COMP318: RDFS vs OWL

www.csc.liv.ac.uk/~valli/Comp318

Dr Valentina Tamma
Room: Ashton 2.12
Dept of computer science
University of Liverpool
V.Tamma@liverpool.ac.uk

based on material by M. Rodriguez Muro, P. Hitzler and S. Rudolph

Where were we

- RDF entailment rules
- RDFS entailment rules

Brief recap of RDF/RDFS

- RDF/RDFS describe subject predicate object triples and basic subsumption relations.
  - Very efficient deduction/querying
    - SPARQL based on simple entailment, but...
  - Very poor expressivity...
    - Describe data in terms of triples (RDF)
    - Describe the model behind the data (RDFS) by:
      - Declare the “types” and properties/relationships of the things we want to make assertions about
      - Allowing to infer new assertions implicitly stated from a set of given facts
  - But sometimes we need to express more advanced notions, e. g.:
    - A person has only one birth date
    - No person can be male and female at the same time

What can you represent with RDFS

- ***RDFS provides:***
  - Classes
    - `Lecturer rdf:type rdfs:Class`
  - Class hierarchies
    - `Lecturer rdfs:subClassOf AcademicStaff`
  - Properties
    - `teachesModule rdf:type rdf:Property`
  - Property hierarchies
    - `teachesModule rdfs:subPropertyOf coordinatesModule`
  - Domain and range declarations
    - `teachesModule rdfs:domain Lecturer`
    - `teachesModule rdfs:range Module`
      - They infer information rather than checking data

What can't you represent in RDFS

- ***RDFS does NOT provide:***
  - Disjointness of Classes
    - *`Male` and `Female` are disjoint*
      - *they cannot have any shared instaces*
  - Property characteristics (inverse, transitive, ...)
    - `Lecturer teachesModule Module` *and*
      `Module isTaughtByLecturer Lecturer`
      are not explicitly related
  - Local scope of properties
    - `Lecturer` has a property `hasTitle` whose values (range) is restricted to all values of the class `PhD`
      - *only people who hold a PhD can be lecturers*
      - but other `AcademicStaffMembers` can hold a BSc
  - Complex concept definitions (Boolean combination of classes)
    - `Person = Man ∪ Woman`
    - `Mother = Woman ∩ Parent`

What can't you represent in RDFS

- ***RDFS does NOT provide:***
  - Cardinality restrictions
    - `Person` may have at most 1 name
    - `Lecturer` *teaches exactly two modules*
  - A way to distinguish between classes and instances:
    - `Feline rdf:type rdfs:Class`
    - `Cat rdf:type Feline`
    - `felix rdf:type Cat`
      - `:felix` is an instance of an instance (`Cat`)
  - A way to distinguish between language constructors and ontology vocabulary:
    - `rdf:type rdfs:range rdfs:Class`
    - `rdfs:Property rdfs:type rdfs:Class`
    - `rdf:type rdfs:subPropertyOf rdfs:subClassOf`
  - Reasoning for these non-standard semantics

What can you infer in RDFS

Schema

```
ex:Lecturer
rdfs:subClassOf
ex: AcademicStaff
```

Assertion

```
staffUniv:john_smith
rdf:type
ex: Lecturer
```

RDFS Entailment ⇒

Inferred Fact

```
staffUniv:john_smith
rdf:type
ex: AcademicStaff
```

What can't you infer in RDFS

- However, not all types of inferences are possible in RDFS

Schema

```
:wife_of rdfs:subPropertyOf married_to.
:married_to rdfs:domain :Spouse;
            rdfs:range  :Spouse.
:wife_of rdfs:domain :Wife;
         rdfs:range  :Husband.
```

Assertion

```
:juliet :wife_of :romeo.
```

Facts Inferred

```
:juliet rdf:type   :Wife;
        rdf:type   :Spouse;
        married_to :romeo;
:romeo rdf:type :Spouse;
       rdf:type :Husband.
```

What can't you infer in RDFS

- What about if we want to model symmetry, *i.e.* `:x :married_to :y` implies `:y :married_to :x`?

Schema

```
:wife_of rdfs:subPropertyOf married_to.
:married_to rdfs:domain :Spouse;
            rdfs:range  :Spouse.
:wife_of rdfs:domain :Wife;
         rdfs:range  :Husband.
:husband_of rdfs:domain :Husband;
            rdfs:range  :Wife.
```

Assertion

```
:juliet :wife_of :romeo.
```

Facts Inferred

```
:juliet rdf:type   :Wife;
        rdf:type   :Spouse;
        married_to :romeo;
:romeo rdf:type :Spouse;
       rdf:type :Husband.
```

Facts ***NOT Inferred***

```
:romeo :married to :juliet.
:romeo :husband_of :juliet.
```

Too much representational freedom is not good!

- We might want to be able to detect what might seem inconsistent facts, but RDFS is not able to constrain models through consistency and axioms:

Schema

```
:wife_of rdfs:subPropertyOf married_to.
:married_to rdfs:domain :Spouse;
            rdfs:range  :Spouse.
:wife_of rdfs:domain :Wife;
         rdfs:range  :Husband.
:husband_of rdfs:domain :Husband;
            rdfs:range  :Wife.
```

Assertions

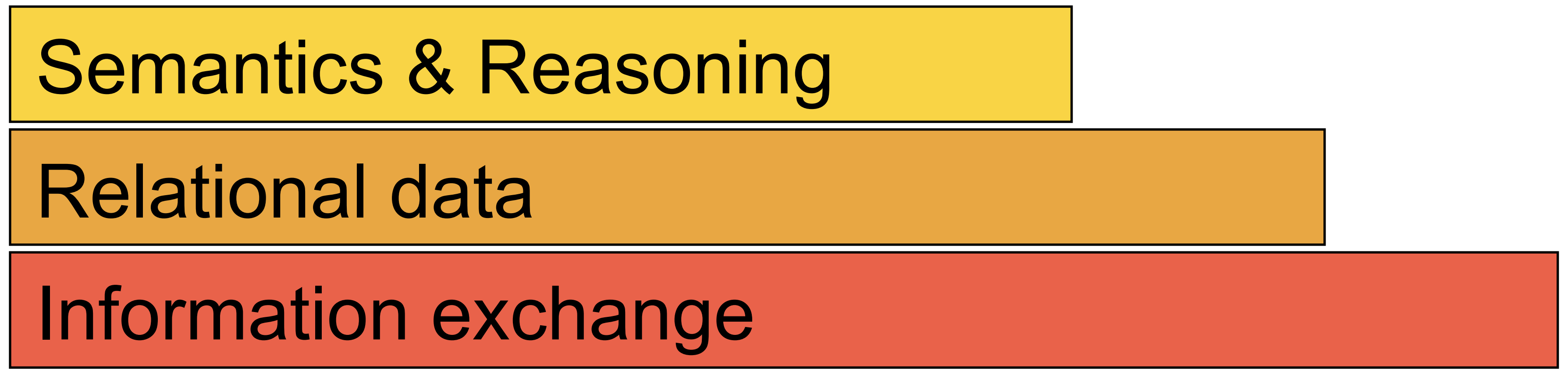
```
:romeo rdf:type :Husband.
:romeo :wife_of : juliet.
```

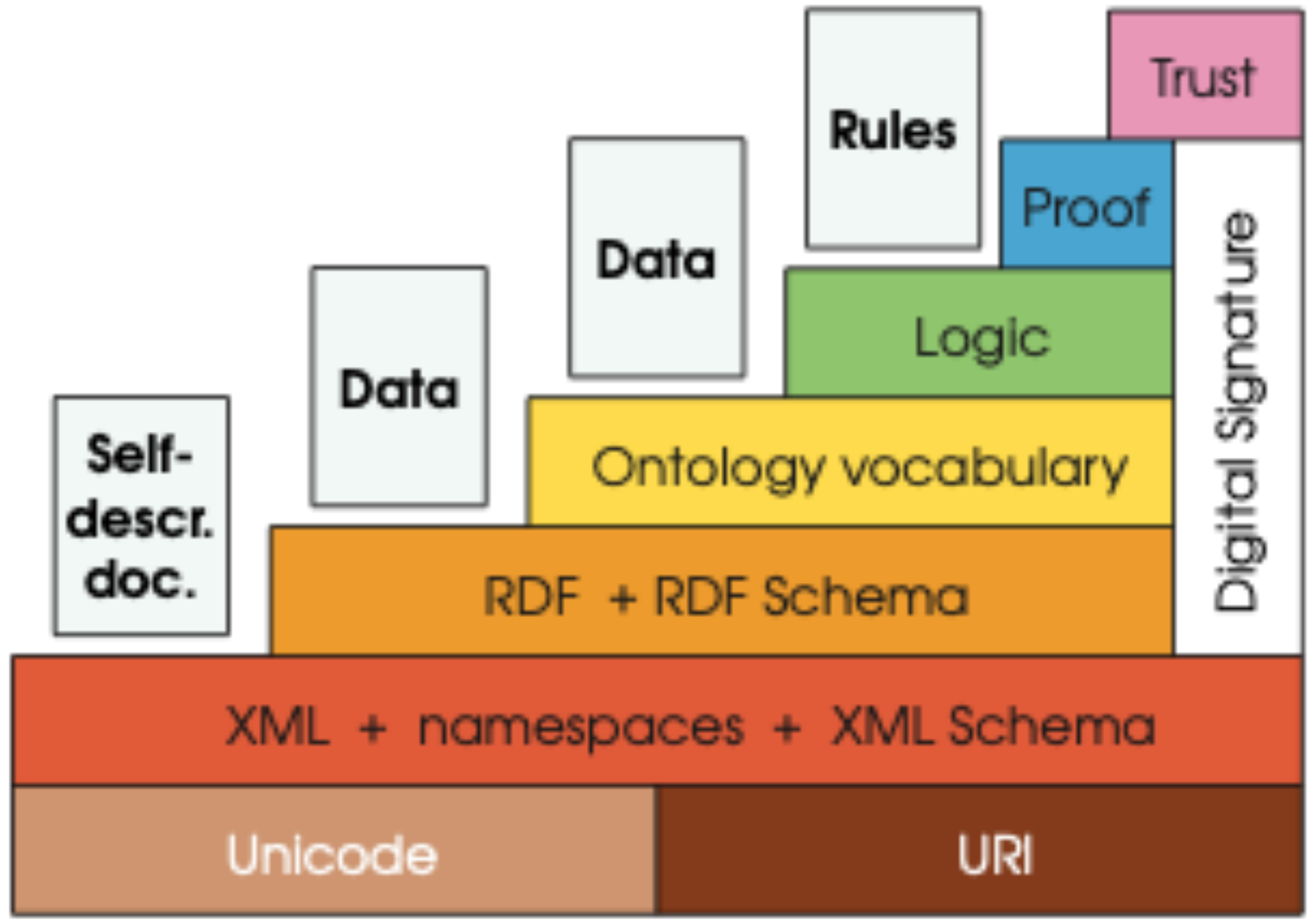
Facts ***INCORRECTLY Inferred***

```
:romeo rdf:type :Wife.
```

There is no contradiction, and the mis-modelling is not diagnosed automatically!

Layering of SW languages





T. Berners-Lee

Semantic + Web = Semantic Web

Represent Web content in a form that is more easily machine-processable:

describe meta-data about resources on the Web

i.e. descriptions about the data being represented, the model and constraints used to represent them.

Use intelligent techniques to take advantage of these representations:

process meta-data in a way that is similar to human reasoning and inference

thus information gathering can be done by a machine in a similar way to how humans currently gather information on the web..

Conflicting aims

- Ontologies provide the structured vocabulary for describing meta-data
  - Languages to represent ontologies need to be as expressive as possible whilst permitting automated deduction:
    - To describe meta-data, we want a (logic-based) language that is as expressive as possible.
    - To simulate human deduction in an efficient way, we want a logic that permits efficient automated deduction.
    - The logic of choice is a compromise between expressiveness and complexity of deduction.

Wish list for ontology languages....

- Compromise between expressivity and scalability
  - Well defined syntax
    - necessary for automatic machine-processing of information;
  - Formal semantics
    - describe the meaning of knowledge precisely, it allows computers to reason precisely about knowledge
      - class membership: if $x \in A$ and $A \subseteq B$ then we can infer that $x \in B$
      - consistency: $x \in A$, $A \subseteq B \cap C$, $A \subseteq D$ and $B \cap D = \varnothing$ then we have an inconsistency!
      - classification: if some property-value pairs are a sufficient condition for membership in A, and x satisfies them, then infer $x \in A$
  - Efficient reasoning
    - derivations can be computed mechanically
      - consistency, classification, detection of unintended relationships between classes;
  - Sufficient expressive power
    - Compatibility with RDF and RDFS;

Requirements for Ontology Languages

- The main requirements are:
  - a well-defined syntax
  - a formal semantics
  - convenience of expression
  - efficient (complex) reasoning support
  - sufficient expressive power

RDFS

OWL

Extending RDFS

- OWL extends RDF Schema to a knowledge representation language for the Web
  - Logical expressions (and, or, not)
    - Woman = Human and Female
    - Person = Man or Woman
    - Man = not (Woman)
  - (in)equality
    - john differentFrom mary
      - There is no assumption that individuals have only one name, thus the need to explicitly state inequality
  - local properties
    - Lecturer only teaches modules

Extending RDFS

- OWL extends RDF Schema to a knowledge representation language for the Web
  - required/optional properties
    - teaches Modules is a required property for Lecturer
  - required values
    - BritishCitizen hasNationality = “British”
      - the value for the hasNationality property for the class BritishCitizen can only be “British”
  - enumerated classes
    - DaysOfTheWeek = {Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday}
  - symmetry, inverse

Recap

- Limitation of RDFS
- Introduction to OWL

Reasoning support

- Formal semantics allows the automatic deduction of new facts and possible conflicts between class definitions (consistency):

An ontology language can be provided with formal semantics and reasoning support by mapping it to a known logical formalism and by using the reasoning tools developed for the chosen formalism.

  - These checks are extremely valuable for designing large ontologies, for collaborative ontology design and for sharing and integrating ontologies from various sources:
    - check the consistency of the ontology;
    - check for unintended relations between classes;
    - check for the unintended classification of instances.

OWL 1 and OWL 2

- OWL: OWL 1 (`http://www.w3.org/TR/owl-features/`) and OWL 2 (`http://www.w3.org/TR/owl2-overview/`)
  - Rationale for OWL
    - Open world assumption:
      - The absence of a particular statement means that the statement has not been made explicitly yet.
        - Whether the statement is true or not, and whether it is believed that it is (or would be) true or not is irrelevant.
      - Thus, from the absence of a statement alone, a deductive reasoner cannot infer that the statement is false.
    - Reasonable trade-off between expressivity and scalability
    - Fully declarative semantics.
- OWL 2 DL
  - Fragment of first order predicate logic, decidable
  - Known complexity classes
  - Reasonably efficient for real ontologies + instances

OWL 1: Three species of OWL

- OWL Lite:
  - Sublanguage of OWL DL but without nominals and XML datatypes:
  - Classification hierarchy
  - Simple constraints
  - It excludes enumerated classes, disjointness statements, and arbitrary cardinality.
  - Reasoning still not tractable.
- OWL DL
  - Sublanguage of OWL Full, it imposes restrictions on the use of OWL/RDFS constructors
  - Application of OWL's constructors to each other not permitted
  - Provides reasonably efficient reasoning support.

OWL 1: Three species of OWL

- OWL Full
  - Very high expressiveness, uses all of the OWL primitives
  - Fully upward compatible with RDF
  - Losing decidability: no complete or efficient reasoning support
  - All syntactic freedom of RDF (self-modifying):
    - primitives can be combined in arbitrary ways with RDF(S)

OWL 2 - Profile

- Sublanguages of OWL2 trading expressive power for efficient reasoning
  - Each supports different application scenarios
- OWL 2 EL
  - very large ontologies, efficient reasoning performance guaranteed at the expenses of expressive power;
- OWL 2 RL
  - subclass axioms understood as rule like implication, with head - superclass and body - subclass
  - different restrictions on subclasses and superclasses
  - allows the integration of OWL with rules

OWL 2 - Profile

- Sublanguages of OWL2 trading expressive power for efficient reasoning
  - Each supports different application scenarios
- OWL 2 QL
  - useful to query data rich applications
  - different restrictions on subclasses and superclasses
  - suitable for simple, lightweight ontologies with a large number of individuals and it is necessary to access the data directly via SQL queries
  - fast implementation on top of legacy DB systems, relational or RDF