

# Inferencing

Erdoğan Doğdu

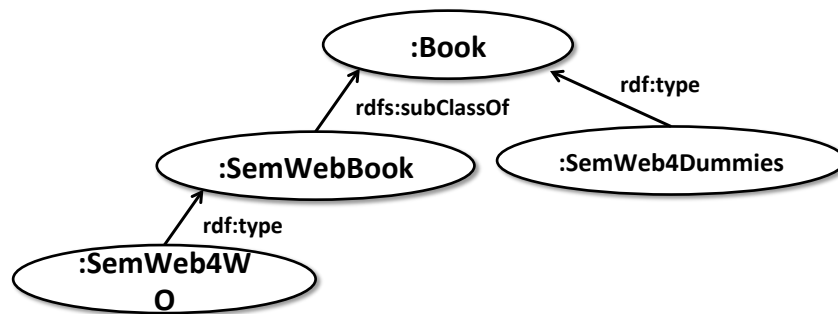
*Notes from "Semantic Web for the Working Ontologist" Book*

## Inferencing in Semantic Web

- Given some information
  - Determine other/related information
  - Discovering new relationships based on the data and based on some additional information in the form of a vocabulary, e.g. a set of rules (w3c)

2

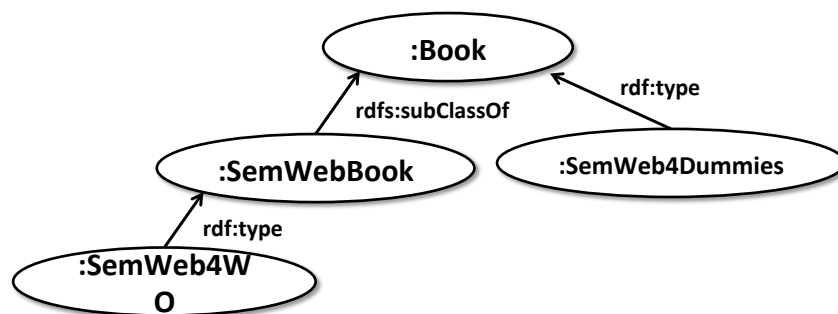
## Example



- Books?
- `SELECT ?item WHERE { ?item a :Book }`

3

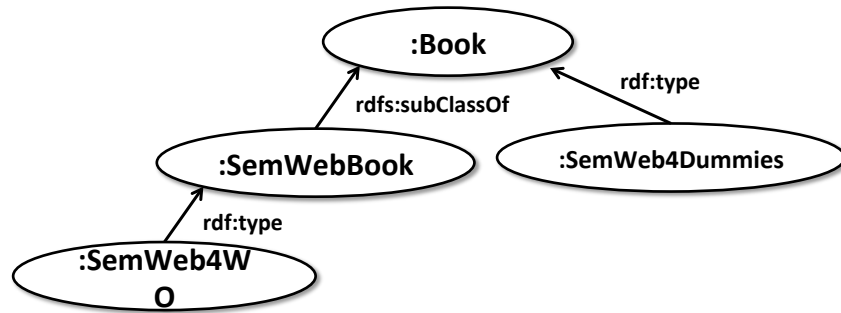
## Example



- `SELECT ?item WHERE { ?item a :Book }`  
`:SemWeb4Dummies`  
~~`:SemWeb4WO`~~

4

## Example



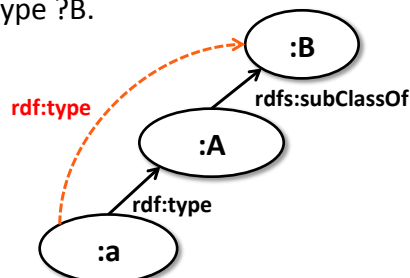
- Rule: all members of the subclass are also members of the superclass

5

## Inferencing

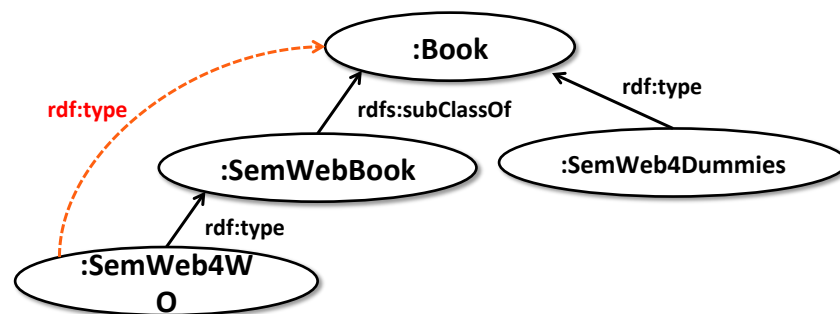
- Rule: all members of the subclass are also members of the superclass
- Meaning of rdfs:subClassOf

IF ?A rdfs:subClassOf ?B. AND ?x rdf:type ?A. *(given)*  
 THEN ?x rdf:type ?B. *(implied)*



6

## Example



- `SELECT ?item WHERE { ?item a :Book }`  
:SemWeb4Dummies  
:SemWeb4WO

7

## Inferencing

- Why?
- How?
- When?

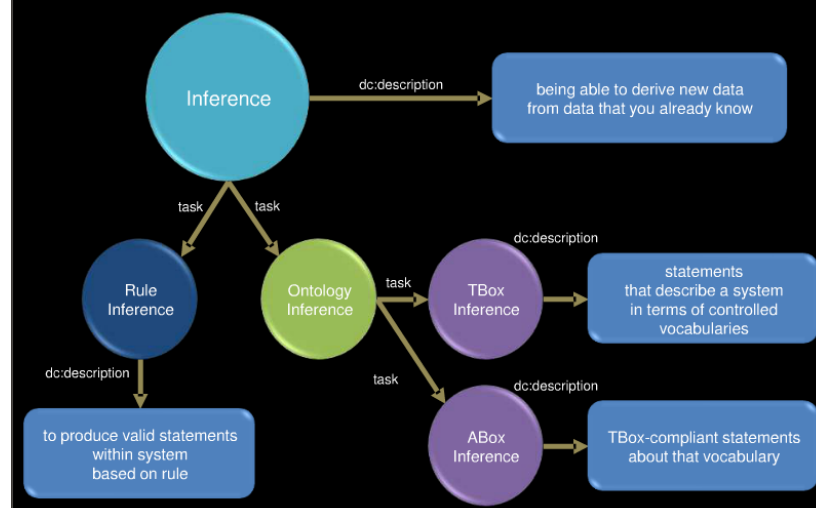
8

## Why inference?

- To get more information
- To keep less information
- To align/integrate/link to more information (on the web)
- ...

9

### Task of Inference



<http://www.slideshare.net/onlyjiny/inference-on-the-semantic-web>

10

## TBox inferencing

- Discovering new statements that describe the system in terms of controlled vocabularies

- Ex:

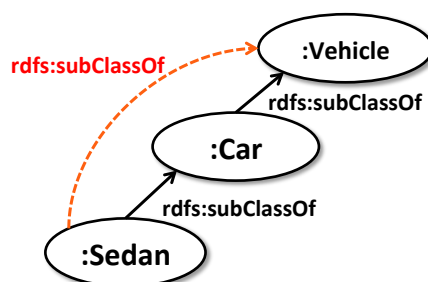
Given

:A rdfs:subClassOf :B

:B rdfs:subClassOf :C

Then

:A rdfs:subClassOf :C



11

## ABox inferencing

- Tbox-complaint statements about the vocabulary

- Ex:

Given

:Sedan rdfs:subClassOf :Car

:Car rdfs:subClassOf :Vehicle

:VWSedan rdf:type :Sedan

Then

:VWSedan rdf:type :Car

:VWSedan rdf:type :Vehicle

12

## Rule inferencing

- To produce valid statements based on rules
- Ex: defining hasWife/hasHusband relationships as a rule
  - If hasParent(?x, ?y) and hasParent(?x, ?z)  
and Man(?y) and Woman(?z)
  - Then hasWife(?y, ?z), hasHusband(?z, ?y)
- Syntax?
  - Rule languages

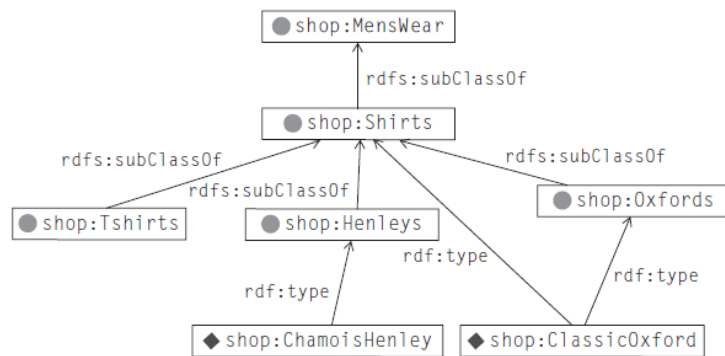
13

## Asserted vs. Inferred Triples

- Asserted triples
  - Triples (statements) in the original RDF store/model
- Inferred triples
  - Additional triples that were asserted by the inference rules
  - Inferred triple could be asserted before!
  - No logical distinction between the two

14

## Example

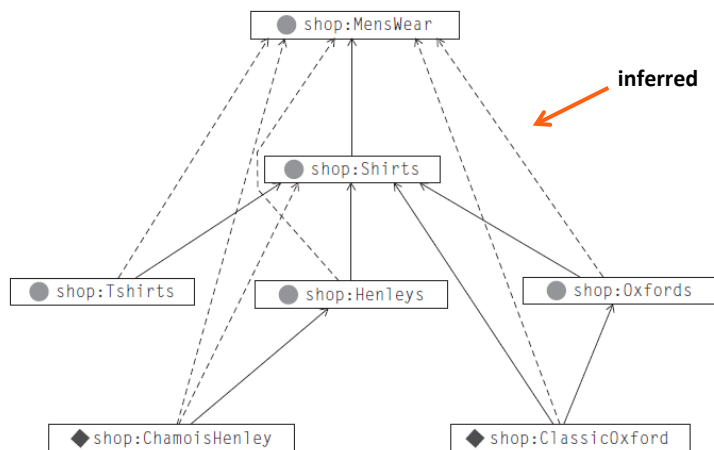


**FIGURE 6.2**

Asserted triples in the catalog model.

15

## Example



**FIGURE 6.3**

All triples in the catalog model. Inferred triples are shown as dashed lines.

16



## How?

- SPARQL
  - express rules in general and the inference rules of RDFS and OWL by **using SPARQL CONSTRUCT**
  - a CONSTRUCT query specifies new triples based on a graph pattern of triples found in the data

```
CONSTRUCT { ?r rdf:type ?B }  
WHERE { ?A rdfs:subClassOf ?B .  
         ?r rdf:type ?A }
```

17

## Inference when?

- When should the inferencing be done?
- Outside the definition of RDFS/OWL
- But important
- Differs from implementation to implementation

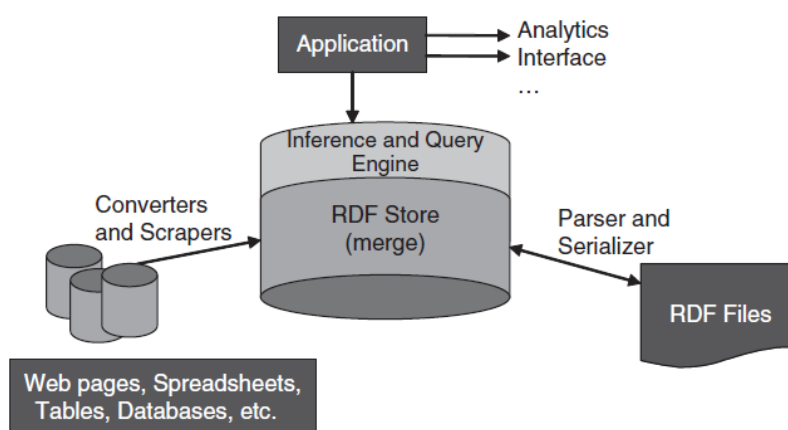
18

## Inferencing when?

- **Cached inferencing**
  - Store all statements (triples) in a single store whether asserted or inferred
  - Risk: explosion of triples in the store
  - Risk: when removing a triple, should we remove inferred triples and how?
- **Just in time inferencing**
  - Never store any inferred triples
  - Inference in response to queries only (computed at the latest possible moment)
  - Risk: duplicating inference work

19

## Inference-based SW App Architecture



20

## More on RDFS inferencing

- `rdfs:subPropertyOf` propagation

**CONSTRUCT** { ?x ?r ?y }

**WHERE** { ?x ?q ?y .

    ?q **rdfs:subPropertyOf** ?r.

}

21

## Example

TBox

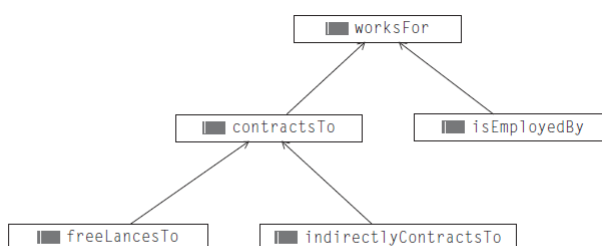


FIGURE 7.1

`rdfs:subPropertyOf` relations for workers in the firm.

ABox

```

:Goldman :isEmployedBy :TheFirm.
:Spence :freeLancesTo :TheFirm.
:Long :indirectlyContractsTo :TheFirm.
  
```

Inferred

```

:Goldman :worksFor :TheFirm.
:Spence :contractsTo :TheFirm.
:Spence :worksFor :TheFirm.
:Long contractsTo :TheFirm.
:Long :worksFor :TheFirm.
  
```

22

## domain and range

- `rdfs:domain` inference

```
CONSTRUCT {?x rdf:type ?D .}
WHERE {?P rdfs:domain ?D .
      ?x ?P ?y .}
```

- `rdfs:range` inference

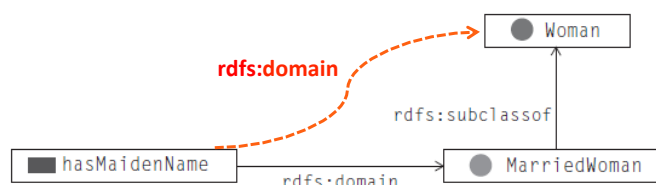
```
CONSTRUCT {?y rdf:type ?D .}
WHERE {?P rdfs:range ?D .
      ?x ?P ?y .}
```

23

## More

- `rdfs:domain` and `rdfs:subClassOf`

```
CONSTRUCT {?P rdfs:domain ?C .}
WHERE {?P rdfs:domain ?D .
      ?D rdfs:subClassOf ?C .}
```



24

## Class intersection

- Given

`:C rdfs:subClassOf :A`

`:C rdfs:subClassOf :B`

$$C \subseteq A \cap B$$

- and

`:x rdf:type :C`

- We can infer

`:x rdf:type :A`

`:x rdf:type :B`

$$A \cap B \subseteq C$$

```
:Surgeon rdfs:subClassOf :Staff.  
:Surgeon rdfs:subClassOf :Physician.  
:Kildare rdf:type :Surgeon.
```

then we can infer that

```
:Kildare rdf:type :Staff.  
:Kildare rdf:type :Physician.25
```

## Class union

- Given

`:A rdfs:subClassOf :C`

`:B rdfs:subClassOf :C.`

$$A \cup B \subseteq C$$

- and

`:x rdf:type :A`

**or**

`:x rdf:type :B`

- implies

`:x rdf:type :C`

## OWL inferencing

```

CONSTRUCT {?y ?q ?x}
WHERE {?p owl:inverseOf ?q .
        ?x ?p ?y . }

CONSTRUCT {?p owl:inverseOf ?p. }
WHERE {?p a owl:SymmetricProperty . }

CONSTRUCT {?x ?p ?z .}
WHERE {?x ?p ?y .
        ?y ?p ?x .
        ?p a owl:TransitiveProperty . }

```

27

## owl:equivalentClass

```

CONSTRUCT {?r rdf:type ?b .}
WHERE {?a owl:equivalentClass ?b .
        ?r rdf:type ?a . }

CONSTRUCT {?r rdf:type ?a .}
WHERE {?a owl:equivalentClass ?b .
        ?r rdf:type ?b . }

owl:equivalentClass rdf:type owl:SymmetricProperty.

```

28

## owl:sameAs

```

CONSTRUCT {?s ?p ?x. }
WHERE {?s ?p ?y.
      ?x owl:sameAs ?y .}
CONSTRUCT {?x ?p ?o. }
WHERE {?y ?p ?o .
      ?x owl:sameAs ?y .}
CONSTRUCT {?s ?x ?o. }
WHERE {?s ?y ?o .
      ?x owl:sameAs ?y .}
owl:sameAs rdf:type owl:SymmetricProperty.

```

29

## owl:FunctionalProperty

```

CONSTRUCT {?a owl:sameAs ?b . }
WHERE {?p rdf:type owl:FunctionalProperty .
      ?x ?p ?a .
      ?x ?p ?b . }

CONSTRUCT {?a owl:sameAs ?b . }
WHERE {?p rdf:type owl:InverseFunctionalProperty .
      ?a ?p ?x .
      ?b ?p ?x . }

```

30