CC7220-1
La Web de Datos
Primavera 2022

Lecture 3: RDF Schema (RDFS) and Semantics

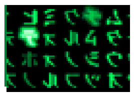
Aidan Hogan
aidhog@gmail.com

Last time …

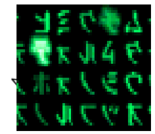
*MORE OR LESS

SEMANTIC WEB: DATA

DATA:

Ireland 

(Ireland,partOf,Europe)
(Ireland,isA,Country)
(Ireland,capital,Dublin)

Dublin 

(Ireland,capital,Dublin)
(Dublin,population,1000000)

LOGIC: "(b,capital,a) $\rightarrow$ (a,partOf,b)"
"(a,partOf,b), (b,partOf,c) $\rightarrow$ (a,partOf,c)"

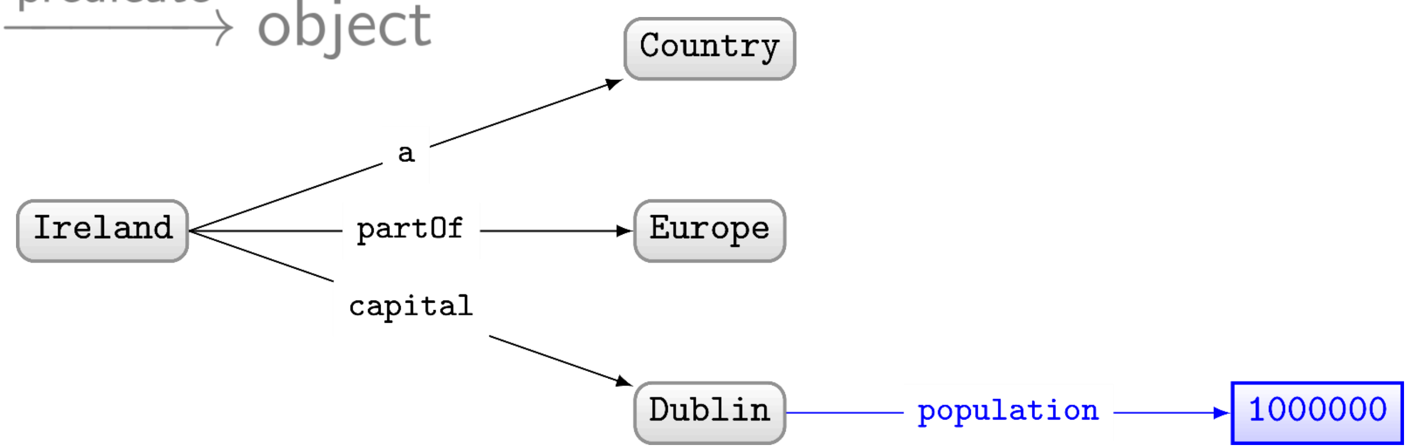
QUERY: "(x,partOf,y)?"

OUTPUT: {($x \mapsto$ Ireland, $y \mapsto$ Europe),
($x \mapsto$ Dublin, $y \mapsto$ Ireland),
($x \mapsto$ Dublin, $y \mapsto$ Europe)}

RDF often drawn as a (directed, labelled) graph

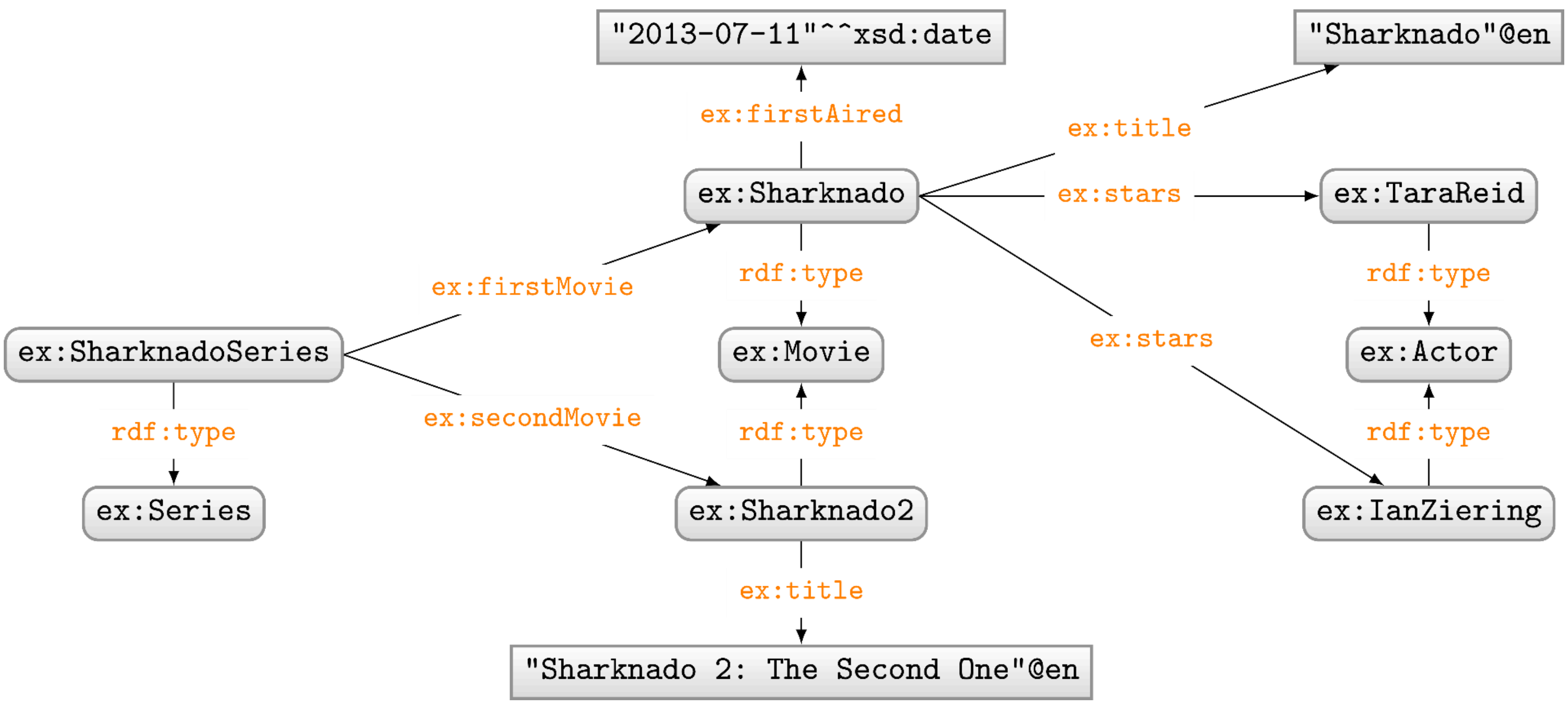
subject	*predicate*	*object*
Ireland	partOf	Europe
Ireland	a	Country
Ireland	capital	Dublin
Dublin	population	1,000,000

$\text{subject} \xrightarrow{\text{predicate}} \text{object}$



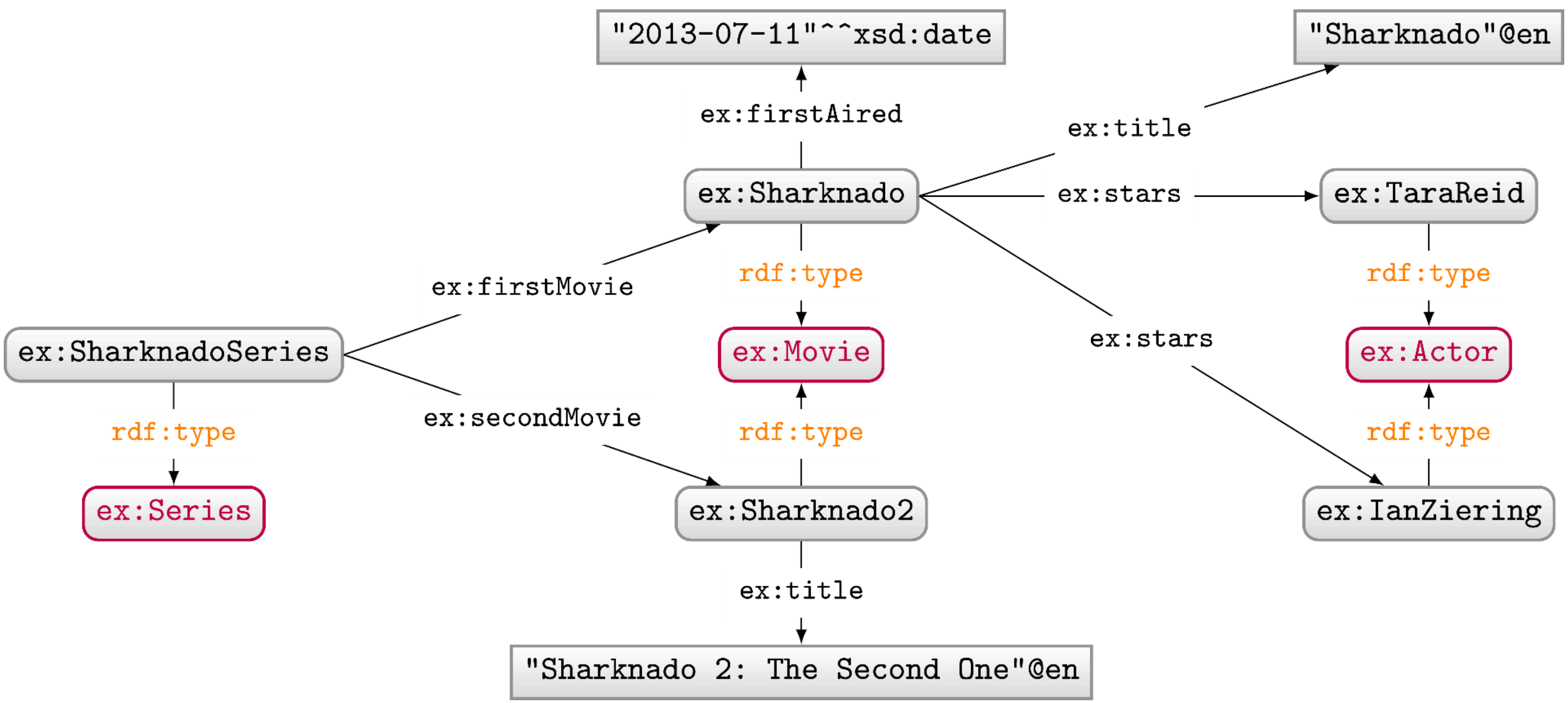
RDF PROPERTIES

- RDF Terms used as predicate
  - `rdf:type`, `ex:firstMovie`, `ex:stars`, etc.



RDF CLASSES

- Used to conceptually group resources
  - ex:Movie, ex:Actor, ex:Series, etc.
  - Uses predicate rdf:type to type a resource



TODAY'S TOPIC ...

Semantic Web: Logic

Data:

Ireland

(Ireland,partOf,Europe)
(Ireland,isA,Country)
(Ireland,capital,Dublin)

Dublin

(Ireland,capital,Dublin)
(Dublin,population,1000000)

Logic: "(b,capital,a) $\rightarrow$ (a,partOf,b)"
"(a,partOf,b), (b,partOf,c) $\rightarrow$ (a,partOf,c)"

Query: "(x,partOf,y)?"

Output: $\{$($x \mapsto$ Ireland, $y \mapsto$ Europe),
($x \mapsto$ Dublin, $y \mapsto$ Ireland),
($x \mapsto$ Dublin, $y \mapsto$ Europe)$\}$

How to capture logic?

How should we capture logic on the Semantic Web?

Logic: “(b,capital,a) $\rightarrow$ (a,partOf,b)”

“(a,partOf,b), (b,partOf,c) $\rightarrow$ (a,partOf,c)”

Semantic Web Answer: Schema/Ontologies

- Instead of rules, we can use RDF!
- Define relationships between classes and properties

What sorts of relationships might be useful to define between the following classes and properties?

ex:Town

ex:City

ex:Country

ex:Place

foaf:Person

ex:CapitalCity

ex:hasCapitalCity

ex:hasPart

foaf:familyName

ex:hasCity

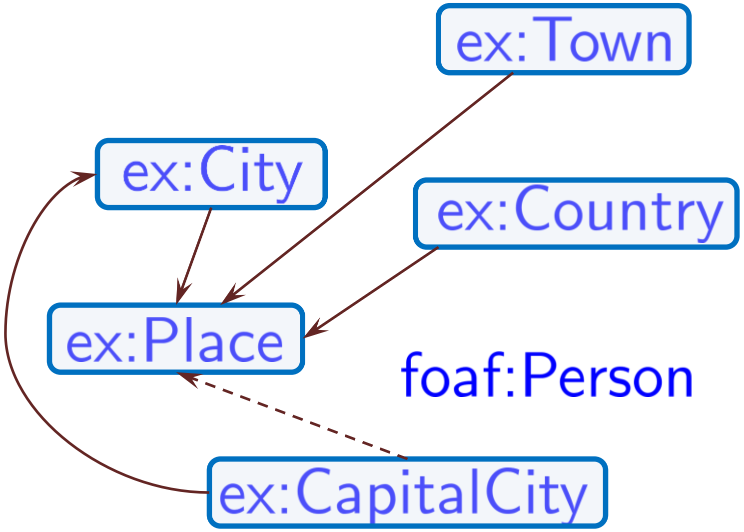
ex:containsPlace

CLASS HIERARCHY

- Class c is a sub-class of Class d
  - If (x,rdf:type,c) then (x,rdf:type,d),

Example: if ex:CapitalCity *sub-class of* ex:City
and if (ex:Dublin,rdf:type,ex:CapitalCity)
then (ex:Dublin,rdf:type,ex:City)

Which classes would be sub-classes of each other?

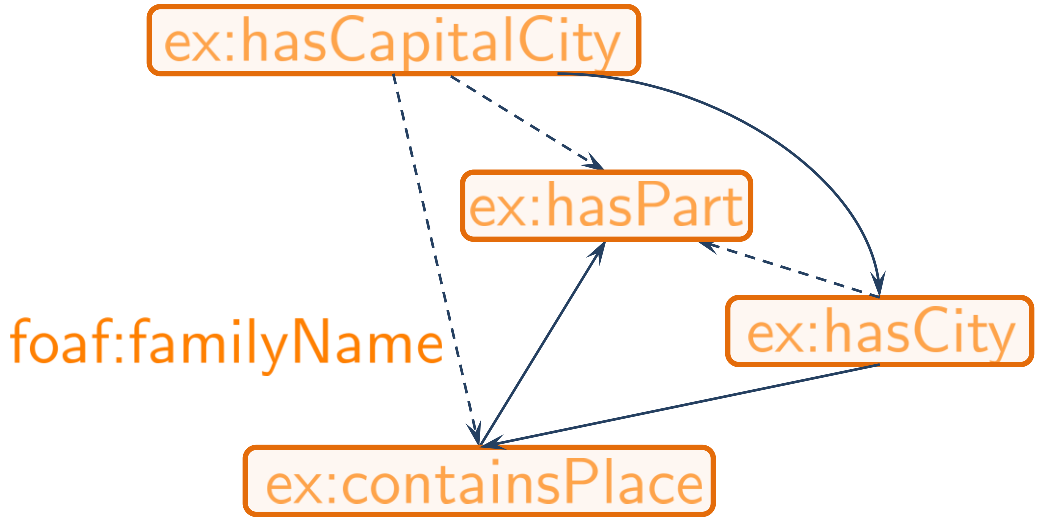


PROPERTY HIERARCHY

- Property p is a sub-property of q
  - If (x,p,y) then (x,q,y)

Example: if ex:hasCapitalCity *sub-property of* ex:hasCity
and if (ex:Ireland,ex:hasCapitalCity,ex:Dublin)
then (ex:Ireland,ex:hasCity,ex:Dublin)

Which properties would be sub-properties of each other?

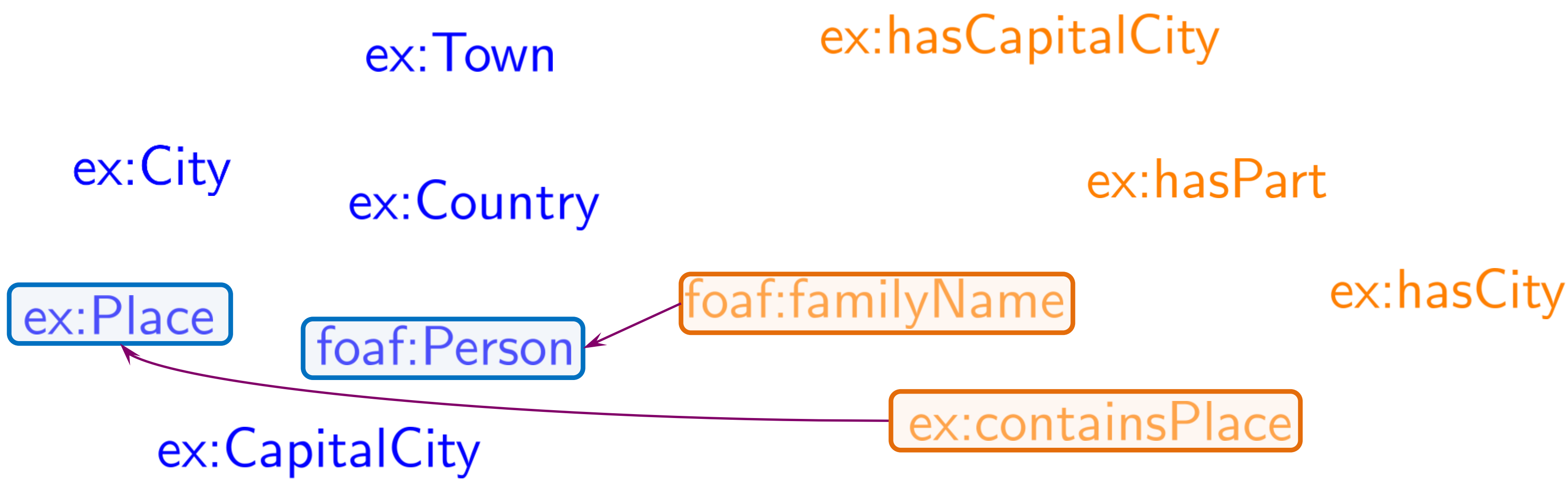


Domain of properties

- Property p has domain class c
  - If (x,p,y) then (x,rdf:type,c)

Example: if foaf:familyName *has domain* foaf:Person
and if (ex:Aidan,foaf:familyName,"Hogan")
then (ex:Aidan,rdf:type,foaf:Person)

Which properties would have which classes as domain?

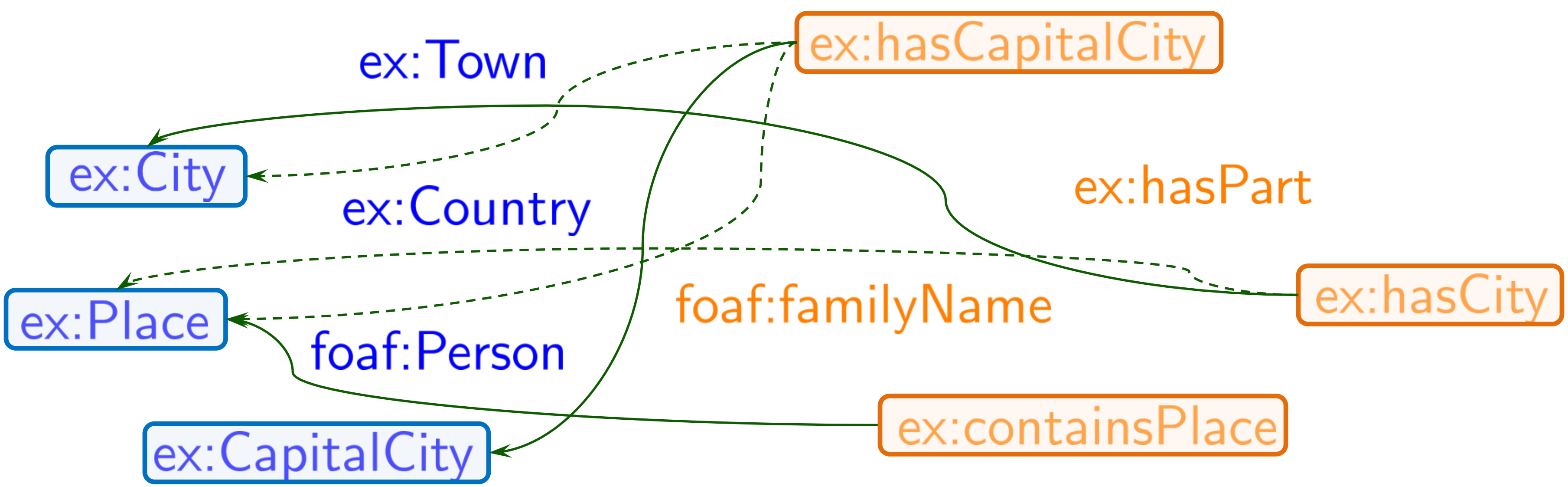


Range of properties

- Property p has range class c
  - If (x,p,y) then (y,rdf:type,c)

Example: if ex:hasCity *has range* ex:City
and if (ex:Ireland,ex:hasCity,ex:Dublin)
then (ex:Dublin,rdf:type,ex:City)

Which properties would have which classes as range?



Trade-off: More Specific / Less Reusable

- More specific → more conclusions
- Less specific → more reusable

Example: `ex:hasCapitalCity` has domain `ex:Country`

PRO: Know that anything that has a capital city is a country

CON: Cannot use for capitals of states, regions, etc.

Trade-off: More Specific / Less Reusable

- Another example:
  - `ex:Mayor` sub-class of `foaf:Person`

Bosco the dog

Mayor of Sunol, California

1981–1994

R.I.P.

Trade-off: More Specific / Less Reusable

- Another example:
  - `ex:spouse` has domain/range `foaf:Person`

Erika Eiffel

Married Eiffel Tower in 2007

Beware of "hidden" definitions!

FOAF Vocabulary Specification 0.99

Namespace Document 14 January 2014 - *Paddington Edition*

Property: foaf:img

image - An image that can be used to represent some thing (ie. those depictions which are particularly representative of something, eg. one's photo on a homepage).

Status: testing

Domain: having this property implies being a Person

Range: every value of this property is a Image

Any potential problems here?

`(ex:Dublin,foaf:img,ex:Dublin_night.jpg)`

Choose names of properties/classes carefully!

RDFS: RDF Schema

RDFS (1.1): A Web Standard

http://www.w3.org/TR/rdf-schema/

W3C Recommendation

RDF Schema 1.1

W3C Recommendation 25 February 2014

This version:
http://www.w3.org/TR/2014/REC-rdf-schema-20140225/
Latest published version:
http://www.w3.org/TR/rdf-schema/
Previous version:
http://www.w3.org/TR/2014/PER-rdf-schema-20140109/
Editors:
Dan Brickley, Google
R.V. Guha, Google
Previous Editors:
Brian McBride

Please check the **errata** for any errors or issues reported since publication.

This document is also available in this non-normative format: diff w.r.t. 2004 Recommendation

RDFS: Describe "schema" in RDF

`@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.`

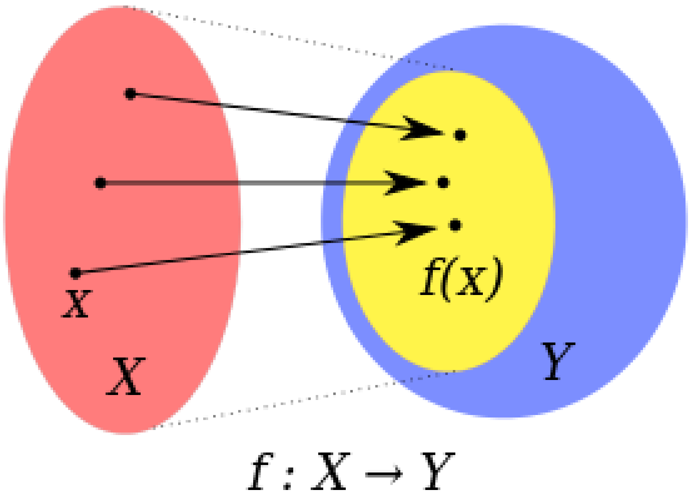
- Sub-class
  - `ex:CapitalCity rdfs:subClassOf ex:City .`
- Sub-property
  - `ex:hasCapitalCity rdfs:subPropertyOf ex:hasCity .`
- Domain
  - `foaf:familyName rdfs:domain foaf:Person .`
- Range
  - `ex:hasCapitalCity rdfs:range ex:CapitalCity .`
  - `foaf:familyName rdfs:range xsd:string .`

Note: Why called "domain" and "range"?

Any guesses why RDFS calls these "domain" and "range"?

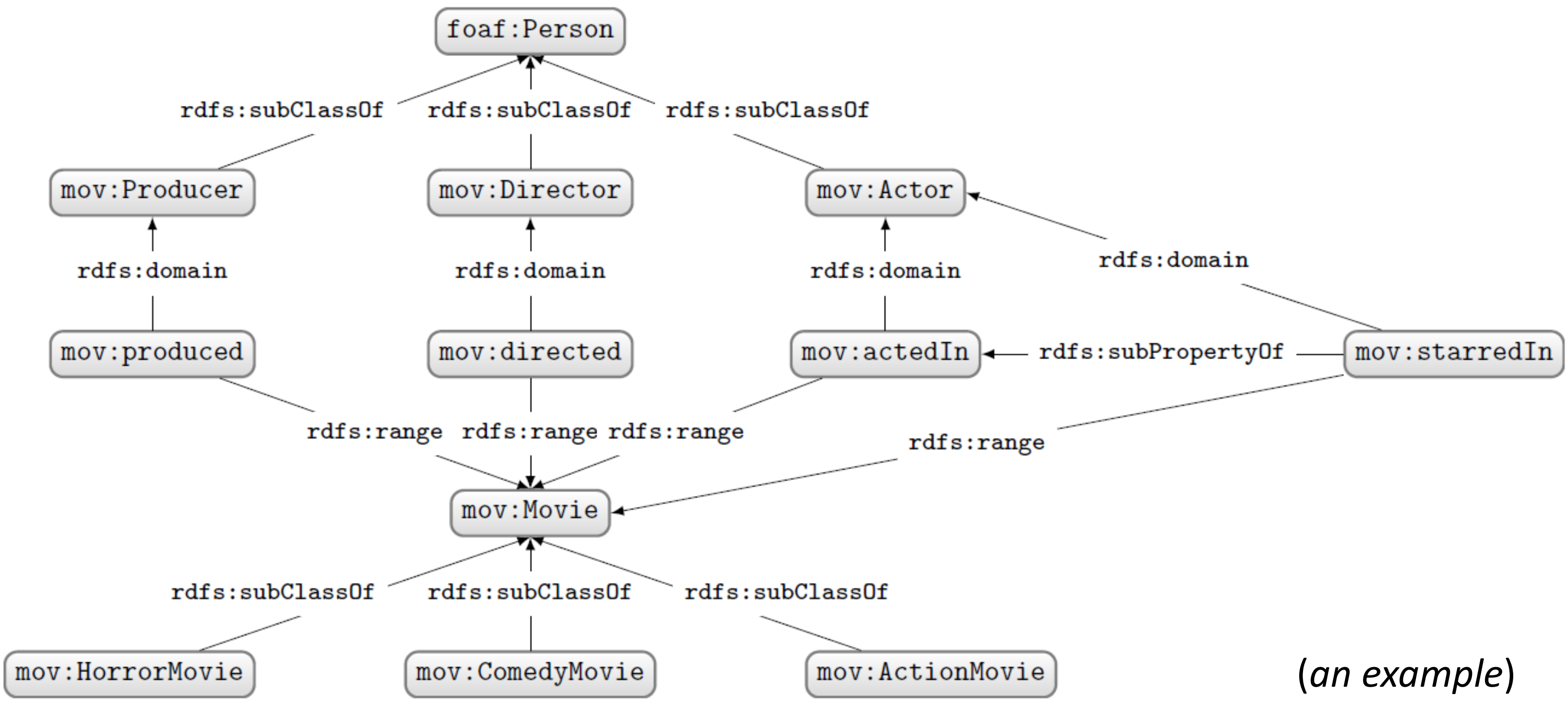
$f : X \rightarrow Y$

- X : **domain** of the function
- Y : **co-domain** of the function
- $\{f(x) \mid x \in X\}$: image or **range** of the function



So let's build an RDF Schema …

Let's model an RDF Schema for movies, including different types of movies (horror, comedy, action), some different types of people involved (actor, producer, director), and how they are related.



(*an example*)

But what, e.g., is the domain of … ?

BUT WHAT, E.G., IS THE DOMAIN OF ... ?

- `rdfs:Resource` the class of everything!
  - Yes, even itself!
    - `(rdfs:Resource,rdf:type,rdfs:Resource)`

(Giving domain/range/sub-class as `rdfs:Resource` says nothing new!)

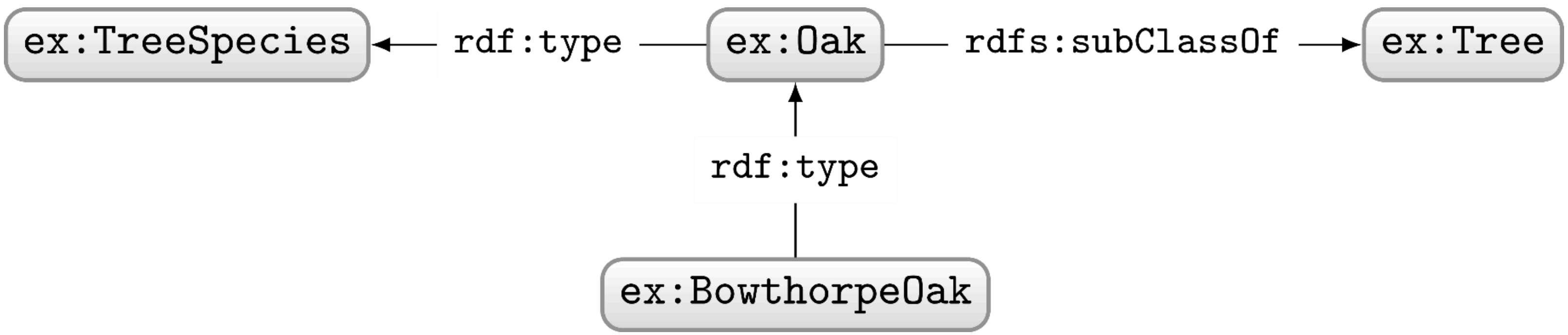
Some meta-classes …

- `rdf:Property`: class of all properties
  - `(ex:hasCity,rdf:type,rdf:Property)`

- `rdfs:Class`: class of all classes
  - `(ex:City,rdf:type,rdfs:Class)`

NOTE: CLASS OR INSTANCE?

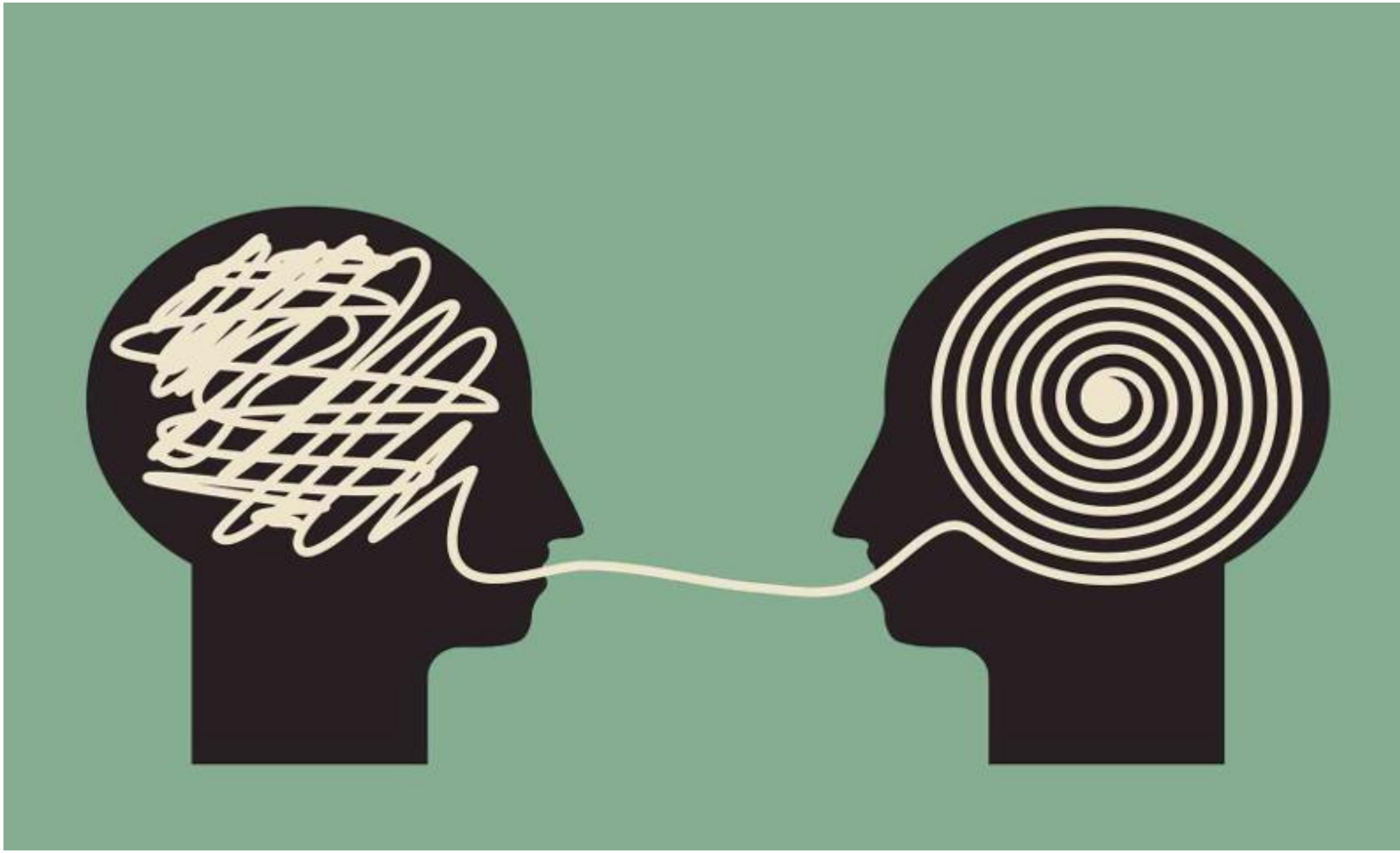
Would you define ex:Oak ("roble"@es)
as a class or an instance?



Classes can also "act" as instances: no strong distinction

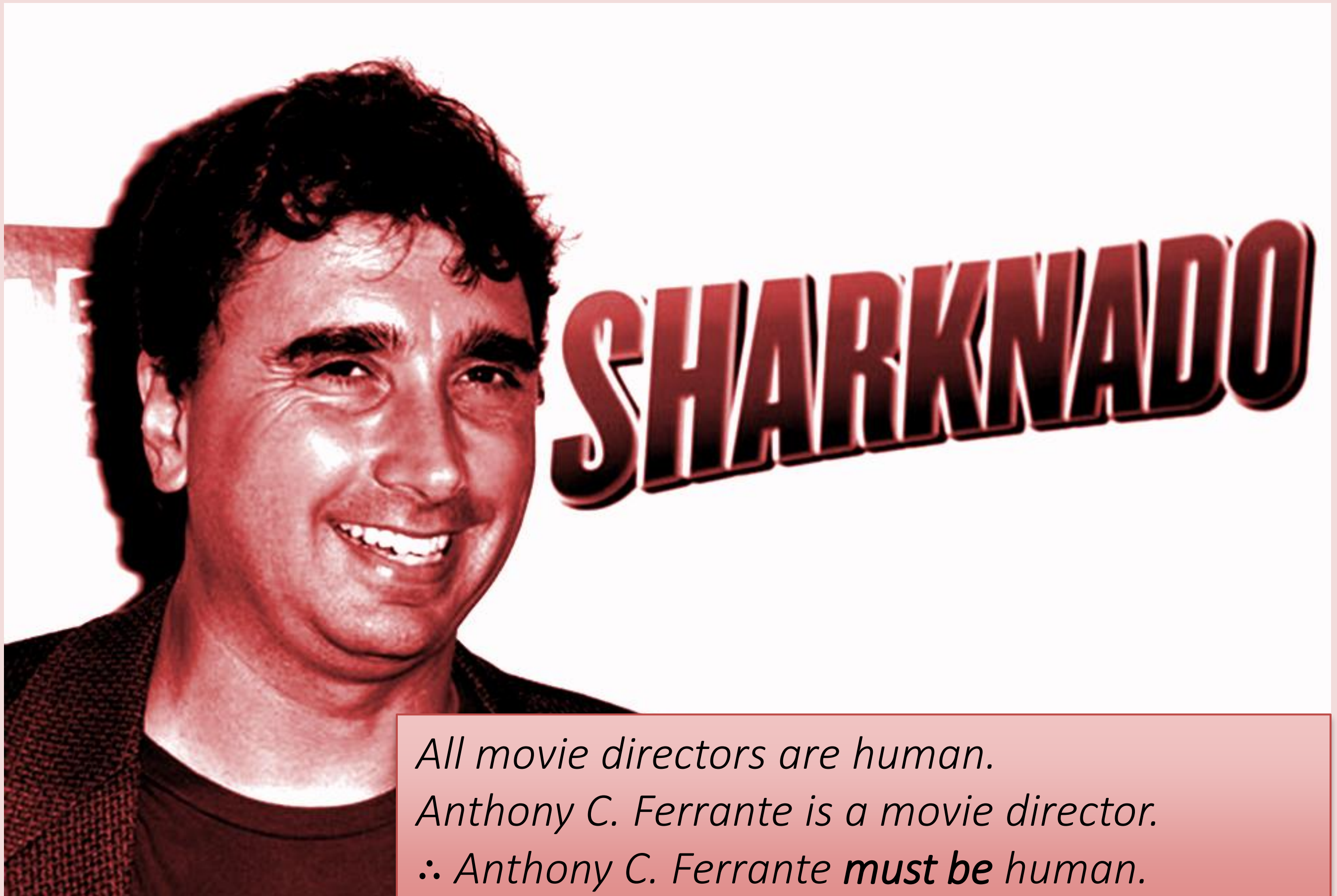
Reasoning with RDFS

What is "Reasoning"?



What general kinds of logical reasoning can we consider?

What is "Reasoning"?



All movie directors are human.
Anthony C. Ferrante is a movie director.
*∴ Anthony C. Ferrante **must be** human.*

Deductive Reasoning: Make logical conclusion from rules/premises

What is "Reasoning"?

The first three Sharknado movies were great.
*∴ Sharknado 4 will **probably** be great too.*

Inductive Reasoning: Learn approximate rule(s) from premises

What is "Reasoning"?

Abductive Reasoning: Guess a premise/explanation

Reasoning: Summary

(1) If (*x*,rdf:type,ex:SharknadoMovie) then (*x*,ex:depicts,ex:SharksInTornados)

(2) (ex:ItsAboutTime,rdf:type,ex:SharknadoMovie)

(3) (ex:ItsAboutTime,ex:depicts,ex:SharksInTornados)

Deductive Reasoning: Given (1,2), conclude (3).

Inductive Reasoning: ???

Abductive Reasoning: ???

Reasoning: Summary

(1) If `(x,rdf:type,ex:SharknadoMovie)` then `(x,ex:depicts,ex:SharksInTornados)`

(2) `(ex:ItsAboutTime,rdf:type,ex:SharknadoMovie)`

(3) `(ex:ItsAboutTime,ex:depicts,ex:SharksInTornados)`

Deductive Reasoning: Given (1,2), conclude (3).

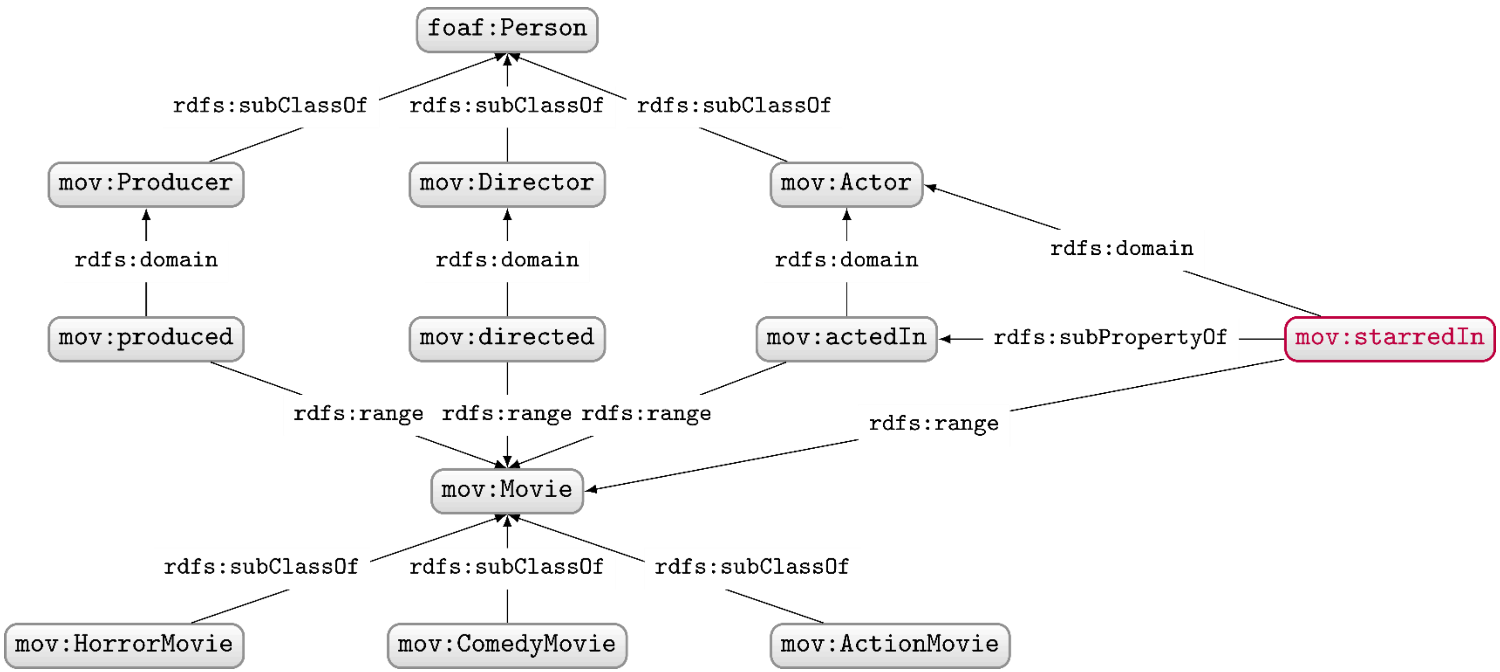
Inductive Reasoning: Given (2,3) (and similar such examples), propose (1).

Abductive Reasoning: Given (1,3), propose (2).

RDFS REASONING IS DEDUCTIVE ...

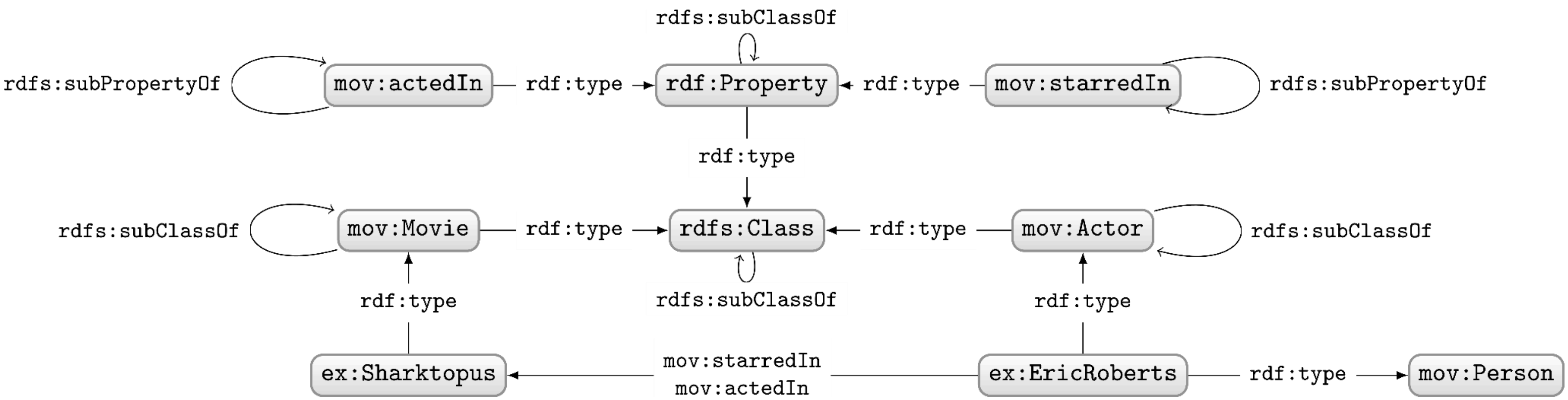
... THE ONLY FORM OF REASONING THAT IS "CERTAIN"

What conclusions can we deduce?



Given the above schema, what can we deduce from …

SOME OF THE CONCLUSIONS ...

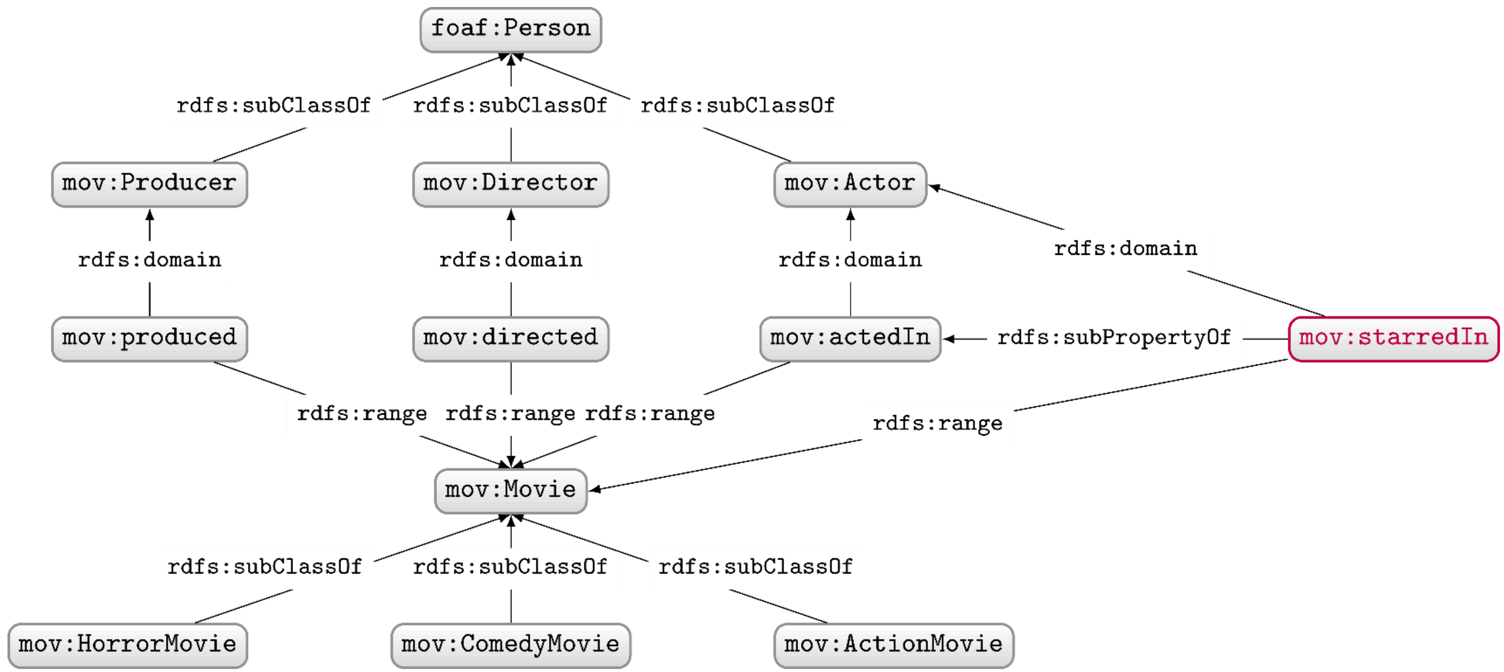


- Not shown (for the sake of my/our sanity):
  - Everything is of type `rdfs:Resource`
  - All classes are sub-class of `rdfs:Resource`
  - RDF/RDFS properties are of type `rdf:Property`

Sharktopus just one movie …

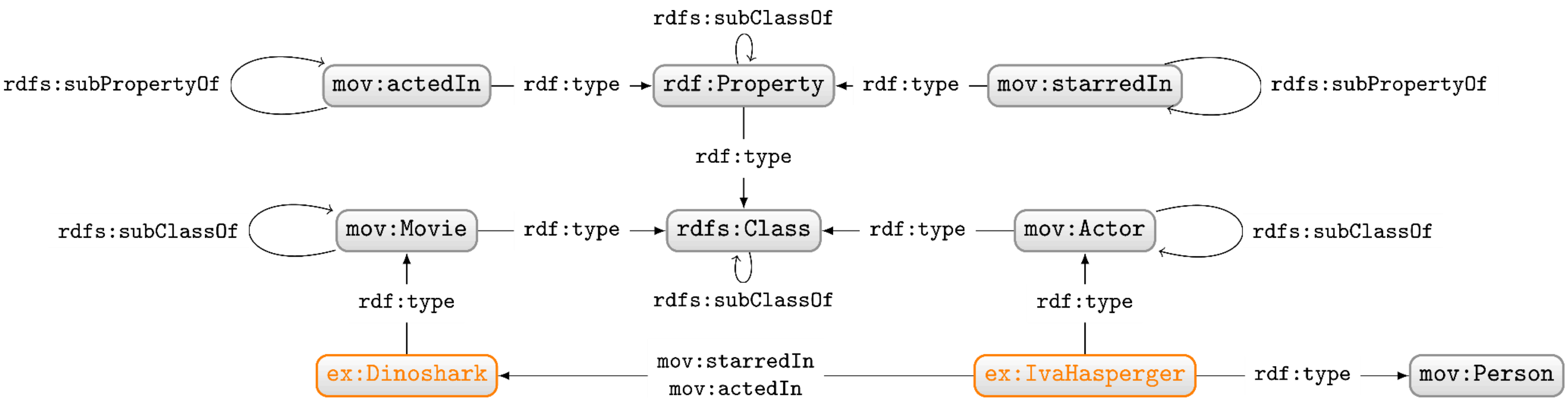
ex:EricRoberts — mov:starredIn → ex:Sharktopus

RDFS DEFINITIONS APPLY TO ANY MOVIE …



Given the above schema, what can we deduce from …

RDFS DEFINITIONS APPLY TO ANY MOVIE ...



- Not shown (for the sake of my/our sanity):
  - Everything is of type `rdfs:Resource`
  - All classes are sub-class of `rdfs:Resource`
  - RDF/RDFS properties are of type `rdf:Property`

Apply RDFS reasoning using "rules"

ID	if G matches	then G RDFS$_D$-entails
rdfD1	?x ?p ?l . (?l a literal with datatype IRI dt(?l) $\in D$)	?x ?p _:b . _:b a dt(?l) .
rdfD2	?x ?p ?y .	?p a rdf:Property .
rdfs1	?u $\in D$	?u a rdfs:Datatype .
rdfs2	?p rdfs:domain ?c . ?x ?p ?y .	?x a ?c .
rdfs3	?p rdfs:range ?c . ?x ?p ?y .	???
rdfs4a	?x ?p ?y .	?x a rdfs:Resource .
rdfs4b	?x ?p ?y .	?y a rdfs:Resource .
rdfs5	?p rdfs:subPropertyOf ?q . ?x ?p ?y .	???
rdfs6	?p a rdf:Property .	?p rdfs:subPropertyOf ?p .
rdfs7	?p rdfs:subPropertyOf ?q . ?q rdfs:subPropertyOf ?r .	?p rdfs:subPropertyOf ?r .
rdfs8	?c a rdfs:Class .	?c rdfs:subClassOf rdfs:Resource .
rdfs9	?c rdfs:subClassOf ?d . ?x a ?c .	?x a ?d .
rdfs10	?c a rdfs:Class .	?c rdfs:subClassOf ?c .
rdfs11	?c rdfs:subClassOf ?d . ?d rdfs:subClassOf ?e .	???
rdfs12	?p a rdfs:ContainerMembershipProperty .	?p rdfs:subPropertyOf rdfs:member .
rdfs13	?d a rdfs:Datatype .	?d rdfs:subClassOf rdf:Literal .

(Don't worry about rdfD1, rdfs1, rdfs12, rdfs13)

APPLY RDFS REASONING USING "RULES"

ID	if G matches	then G RDFS$_D$-entails
rdfD1	?x ?p ?l . (?l a literal with datatype IRI dt(?l) $\in D$)	?x ?p _:b . _:b a dt(?l) .
rdfD2	?x ?p ?y .	?p a rdf:Property .
rdfs1	?u $\in D$	?u a rdfs:Datatype .
rdfs2	?p rdfs:domain ?c . ?x ?p ?y .	?x a ?c .
rdfs3	?p rdfs:range ?c . ?x ?p ?y .	?y a ?c .
rdfs4a	?x ?p ?y .	?x a rdfs:Resource .
rdfs4b	?x ?p ?y .	?y a rdfs:Resource .
rdfs5	?p rdfs:subPropertyOf ?q . ?x ?p ?y .	?x ?q ?y .
rdfs6	?p a rdf:Property .	?p rdfs:subPropertyOf ?p .
rdfs7	?p rdfs:subPropertyOf ?q . ?q rdfs:subPropertyOf ?r .	?p rdfs:subPropertyOf ?r .
rdfs8	?c a rdfs:Class .	?c rdfs:subClassOf rdfs:Resource .
rdfs9	?c rdfs:subClassOf ?d . ?x a ?c .	?x a ?d .
rdfs10	?c a rdfs:Class .	?c rdfs:subClassOf ?c .
rdfs11	?c rdfs:subClassOf ?d . ?d rdfs:subClassOf ?e .	?c rdfs:subClassOf ?e .
rdfs12	?p a rdfs:ContainerMembershipProperty .	?p rdfs:subPropertyOf rdfs:member .
rdfs13	?d a rdfs:Datatype .	?d rdfs:subClassOf rdf:Literal .

(Don't worry about rdfD1, rdfs1, rdfs12, rdfs13)

Axiomatic triples: Always true in RDFS

rdf:type	rdfs:domain	rdfs:Resource	; rdfs:range	rdfs:Class	.
rdfs:domain	rdfs:domain	rdf:Property	; rdfs:range	rdfs:Class	.
rdfs:range	rdfs:domain	rdf:Property	; rdfs:range	rdfs:Class	.
rdfs:subPropertyOf	rdfs:domain	rdf:Property	; rdfs:range	rdf:Property	.
rdfs:subClassOf	rdfs:domain	rdfs:Class	; rdfs:range	rdfs:Class	.
rdf:subject	rdfs:domain	rdf:Statement	; rdfs:range	rdfs:Resource	.
rdf:predicate	rdfs:domain	rdf:Statement	; rdfs:range	rdfs:Resource	.
rdf:object	rdfs:domain	rdf:Statement	; rdfs:range	rdfs:Resource	.
rdfs:member	rdfs:domain	rdfs:Resource	; rdfs:range	rdfs:Resource	.
rdf:first	rdfs:domain	rdf:List	; rdfs:range	rdfs:Resource	.
rdf:rest	rdfs:domain	rdf:List	; rdfs:range	rdfs:List	.
rdfs:seeAlso	rdfs:domain	rdfs:Resource	; rdfs:range	rdfs:Resource	.
rdfs:isDefinedBy	rdfs:domain	rdfs:Resource	; rdfs:range	rdfs:Resource	.
rdfs:comment	rdfs:domain	rdfs:Resource	; rdfs:range	rdfs:Literal	.
rdfs:label	rdfs:domain	rdfs:Resource	; rdfs:range	rdfs:Literal	.
rdf:value	rdfs:domain	rdfs:Resource	; rdfs:range	rdfs:Resource	.
rdf:_n	rdfs:domain	rdfs:Resource	; rdfs:range	rdfs:Resource	.

rdf:Alt	rdfs:subClassOf	rdfs:Container	.
rdf:Bag	rdfs:subClassOf	rdfs:Container	.
rdf:Seq	rdfs:subClassOf	rdfs:Container	.
rdfs:ContainerMembershipProperty	rdfs:subClassOf	rdf:Property	.
rdfs:Datatype	rdfs:subClassOf	rdfs:Class	.

rdfs:isDefinedBy rdfs:subPropertyOf rdfs:seeAlso .

rdf:_n rdf:type rdfs:ContainerMembershipProperty .

(Don’t worry about greyed-out triples)

Reasoning in RDFS over RDF graph *G*

1. Add axiomatic triples to *G*
2. Apply rules exhaustively, adding conclusions to *G*, until nothing new found

Will this always finish? Or can it run forever?

So long as we do not "invent" new terms, and axiomatic triples are finite, the process will end once *G* has all possible combinations of terms as triples.

* More or less

Semantic Web: Logic

Data:

Ireland

(Ireland,partOf,Europe)
(Ireland,isA,Country)
(Ireland,capital,Dublin)

Dublin

(Ireland,capital,Dublin)
(Dublin,population,1000000)

Logic: "(b,capital,a) $\rightarrow$ (a,partOf,b)"
"(a,partOf,b), (b,partOf,c) $\rightarrow$ (a,partOf,c)"

Query: "(x,partOf,y)?"

Output: {($x \mapsto$ Ireland, $y \mapsto$ Europe),
($x \mapsto$ Dublin, $y \mapsto$ Ireland),
($x \mapsto$ Dublin, $y \mapsto$ Europe)}

RDFS (1.1): A Web Standard

http://www.w3.org/TR/rdf-schema/

W3C Recommendation

RDF Schema 1.1

W3C Recommendation 25 February 2014

This version:
http://www.w3.org/TR/2014/REC-rdf-schema-20140225/
Latest published version:
http://www.w3.org/TR/rdf-schema/
Previous version:
http://www.w3.org/TR/2014/PER-rdf-schema-20140109/
Editors:
Dan Brickley, Google
R.V. Guha, Google
Previous Editors:
Brian McBride

Please check the **errata** for any errors or issues reported since publication.

This document is also available in this non-normative format: diff w.r.t. 2004 Recommendation

Questions?