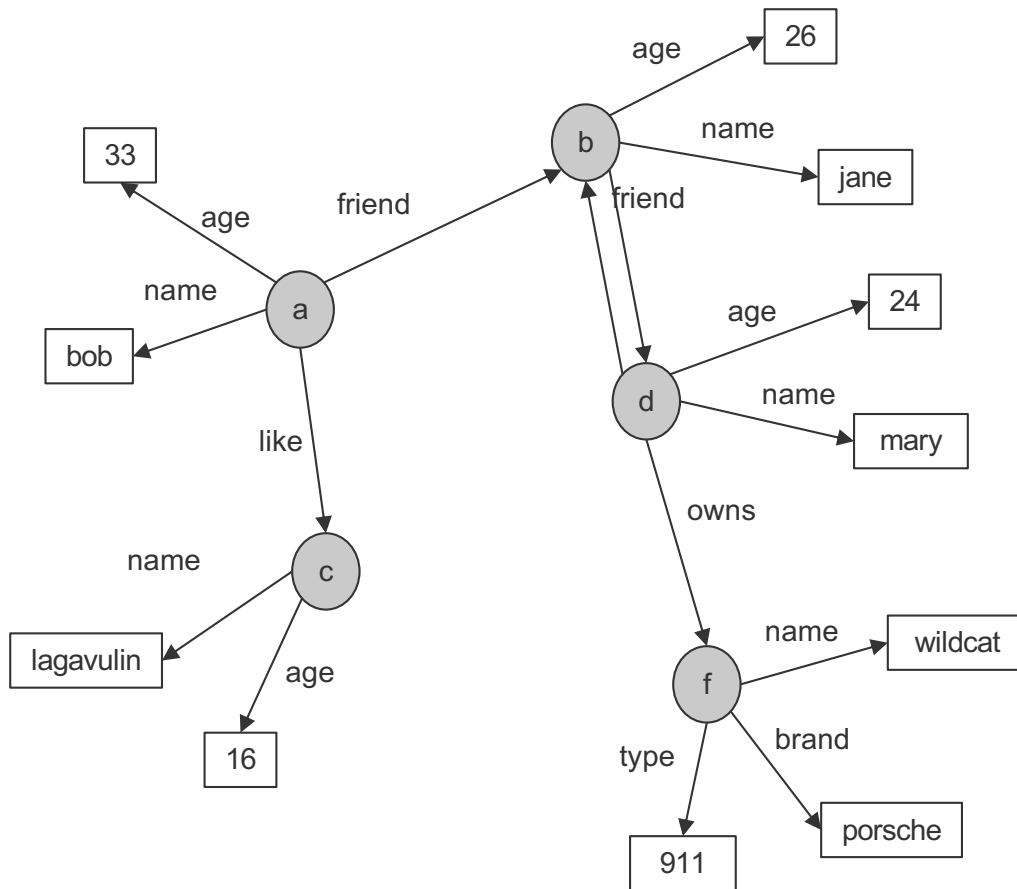
# Triples

- Everything is stored in Triples

`triple(<start-node>, <relationship>, <end-node>)`

- Many triples are stored in a Triplestore.

# Example: KnowledgeNets and Triples



```

triple(a,name,bob).
triple(a,age,33).
triple(a,friend,b).
triple(a,like,c).

triple(b,name,jane).
triple(b,age,26).
triple(b,friend,d).

triple(c,name,lagavulin).
triple(c,age,16).

triple(d,name,tom).
triple(d,age,24).
triple(d,friend,b).
triple(d,owns,f).

triple(f,name,wildcat).
triple(f,brand,porsche).
triple(f,type,911).
  
```

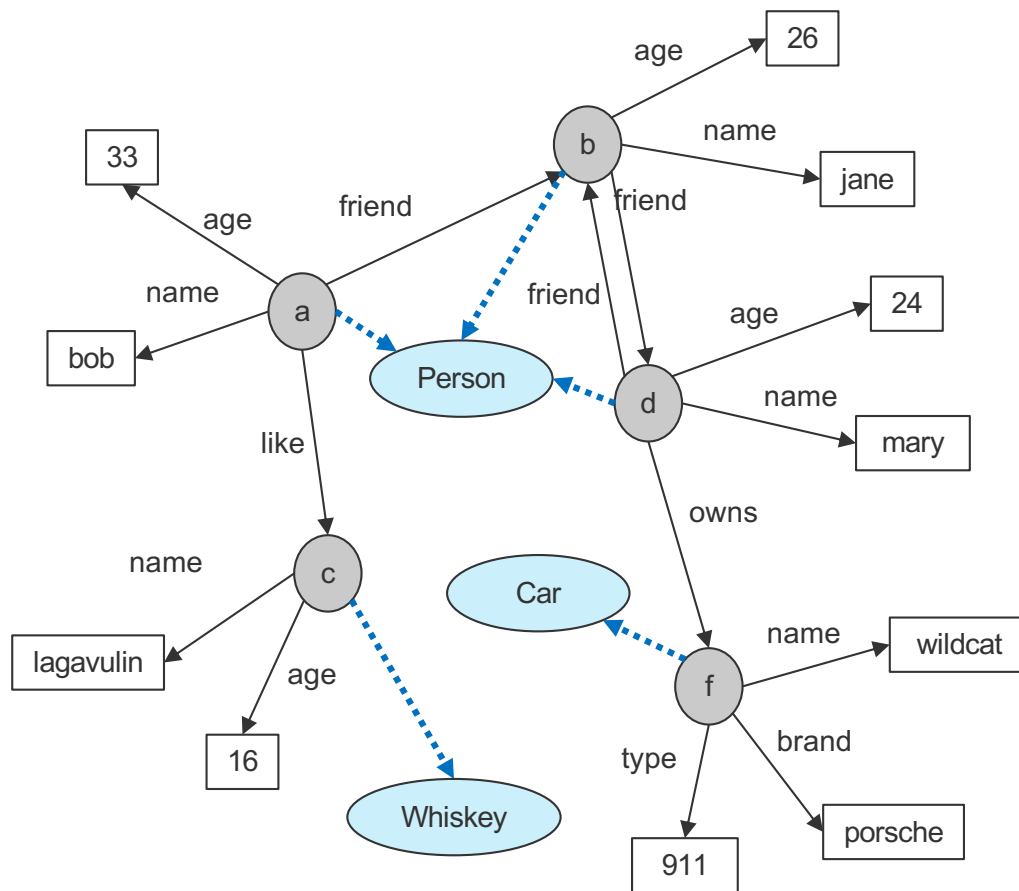
# Types

- Some nodes are of a specific type
- Relationship to a type can be stored with “**rdf:type**”

```
triple(<start-node>, rdf:type, <Type>)
```

- Types are schema elements

# Example: KnowledgeNets, Triples and Types



```
triple(a, rdf:type, Person)
triple(a, name, bob).
triple(a, age, 33).
triple(a, friend, b).
triple(a, like, c).
```

```
triple(b, rdf:type, Person)
triple(b, name, jane).
triple(b, age, 26).
triple(b, friend, d).
```

```
triple(c, rdf:type, Whiskey)
triple(c, name, lagavulin).
triple(c, age, 16).
```

```
triple(d, rdf:type, Person)
triple(d, name, tom).
triple(d, age, 24).
triple(d, friend, b).
triple(d, owns, f).
```

```
triple(f, rdf:type, Car)
triple(f, name, wildcat).
triple(f, brand, porsche).
triple(f, type, 911).
```





# RDF

## Basic Ideas of RDF

- Basic building block: **object-attribute-value** triple
  - ◆ Simple form of object-orientation
  - ◆ Triple is (also) called a **statement**
  - ◆ Sometimes also called **Subject-Predicate-Object** in analogy to the simplified structure of English sentences.
- RDF data source = set of object-attribute-value triples
- RDF has been given a syntax in XML
  - ◆ This syntax inherits the benefits of XML
  - ◆ Other syntactic representations of RDF possible

# The fundamental concepts of RDF

- Resources
- Properties
- Statements

## Resources

- We can think of a resource as an object, an instance, a “thing” we want to talk about
  - ◆ In the example: a, b, c, d, .....
- Every resource has a **URI**, a Universal Resource Identifier
- A URI can be
  - ◆ a URL (Web address) or
  - ◆ some other kind of unique identifier
- Example: Every node is a resource

# Properties

- They describe relations between resources
  - ◆ E.g. “name”, “age”, “brand”, etc.
- Properties are a special kind of resources
- Properties are also identified by URIs !!
- Advantages of using URIs:
  - ◆ A global, worldwide, unique naming scheme (idea)
  - ◆ Reduces the homonym problem of distributed data representation

# Statements

- Statements assert the properties to resources
- A statement is an object-attribute-value triple
  - ◆ It consists of a resource, a property, and a value
- Values can be resources or **literals**
  - ◆ Literals are atomic values (strings, numbers, etc)

## Three Views of a Statement

- A triple
- A piece of a graph
- A piece of XML code

Thus an RDF document can be viewed as:

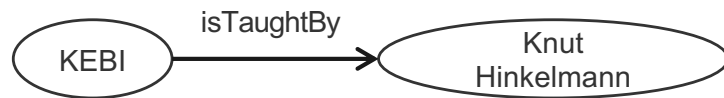
- A set of triples
- A graph (Knowledge Net)
- An XML document



# RDF SCHEMA



# RDF and RDF Schema



## ■ Want to express:

- ◆ There are classes like Courses, Lecturers, Professors, Staff Members etc
- ◆ Relationships between these classes
- ◆ Relationship “isTaughtBy” starts from a Course and ends in a Academic Staff Member

## ■ Want to do:

- ◆ Assign (automatically) instances to (all of) their classes
- ◆ Complete the relationships (also on the schema level)

## Basic Ideas of RDF Schema

- RDF is a universal language that lets users describe resources in their own vocabularies
  - ◆ RDF does not assume, nor does it define semantics of any particular application domain
- The user can do so in **RDF Schema** using:
  - ◆ Classes and Properties
  - ◆ Class Hierarchies and Inheritance
  - ◆ Property Hierarchies
- The reasoning comes with the schema!

## Classes and their Instances

- We must distinguish between
  - ◆ Concrete “things” (individual objects) in the domain: “KEBI”, “Knut Hinkelmann”, etc.
  - ◆ Sets of individuals sharing properties called **classes**: Course, Lecturer, Staff Member etc.
  
- Individual objects that belong to a class are referred to as **instances** of that class
  
- The relationship between instances and classes in RDF is defined through **rdf:type**

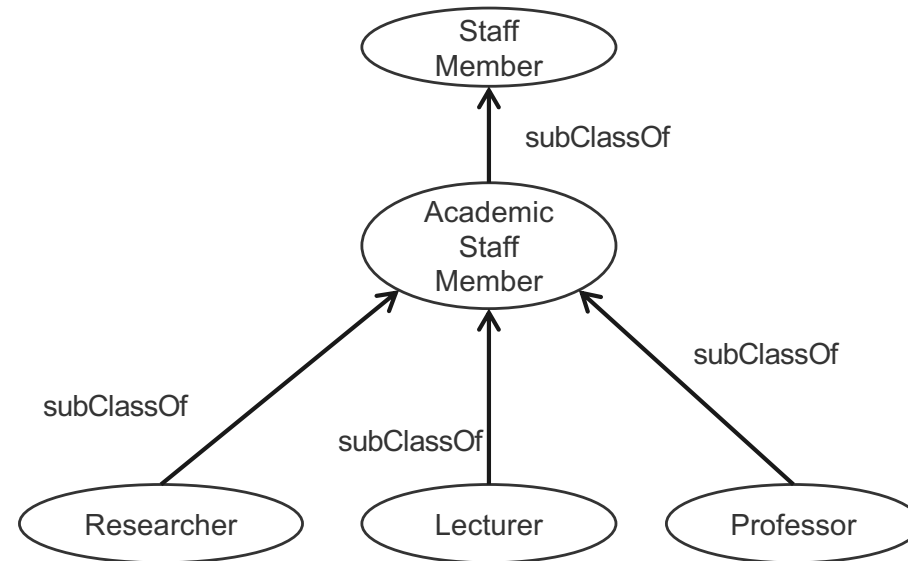
# Nonsensical Statements disallowed through the Use of Classes

- **KEBI is taught by Concrete Maths**
  - ◆ We want courses to be taught by lecturers only
  - ◆ Restriction on values of the property “is taught by” (**range restriction**)
  
- **Room AB1 is taught by Knut Hinkelmann**
  - ◆ Only courses can be taught
  - ◆ This imposes a restriction on the objects to which the property can be applied (**domain restriction**)
  
- Range and Domain restrictions are defined with the relations **rdfs:range** and **rdfs:domain**

## Class Hierarchies

- Classes can be organised in hierarchies
  - ◆ A is a **subclass** of B if every instance of A is also an instance of B
  - ◆ Then B is a **superclass** of A
- A subclass graph need not be a tree
- A class may have multiple superclasses
  
- The relationship between sub- and superclass is defined through **rdfs:subClassOf**

# Class Hierarchy Example



## Inheritance in Class Hierarchies

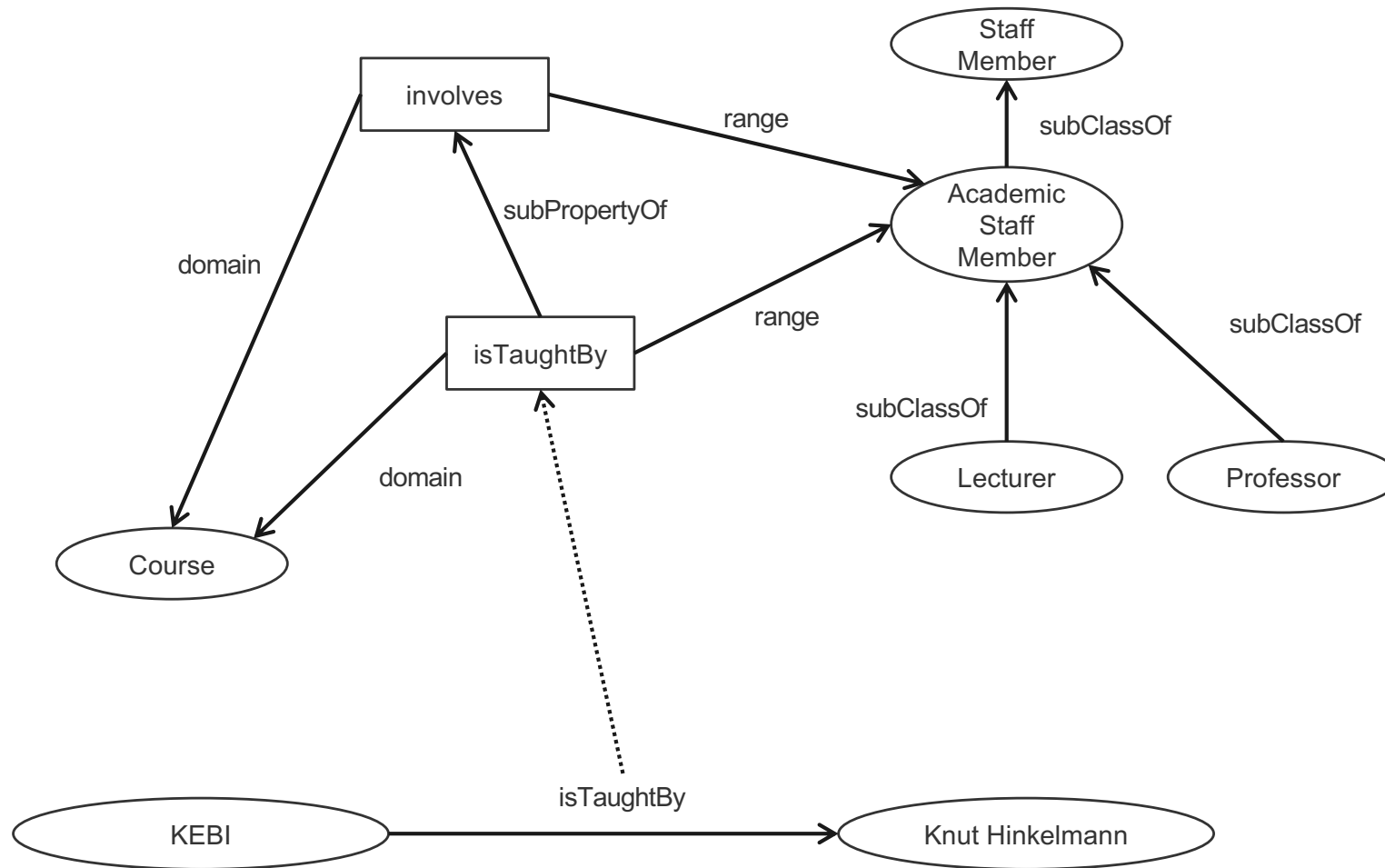
- Range restriction: **Courses must be taught by academic staff members only**
- Barbara Re is a professor
- She **inherits** the ability to teach from the class of academic staff members
- This is done in RDF Schema by fixing the semantics of “is a subclass of”
  - ◆ It is not up to an application (RDF processing software) to interpret “is a subclass of”

# Property Hierarchies

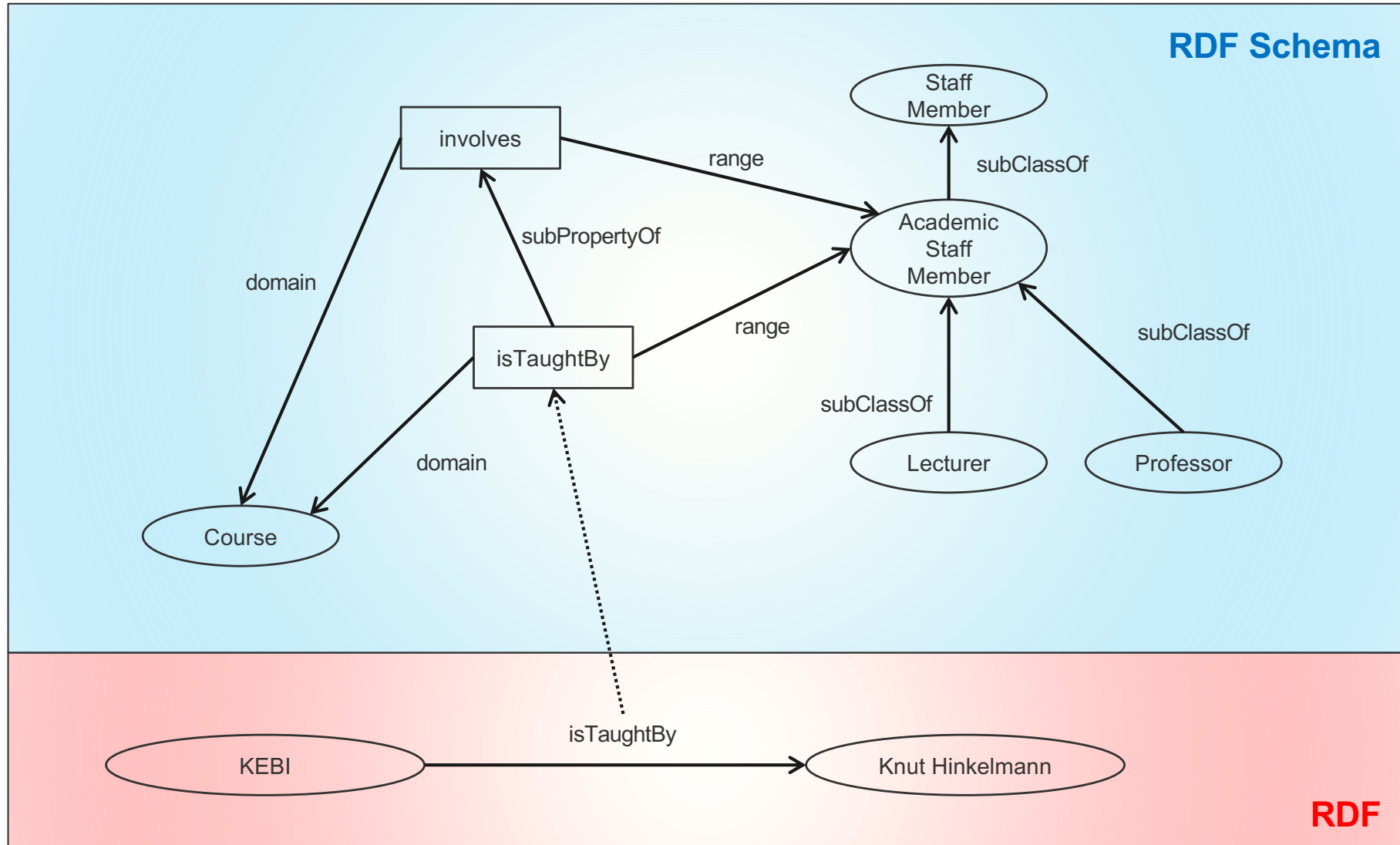
- Hierarchical relationships for properties
  - ◆ E.g., “is taught by” is a subproperty of “involves”
  - ◆ If a course C is taught by an academic staff member A, then C also involves A
- The converse is not necessarily true
  - ◆ E.g., A may be the teacher of the course C, or
  - ◆ a tutor who marks student homework but does not teach C
- P is a **subproperty** of Q, if  $Q(x,y)$  is true whenever  $P(x,y)$  is true
  
- The relationship between sub- and superproperties is defined through **`rdfs:subPropertyOf`**



# RDF Layer vs RDF Schema Layer



# RDF Layer vs RDF Schema Layer





# REASONING IN RDF SCHEMA

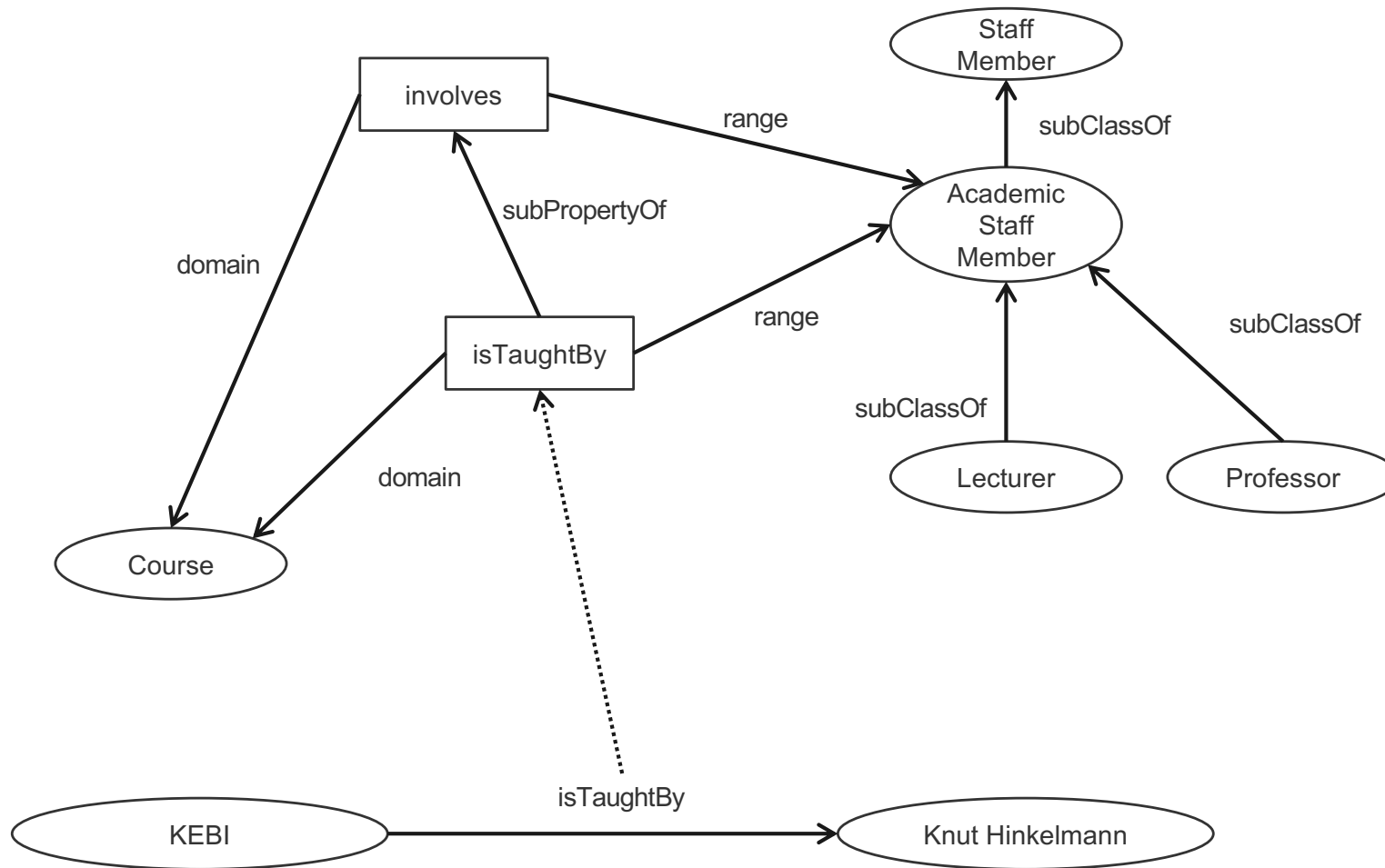
## Semantics based on Inference Rules

- Semantics in terms of RDF triples
- ... and sound and complete inference systems
- This inference system consists of **inference rules** of the form:

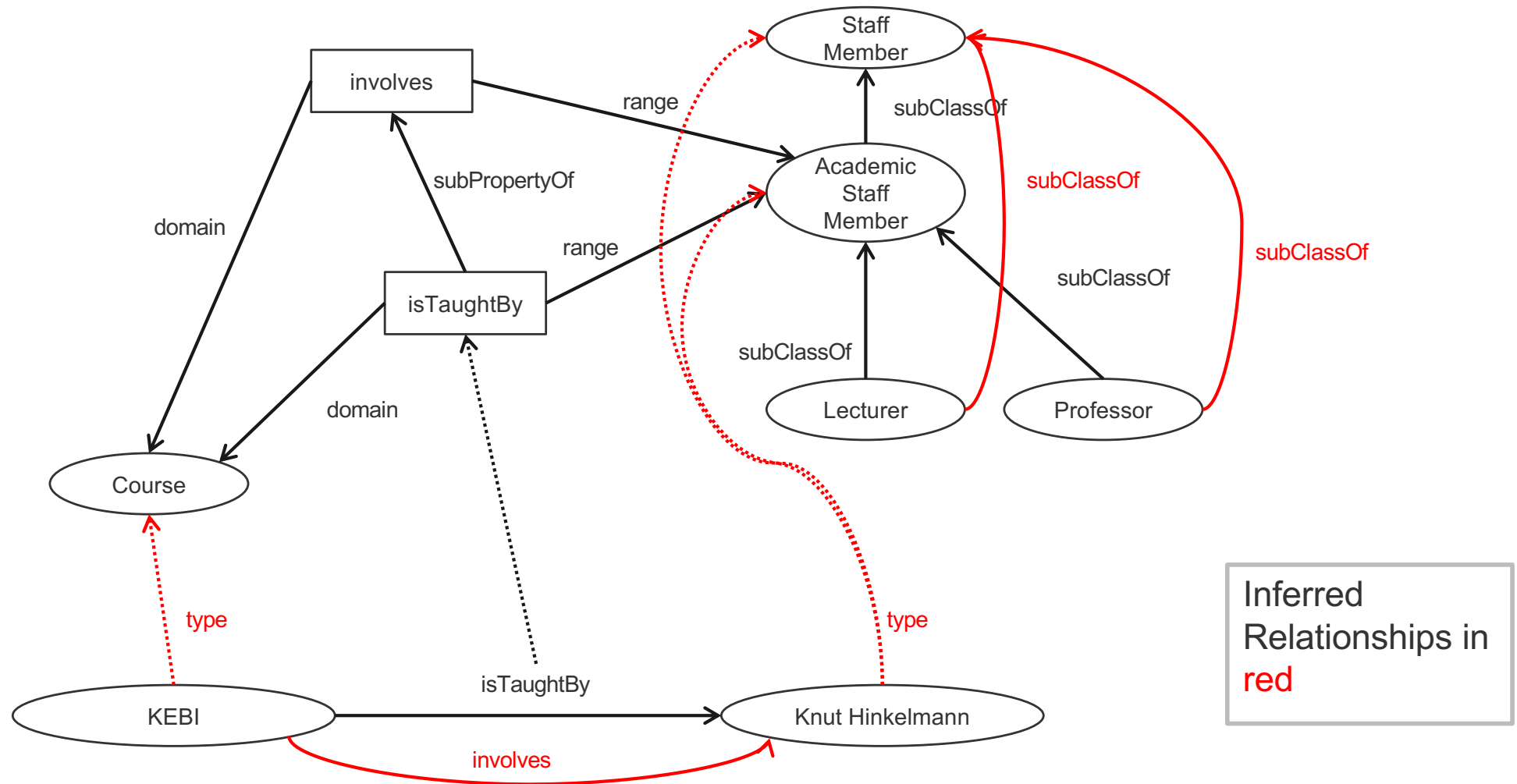
**IF E contains certain triples**  
**THEN add to E certain additional triples**

- where **E** is an arbitrary set of RDF triples
- Sometimes inference rules also called entailment rules.

# Example for Inferences made by Inference Rules (1/2)



# Example for Inferences made by Inference Rules (2/2)



## Example of Inference Rules

IF (?x, ?p, ?y),

(?p, **rdfs:range**, ?u)

THEN (?y, **rdf:type**, ?u)

- Any resource **?y** which appears as the value of a property **?p** can be inferred to be a member (type) of **?u**
  - ◆ This shows that range definitions in RDF Schema are not used to restrict the range of a property, but rather to infer the membership of the range

## Example of Inference Rules

**IF** (?x, ?p, ?y),

(?p, **rdfs:domain**, ?u)

**THEN** (?x, **rdf:type**, ?u)v

- Any resource **?x** which appears as the domain of a property **?p** can be inferred to be a member of the domain of **?p**, i.e. **?u**
  - ◆ This shows that range definitions in RDF Schema are not used to restrict the range of a property, but rather to infer the membership of the range



## Further Examples of Inference Rules

**IF** (?x, rdf:type, ?u),  
    (?u, rdfs:subClassOf, ?v)  
**THEN** (?x, rdf:type, ?v)

**IF** (?u, rdfs:subClassOf, ?v),  
    (?v, rdfs:subClassOf, ?w)  
**THEN** (?u, rdfs:subClassOf, ?w)

**IF** (?x, ?p, ?y)  
**THEN** (?p, rdf:type, rdf:property)

## RDF(S) Semantics: Examples

**IF** (netherlands, **rdf:type**, EuropeanCountry),  
(EuropeanCountry, **rdfs:subClassOf**, Country)  
**THEN** (netherlands, **rdf:type**, Country)

**IF** (aspirin, alleviates, headache),  
(alleviates, **rdfs:range**, Symptom)  
**THEN** (headache, **rdf:type**, Symptom)

## RDF(S) Semantics: Examples

**IF** (Νετηερλανδσ, **rdf:type**, ΕυροπεανΧουντρψ),  
(ΕυροπεανΧουντρψ, **rdfs:subClassOf**, Χουντρψ)  
**THEN** (Νετηερλανδσ, **rdf:type**, Χουντρψ)

**IF** (ασπιριν, αλλεπιατεσ, ηεαδαχηε),  
(αλλεπιατεσ, **rdfs:range**, σψμπτομ)  
**THEN** (ηεαδαχηε, **rdf:type**, σψμπτομ)

# All 13 RDFS entailment rules

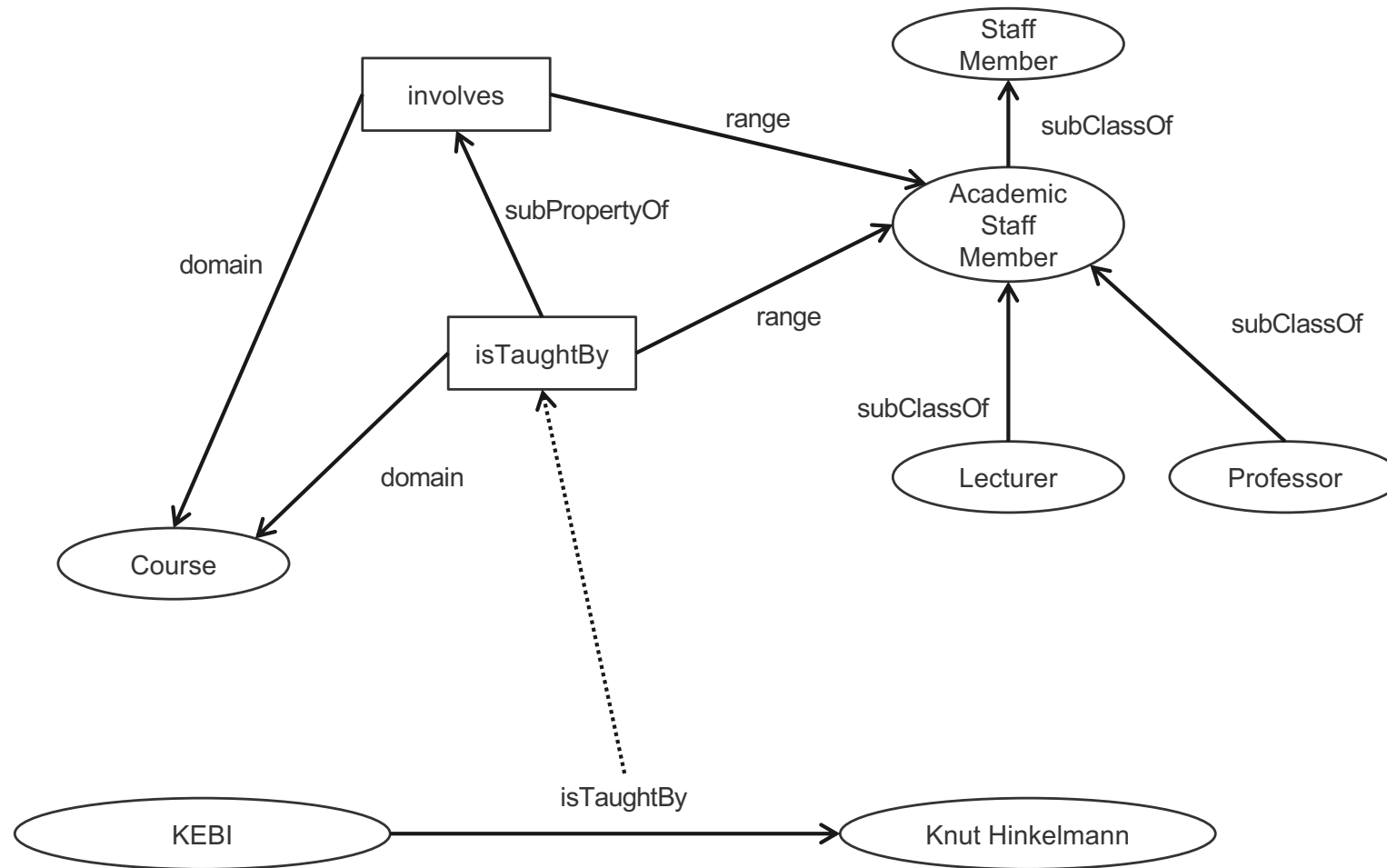
Rule Name	If E contains:	then add:
rdfs1	?u ?p ?n. where ?n is a plain literal (with or without a language tag).	_:nnn rdf:type rdfs:Literal . where _:nnn identifies a blank node <u>allocated to</u> ?n by rule <u>rule lg</u> .
rdfs2	?p rdfs:domain ?x . ?u ?p ?y .	?u rdf:type ?x .
rdfs3	?p rdfs:range ?x . ?u ?p ?v .	?v rdf:type ?x .
rdfs4a	?u ?p ?x .	?u rdf:type rdfs:Resource .
rdfs4b	?u ?p ?v.	?v rdf:type rdfs:Resource .
rdfs5	?u rdfs:subPropertyOf ?v . ?v rdfs:subPropertyOf ?x .	?u rdfs:subPropertyOf ?x .
rdfs6	?u rdf:type rdf:Property .	?u rdfs:subPropertyOf ?u .
rdfs7	?p rdfs:subPropertyOf ?q . ?u ?p ?y .	?u ?q ?y .
rdfs8	?u rdf:type rdfs:Class .	?u rdfs:subClassOf rdfs:Resource .
rdfs9	?u rdfs:subClassOf ?x . ?v rdf:type ?u .	?v rdf:type ?x .
rdfs10	?u rdf:type rdfs:Class .	?u rdfs:subClassOf ?u .
rdfs11	?u rdfs:subClassOf ?v . ?v rdfs:subClassOf ?x .	?u rdfs:subClassOf ?x .
rdfs12	?u rdf:type rdfs:ContainerMembershipProperty .	?u rdfs:subPropertyOf rdfs:member .
rdfs13	?u rdf:type rdfs:Datatype .	?u rdfs:subClassOf rdfs:Literal .

# All 13 RDFS entailment rules (sorted)

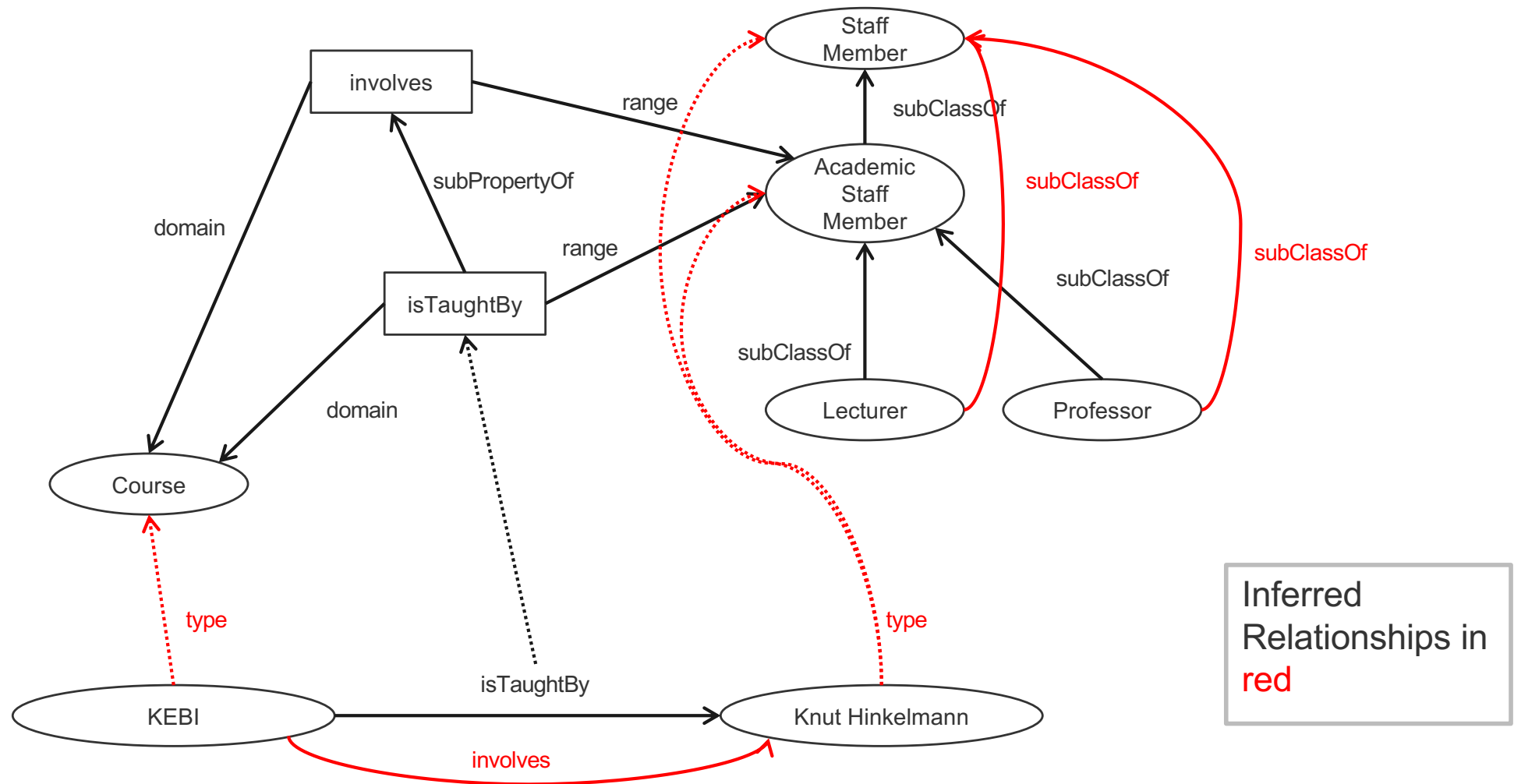
Rule Name	If E contains:	then add:
rdfs2	?p rdfs:domain ?x . ?u ?p ?y .	?u rdf:type ?x .
rdfs3	?p rdfs:range ?x . ?u ?p ?v .	?v rdf:type ?x .
rdfs5	?u rdfs:subPropertyOf ?v . ?v rdfs:subPropertyOf ?x .	?u rdfs:subPropertyOf ?x .
rdfs7	?p rdfs:subPropertyOf ?q . ?u ?p ?y .	?u ?q ?y .
rdfs9	?u rdfs:subClassOf ?x . ?v rdf:type ?u .	?v rdf:type ?x .
rdfs11	?u rdfs:subClassOf ?v . ?v rdfs:subClassOf ?x .	?u rdfs:subClassOf ?x .

Rule Name	If E contains:	then add:
rdfs6	?u rdf:type rdf:Property .	?u rdfs:subPropertyOf ?u .
rdfs10	?u rdf:type rdfs:Class .	?u rdfs:subClassOf ?u .
rdfs1	?u ?p ?n. where ?n is a plain literal (with or without a language tag).	_:nnn rdf:type rdfs:Literal . where _:nnn identifies a blank node allocated to ?n by rule <a href="#">rule lg</a> .
rdfs4a	?u ?p ?x .	?u rdf:type rdfs:Resource .
rdfs4b	?u ?p ?v.	?v rdf:type rdfs:Resource .
rdfs8	?u rdf:type rdfs:Class .	?u rdfs:subClassOf rdfs:Resource .
rdfs12	?u rdf:type rdfs:ContainerMembershipProperty .	?u rdfs:subPropertyOf rdfs:member .
rdfs13	?u rdf:type rdfs:Datatype .	?u rdfs:subClassOf rdfs:Literal .

# Example for Inferences made by Inference Rules (no Inference)

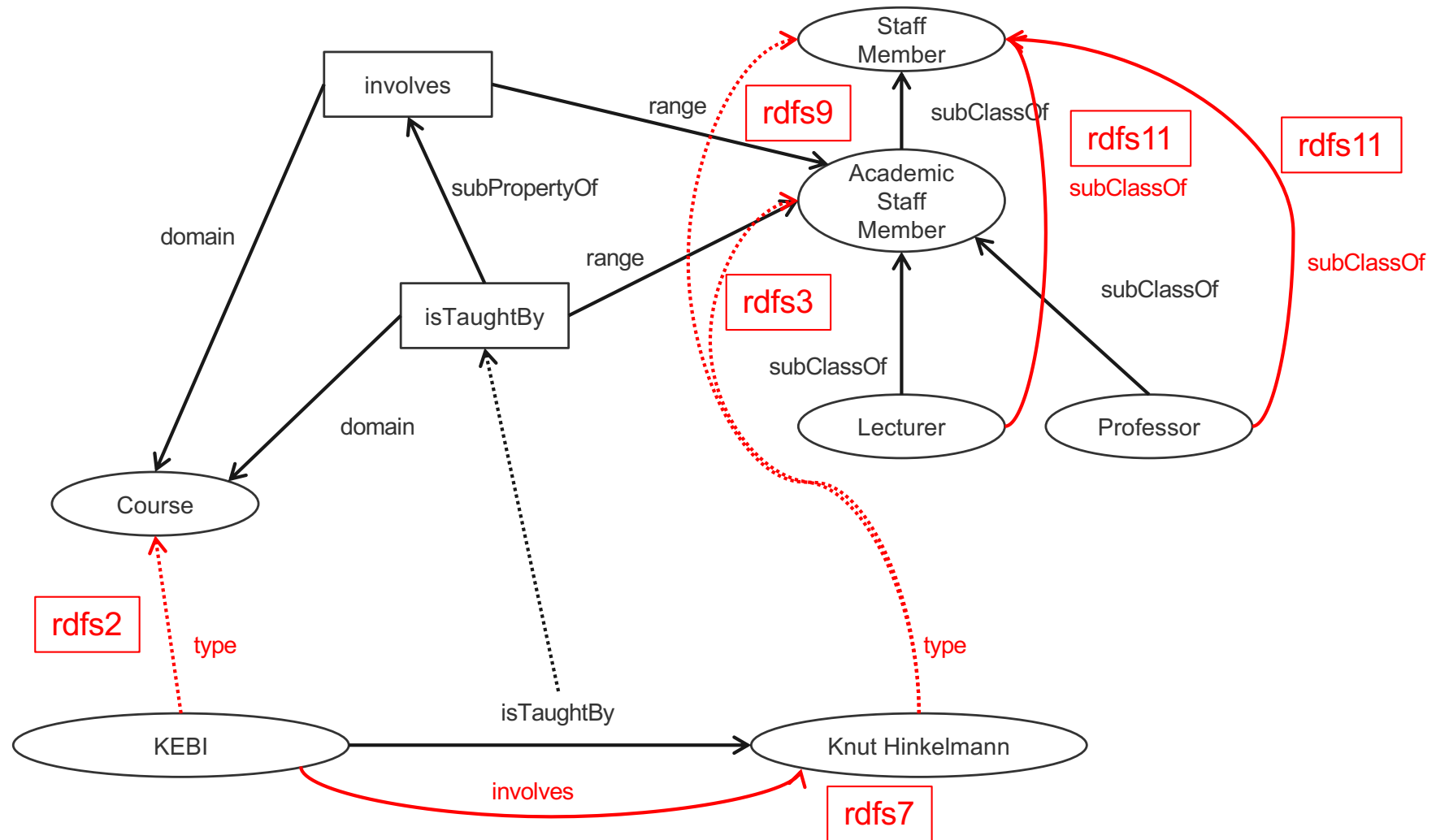


# Example for Inferences made by Inference Rules (“useful” inferences)



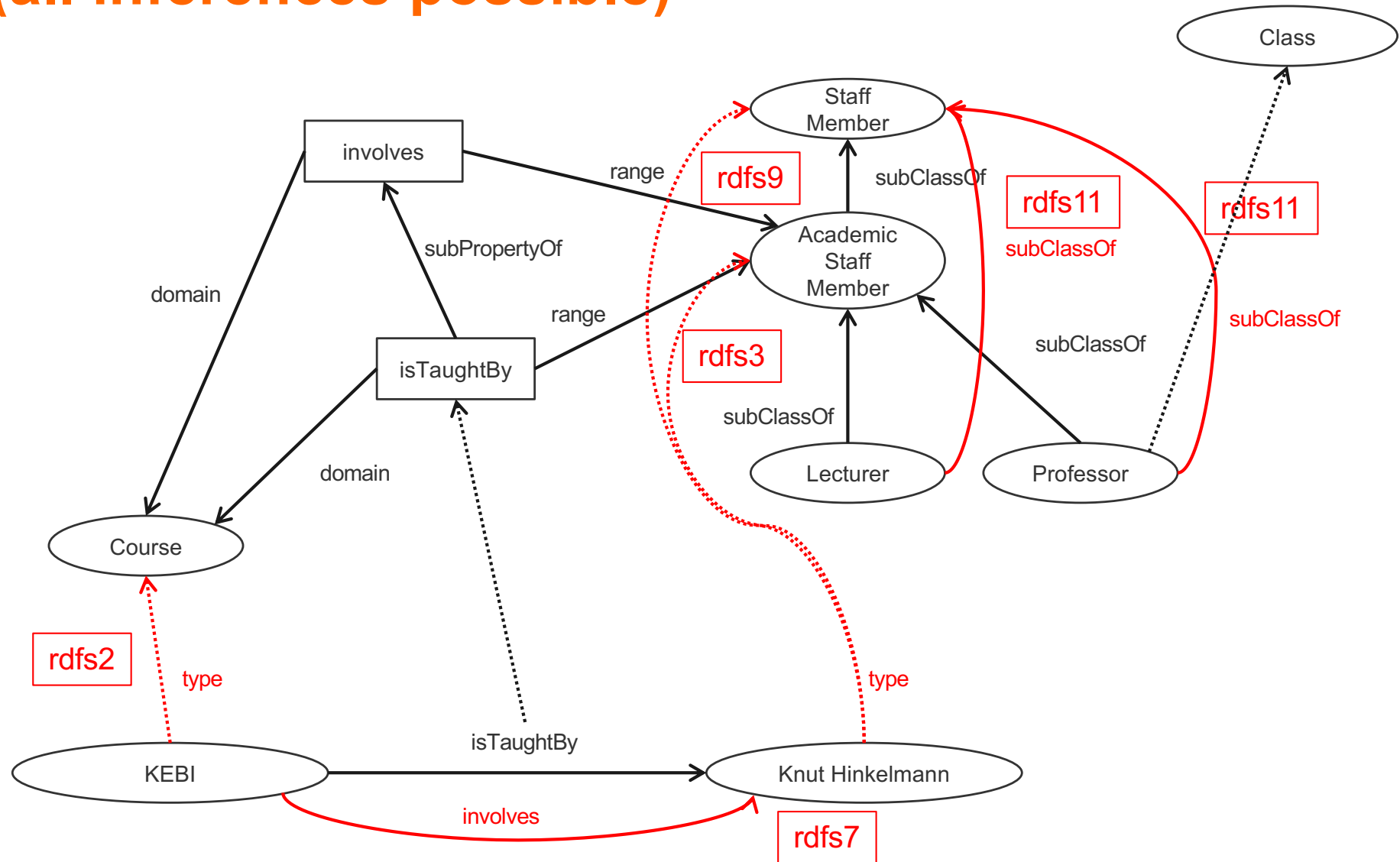
Inferred Relationships in red

# Example for Inferences made by Inference Rules (rules for “usefull” inferences)

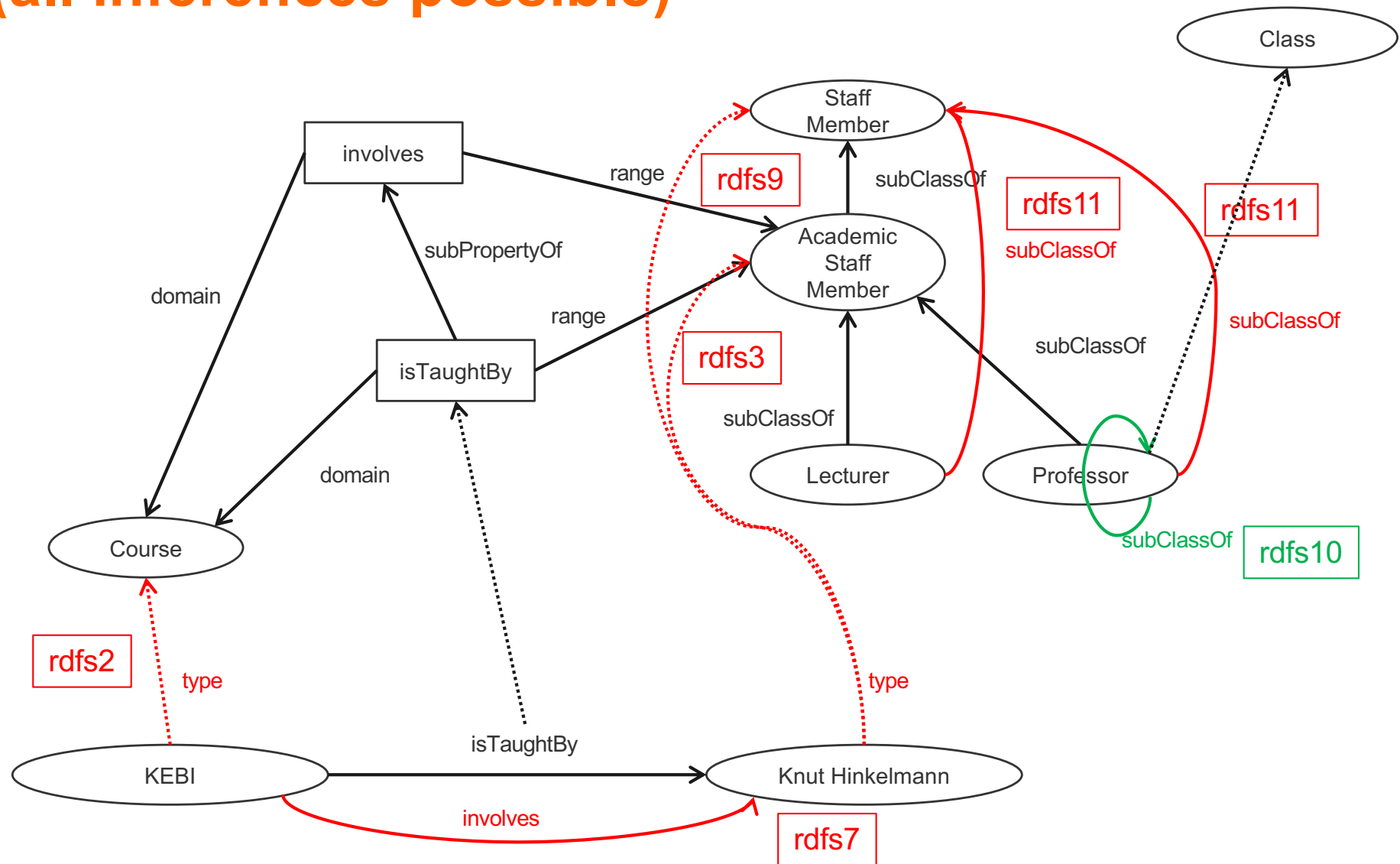




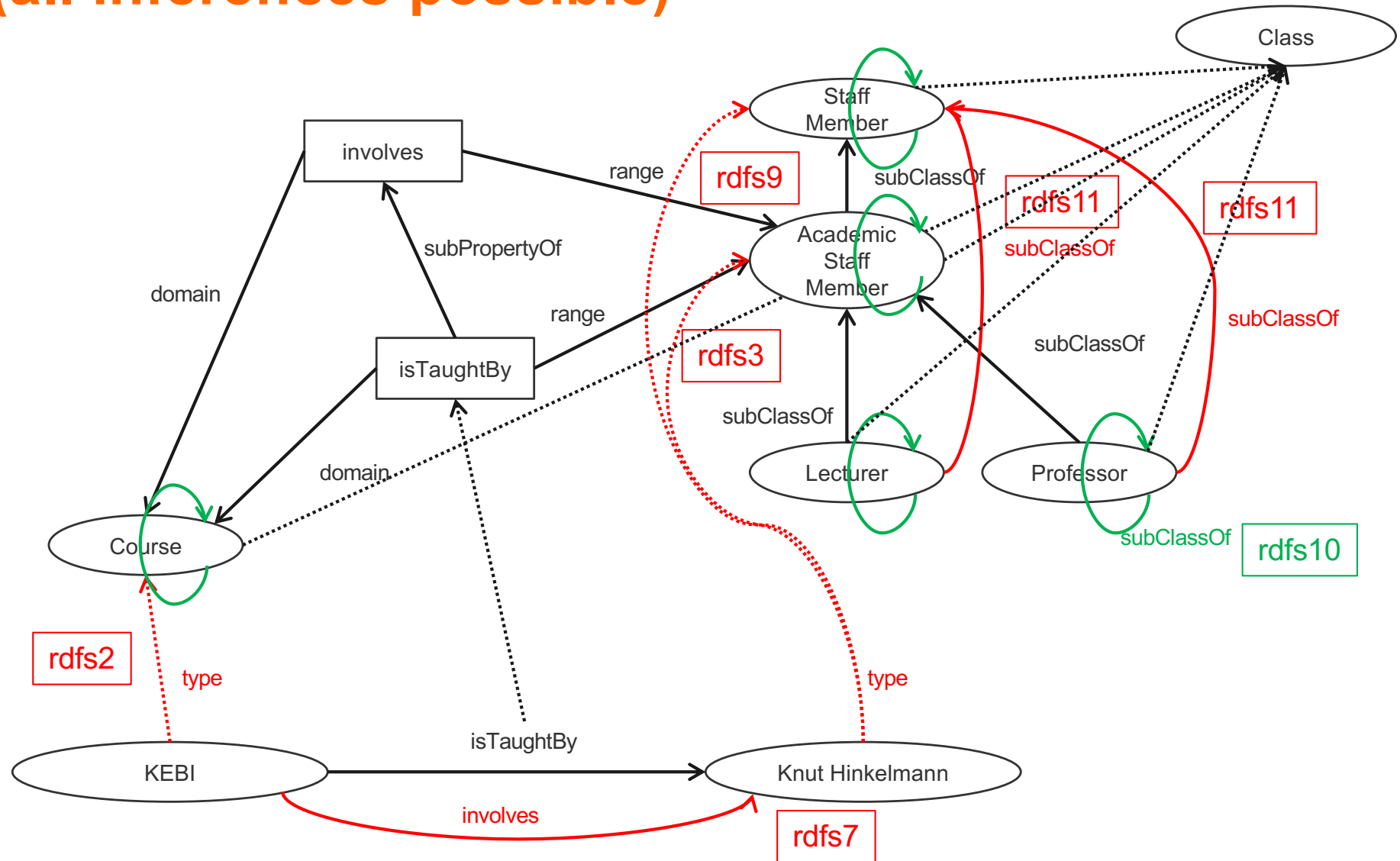
# Example for Inferences made by Inference Rules (all inferences possible)



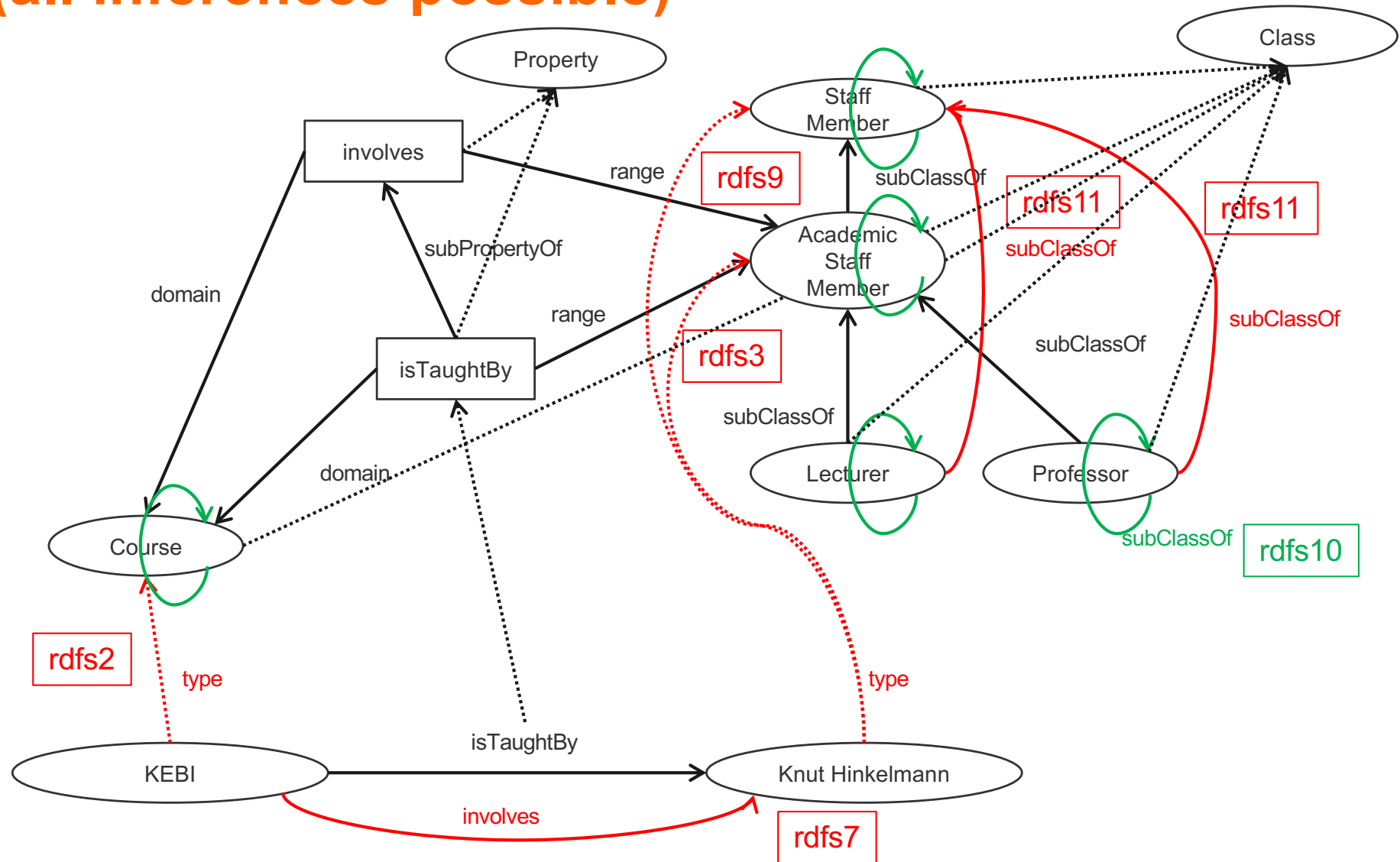
# Example for Inferences made by Inference Rules (all inferences possible)



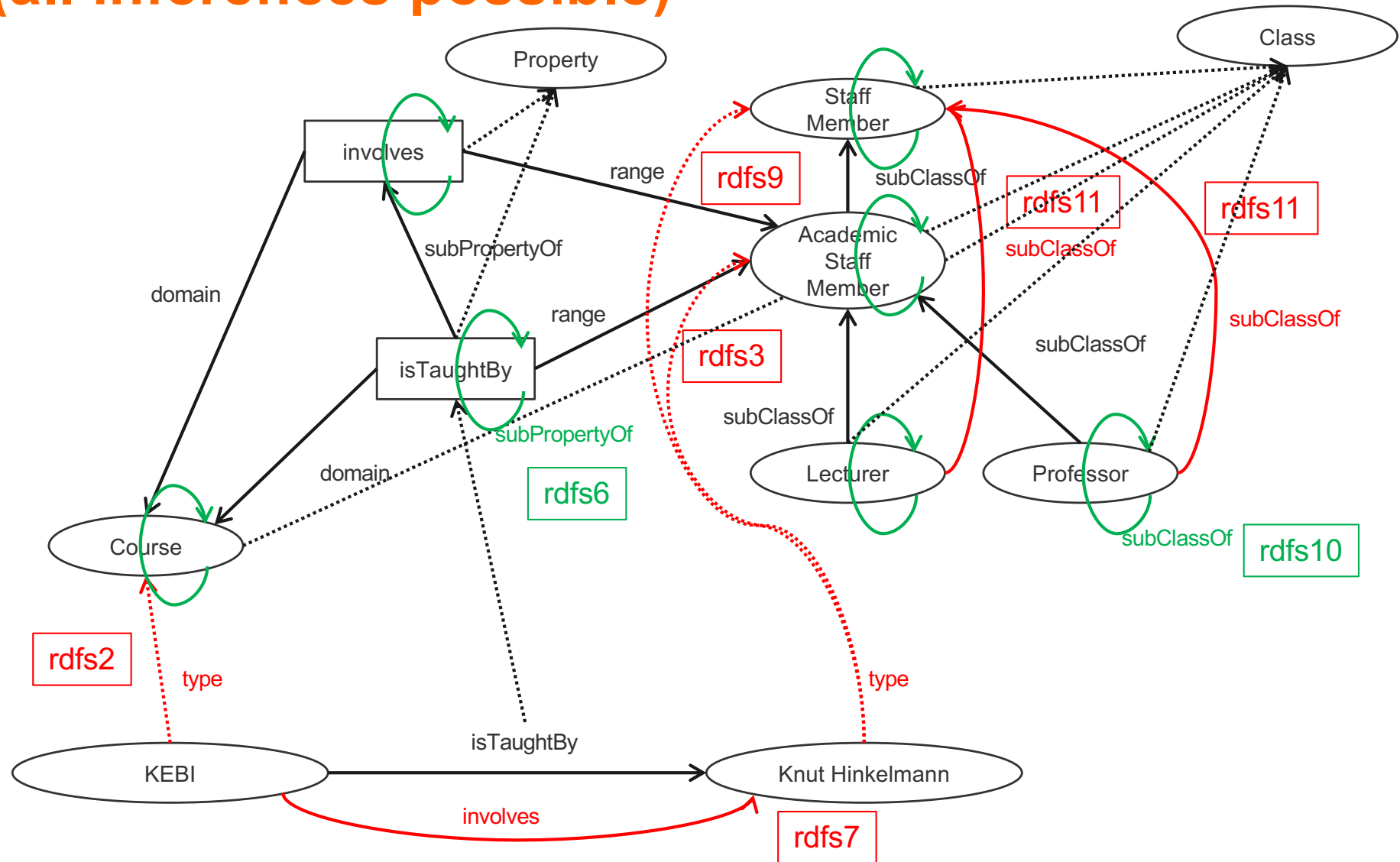
# Example for Inferences made by Inference Rules (all inferences possible)



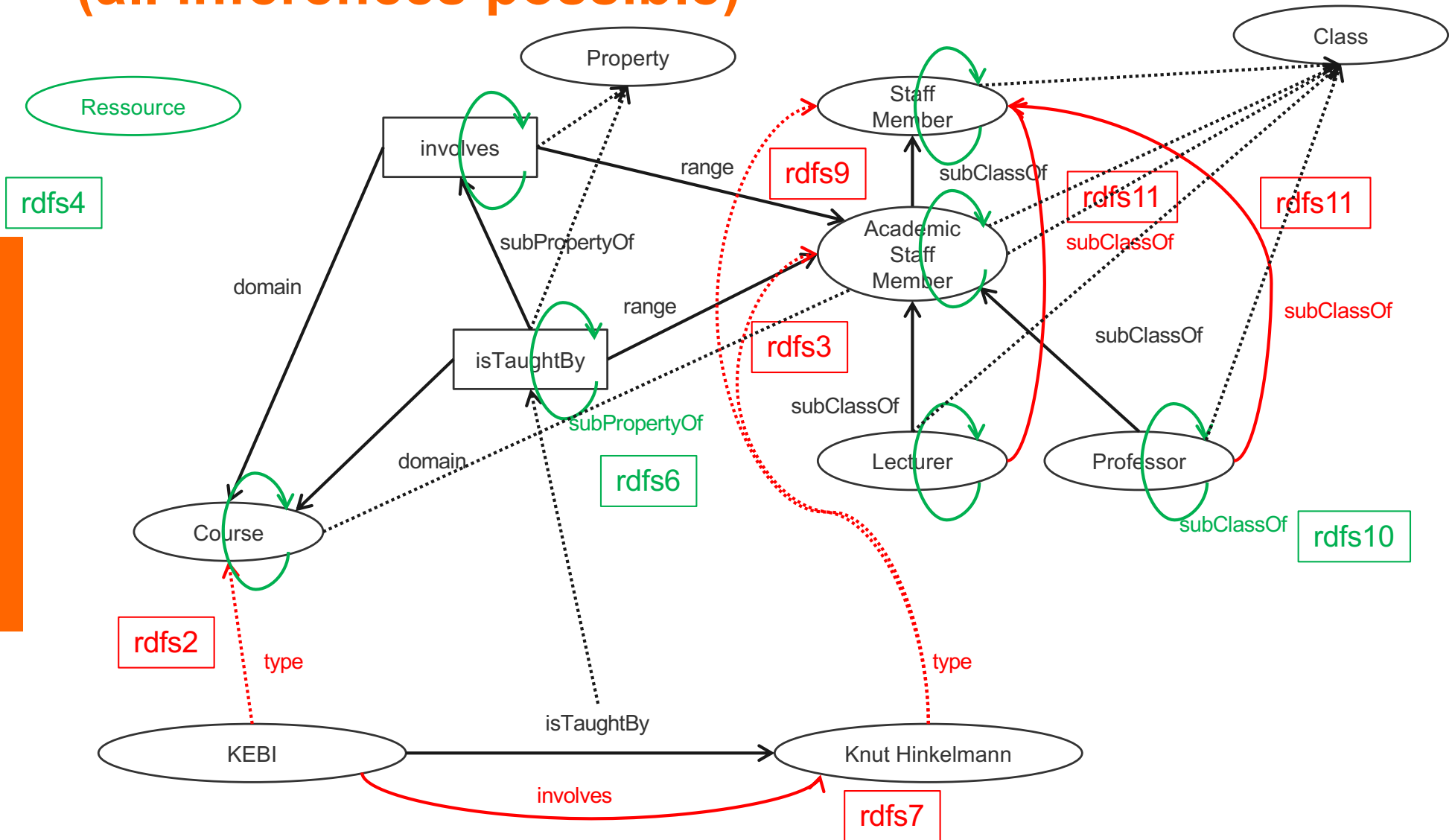
# Example for Inferences made by Inference Rules (all inferences possible)



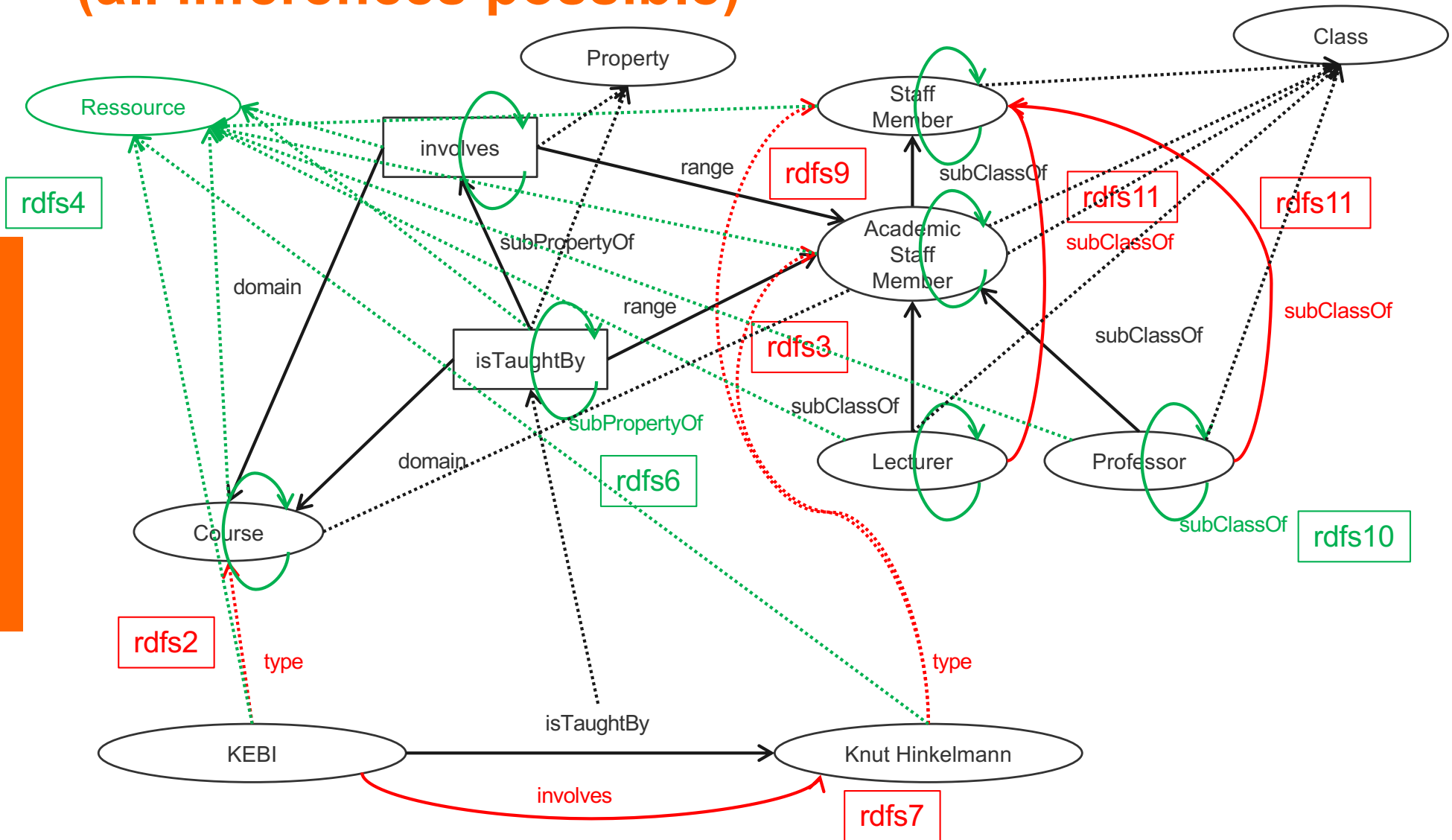
# Example for Inferences made by Inference Rules (all inferences possible)



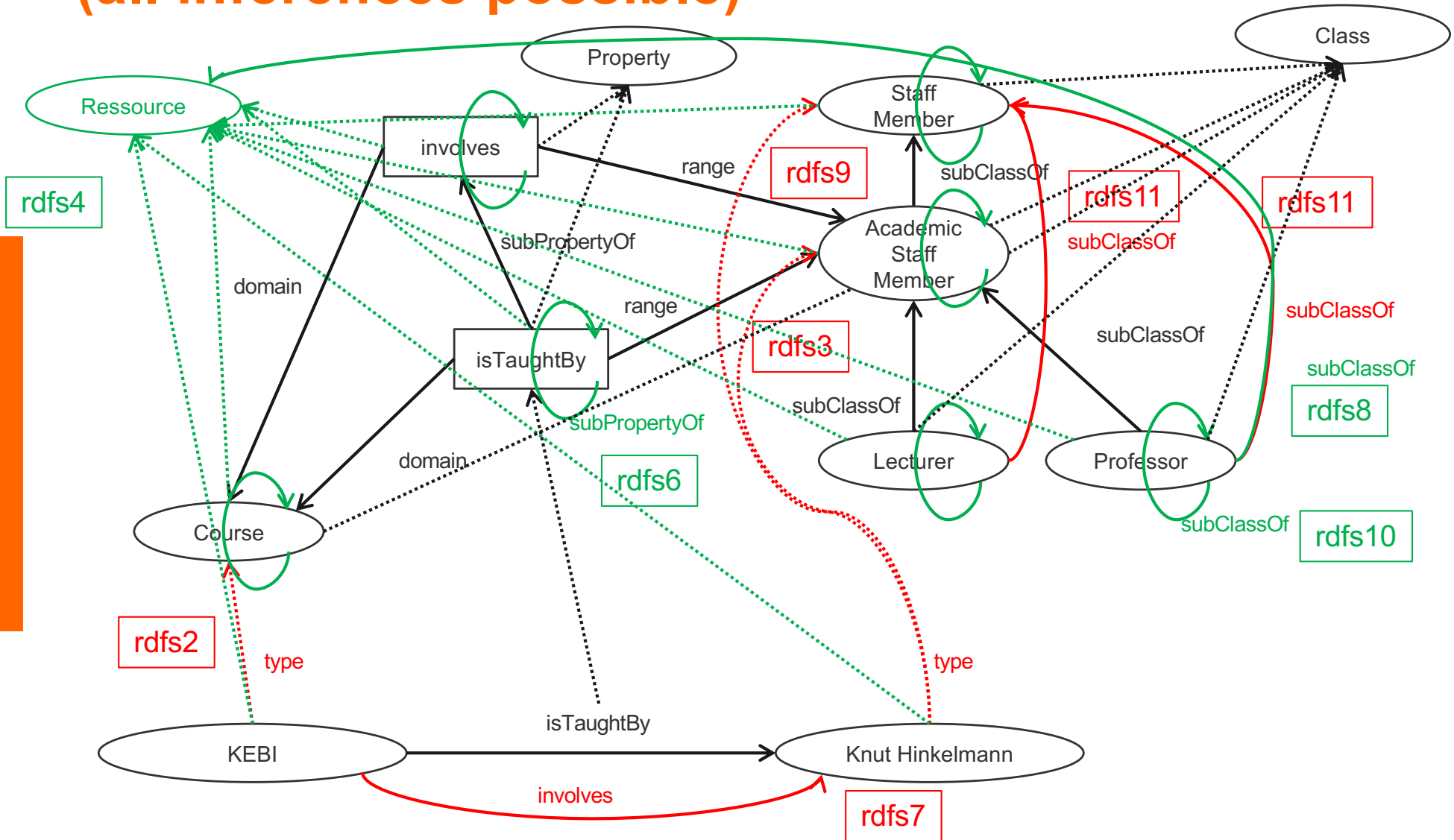
# Example for Inferences made by Inference Rules (all inferences possible)



# Example for Inferences made by Inference Rules (all inferences possible)



# Example for Inferences made by Inference Rules (all inferences possible)





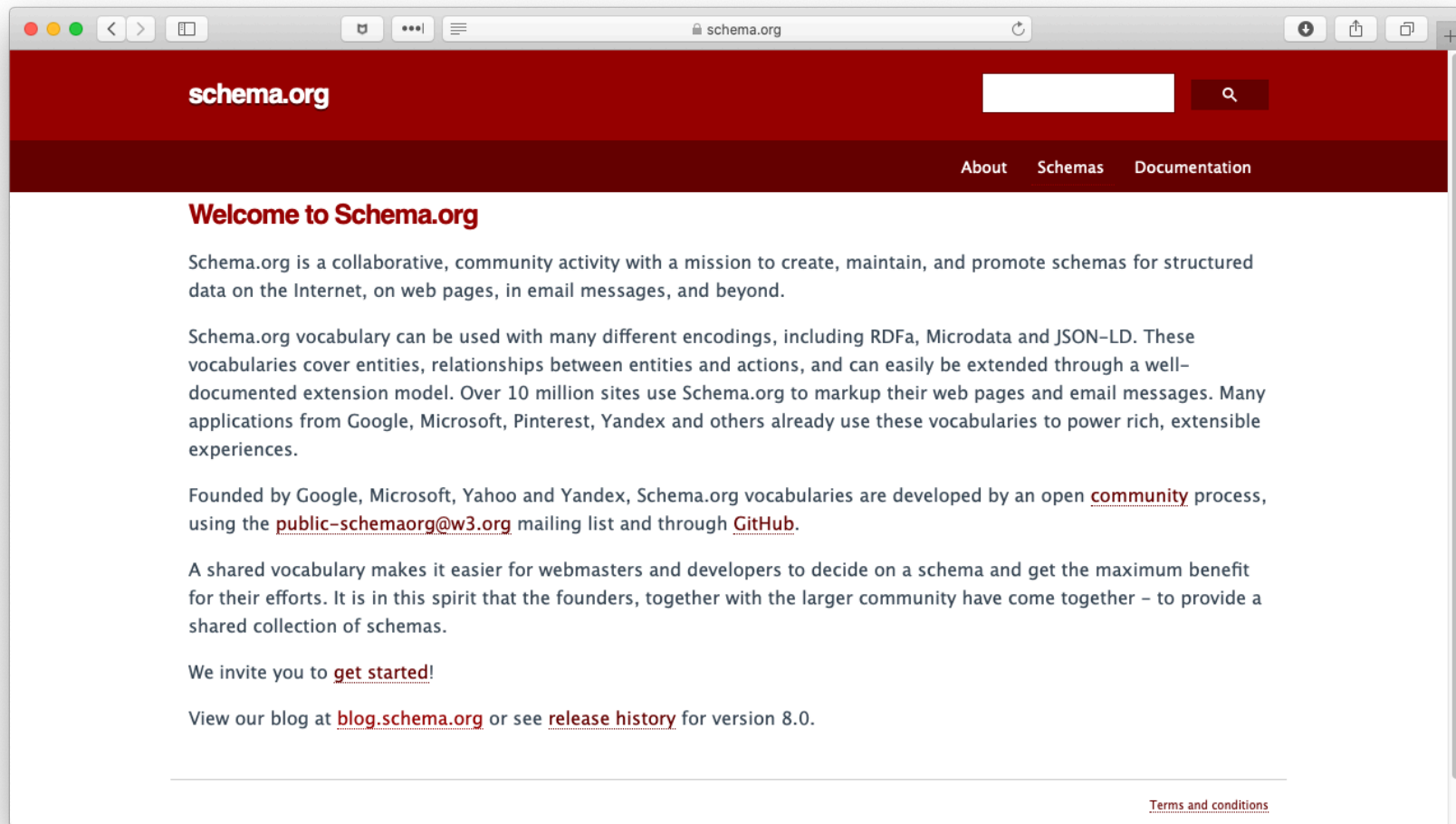
## Summary

- RDF has a graph-based data model
- RDF has an XML-based syntax to support syntactic interoperability.
  - ◆ XML and RDF complement each other because RDF supports semantic interoperability
- RDF is domain-independent
- RDF Schema provides a mechanism for describing specific domains
- RDF Schema is a primitive ontology language
  - ◆ It offers certain modelling primitives with fixed meaning
- Key concepts of RDF Schema are class, subclass relations, property, subproperty relations, and domain and range restrictions
- There exist query languages for RDF and RDFS



# GOOGLE'S KNOWLEDGE GRAPHS

# schema.org



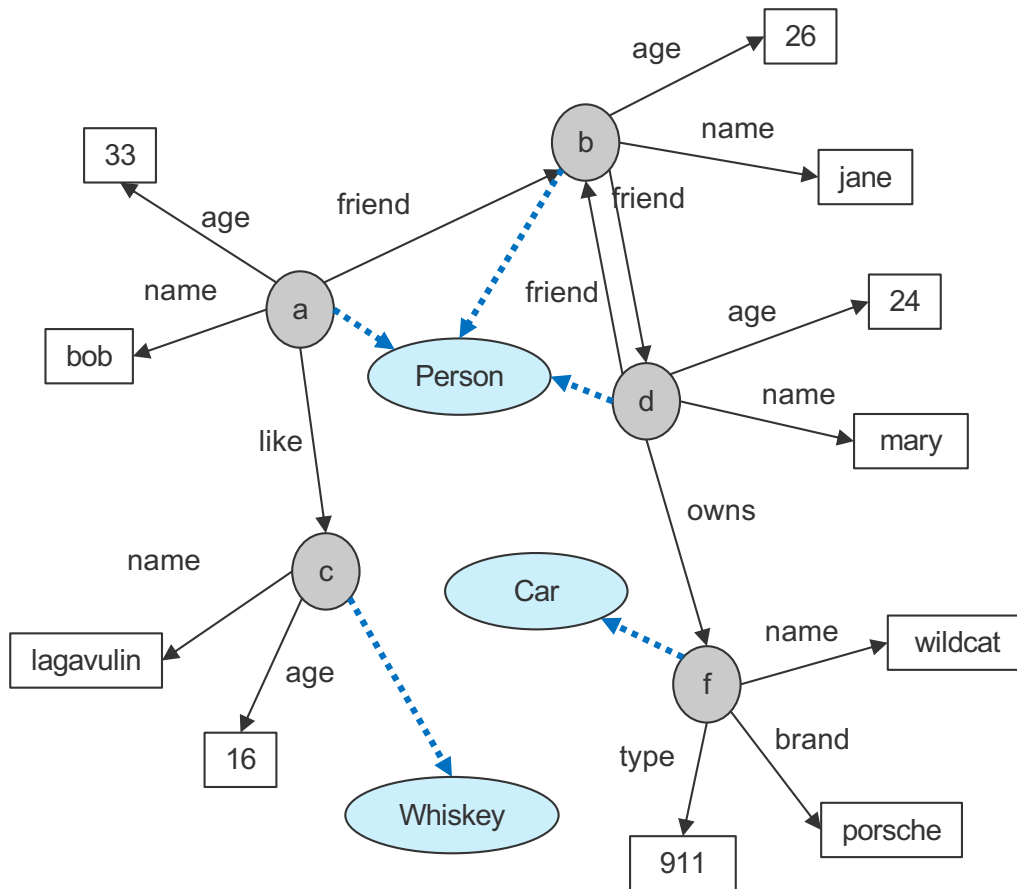


# SPARQL: A QUERY LANGUAGE

# Example: Querying

```

name(a,bob).      triple(a,name,bob).
age(a,33).        triple(a,age,33).
friend(a,b).      triple(a,friend,b).
like(a,c).        triple(a,like,c).
name(b,jane).     triple(b,name,jane).
age(b,26).        triple(b,age,26).
friend(b,d).      triple(b,friend,d).
    
```



Formulate the following queries

- Who has whom as friend?  
 ?- friend(X,Y).  
 ?- triple(X,friend,Y).  
 SELECT ?X ?Y  
 WHERE { ?X friend ?Y. }
- Who are the friend-of-a-friend of whom?  
 ?- friend(X,Y), friend(Y,Z).  
 ?- triple(X,friend,Y), triple(Y,friend,Z).  
 SELECT ?X ?Z  
 WHERE { ?X friend ?Y.  
 ?Y friend ?Z. }
- Who has friends older than 25  
 ?- friend(X,Y), age(Y,A), A > 25.  
 ?- triple(X,friend,Y), triple(Y,age,A), A > 25.  
 SELECT ?X ?Y ?A  
 WHERE { ?X friend ?Y.  
 ?Y age ?A.  
 FILTER ?A > 25. }

# SPARQL Query Syntax

SPARQL similar to select-from-where syntax (like SQL):

- *PREFIX*: prefix information

`prefix`

`uni: <http://www.fhnw.ch/schema.rdfs#>`

- *SELECT*: the entities (variables) you want to return

`select ?X ?Y ?A`

- *WHERE*: the (sub)graph you want to get the information from

`where { ?X friend ?Y. ?Y age ?A.`

- additional constraints on objects, using operators

`FILTER ?A > 25. }`

# SPARQL

## ■ It provides facilities to:

- ◆ Extract information in the form of URIs, blank nodes, plain and typed literals
- ◆ Extract RDF subgraphs
- ◆ Construct new RDF graphs based on information in the queried graphs

## ■ Feature

- ◆ Matching graph patterns
- ◆ Query terms – based on Turtle syntax
- ◆ Terms delimited by "<>" are relative URI references
- ◆ Data description format - Turtle

# Query forms

## ■ SELECT

- ◆ returns all, or a subset of the variables bound in a query pattern match
- ◆ returned in a table
- ◆ formats : XML or RDF/XML

## ■ CONSTRUCT

- ◆ returns an RDF graph constructed by substituting variables in a set of
- ◆ triple templates

## ■ DESCRIBE

- ◆ returns an RDF graph that describes the resources found.

## ■ ASK

- ◆ returns whether a query pattern matches or not.



## Query result: example

- Query: “return all those which has friends older than 25 with the cities they contain, and their areacodes, if known”

```
select ?X ?Y ?A
where { ?X friend ?Y. ?Y age ?A.
       FILTER ?A > 25 }
```

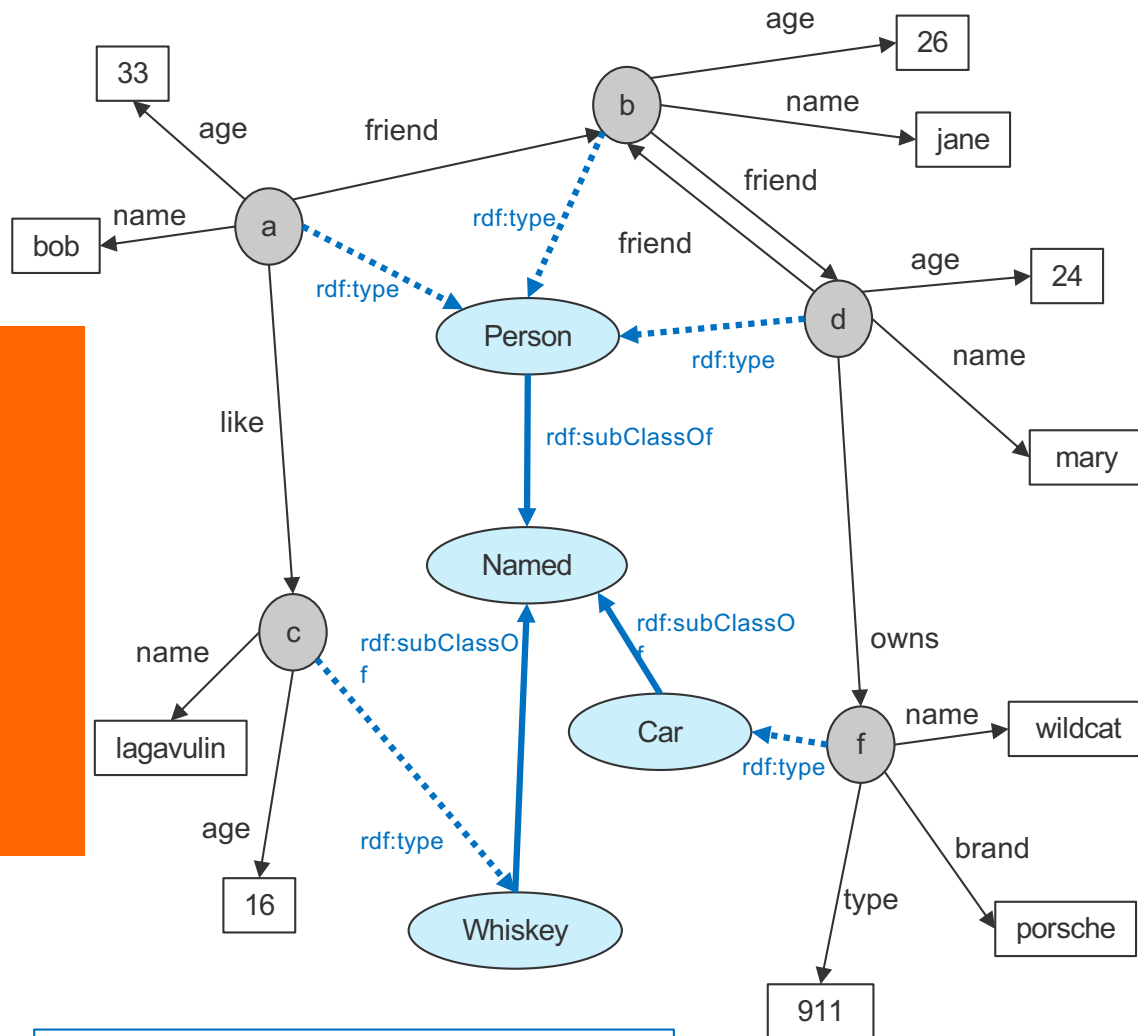
- Result (table of bindings):

X	Y	A
a	b	26
c	b	26

## Schema Querying

- SPARQL has support for Schema querying
  - ◆ Class instances
  - ◆ Subclasses, Subproperties
  - ◆ etc.
  
- SPARQL “interprets” RDF(S) semantics
  - ◆ RDF and RDFS predicates explicitly mapped to their formal semantics
    - Transitivity of subClassOf property, inheritance of class instances, etc.
  - ◆ So it is not just querying the data graph (but the graph which is computed virtually with the entailment rules)

# Example: Schema Quering



**name rdfs:domain Named**

Formulate the following queries

- Who are Persons?

```
?- triple(X,rdf:type,Person).
SELECT ?X
WHERE { ?X rdf:type Person.}
```
- Which are named items?

```
?- triple(X,rdf:type,Named).
SELECT ?X
WHERE { ?X rdf:type Named.}
```
- What are the subclasses of Named?

```
SELECT ?X ?Y
WHERE { ?X rdfs:subClassOf ?Y.}
```