UNIVERSITÄT MANNHEIM

Semantic Web Technologies
Web Ontology Language (OWL)
Part II

Heiko Paulheim

Previously on “Semantic Web Technologies”

- We have got to know
  - OWL, a more powerful ontology language than RDFS
  - Simple ontologies and some reasoning
  - Sudoku solving

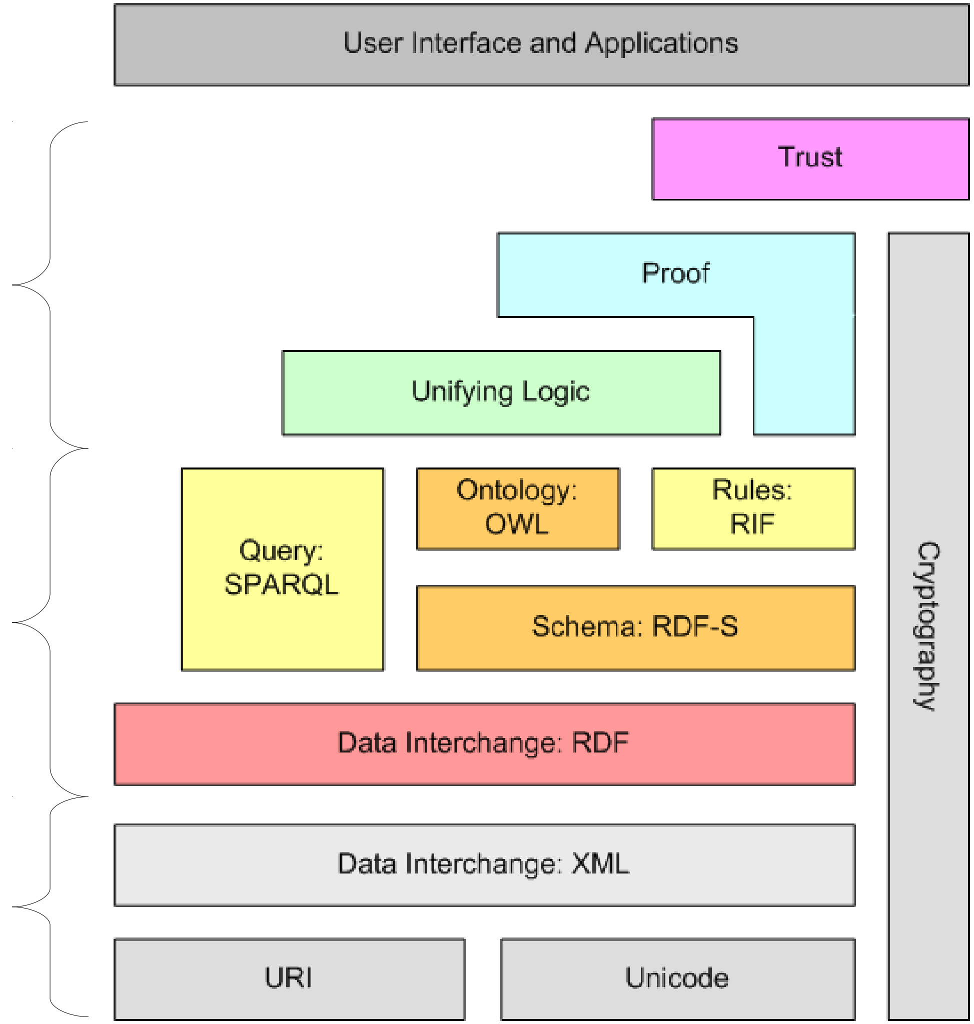
- Today
  - New constructs in OWL2
  - Russell's paradox
  - Reasoning in OWL
  - Complexity of ontologies
  - A peek at rule languages for the Semantic Web

Semantic Web – Architecture

here be dragons...

Semantic Web Technologies (This lecture)

Technical Foundations



Berners-Lee (2009): *Semantic Web and Linked Data*
http://www.w3.org/2009/Talks/0120-campus-party-tbl/

OWL2 – New Constructs and More

- Five years after the first OWL standard
- OWL2: 2009
  - Syntactic sugar
  - New language constructs
  - OWL profiles

- We have already encountered some, e.g.,
  - Qualified relations
  - Reflexive, irreflexive, and antisymmetric properties

OWL2: Syntactic Sugar

- Disjoint classes and disjoint unions
  - OWL 1:

```
:Wine owl:equivalentClass [
  a owl:Class ;
  owl:unionOf (:RedWine :RoséWine :WhiteWine) ] .
```

```
:RedWine  owl:disjointWith :RoséWine, :WhiteWine .
:RoséWine owl:disjointWith :WhiteWine .
```

  - OWL 2:

```
:Wine owl:disjointUnionOf
  (:RedWine :RoséWine :WhiteWine ).
```

  - Also possible:

```
_:x a owl:AllDisjointClasses ;
    owl:members (:RedWine :RoséWine WhiteWine ).
```

OWL2: Syntactic Sugar

- Negative(Object|Data)PropertyAssertation
- Allow negated statements
- e.g.: Paul is not Peter's father

```
_x [ a owl:NegativeObjectPropertyAssertion;
       owl:sourceIndividual :Paul ;
       owl:targetIndividual :Peter ;
       owl:assertionProperty :fatherOf ] .
```

- If that's syntactic sugar, it must also be possible differently
  - But how?

OWL2: Syntactic Sugar

- Negative(Object|Data)PropertyAssertion
- Replaces less intuitive set constructs
- Paul is not Peter's father

```
Paul a [ owl:complementOf [
               a owl:Restriction ;
               owl:onProperty :fatherOf ;
               owl:hasValue :Peter
          ]
        ].
```

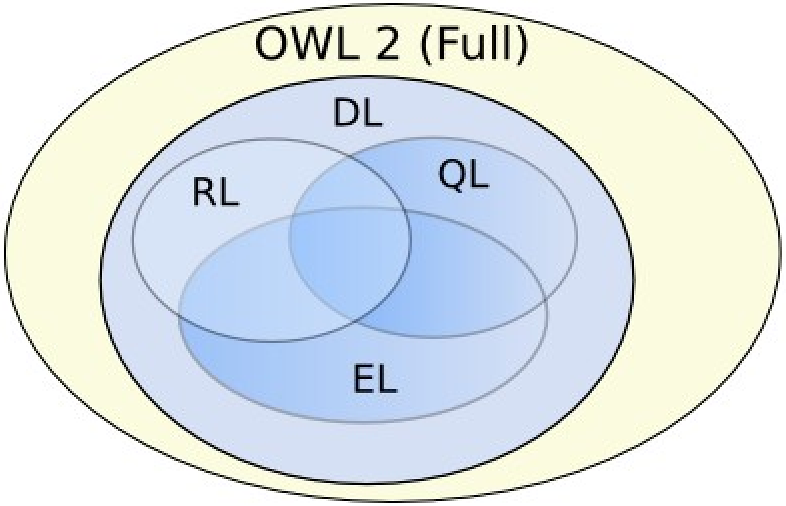
OWL2: Reflexive Class Restrictions

- Using `hasSelf`
- Example: defining the set of all autodidacts:

```
:AutoDidact owl:equivalentClass [
  a owl:Restriction ;
  owl:onProperty :teaches ;
  owl:hasSelf "true"^^xsd:boolean ] .
```

OWL2: Profiles

- Profiles are subsets of OWL2 DL
  - EL, RL und QL
  - Similar to complexity classes
- Different runtime and memory complexity
- Depending on requirements



OWL2 Profile

- OWL2 EL (Expressive Language)
  - Fast reasoning on many standard ontologies
  - Restrictions, e.g.:
    - someValuesFrom, but not allValuesFrom
    - No inverse and symmetric properties
    - No unionOf and complementOf

- OWL2 QL (Query Language)
  - Fast query answering on relational databases
  - Restrictions, e.g.:
    - No unionOf, allValuesFrom, hasSelf, …
    - No cardinalities and functional properties

OWL2 Profile

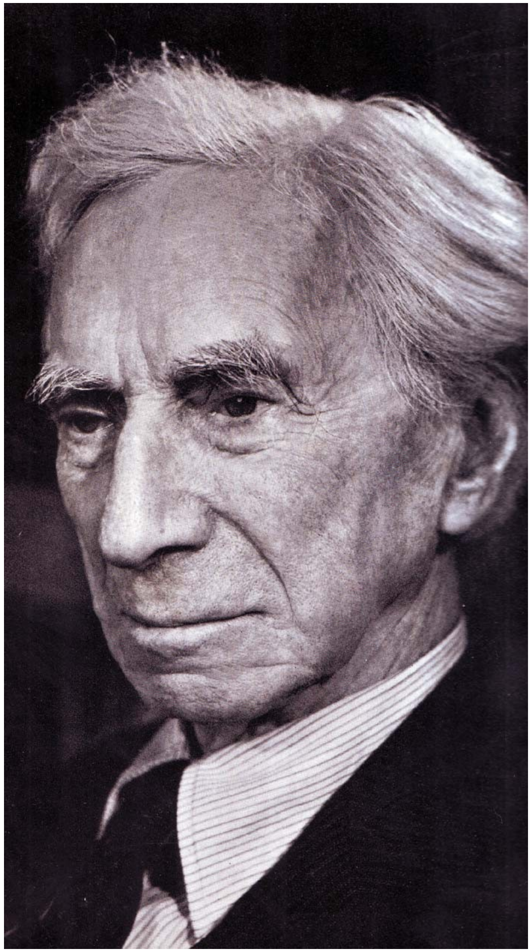
- OWL2 RL (Rule Language)
  - Subset similar to rule languages such as datalog
    - subClassOf is translated to a rule (Person ← Student)
  - Restrictions, e.g.:
    - Only qualified restrictions with 0 or 1
    - Some restrictions for head and body

- The following holds for all three profiles:
  - Reasoning can be implemented in polynomial time for each of the three
  - Reasoning on the union of two profiles only possible in exponential time

OWL2 Example: Russell's Paradox

- A classic paradox by Bertrand Russell, 1918
- In a city, there is exactly one barber who shaves everybody who does not shave themselves.

  Who shaves the barber?



OWL2 Example: Russell's Paradox

- Class definitions

```
:People owl:disjointUnionOf
(:PeopleWhoShaveThemselves
 :PeopleWhoDoNotShaveThemselves ) .
```

- Relation definitions:

```
:shavedBy rdfs:domain :People .
:shavedBy rdfs:range :People .
:shaves owl:inverseOf :shavedBy .
```

- Every person is shaved by exactly one person:

```
:People rdfs:subClassOf [
 a owl:Restriction ;
 owl:onProperty :shavedBy ;
 owl:cardinality "1"^^xsd:integer ] .
```

OWL2 Example: Russell's Paradox

- Then, we define the barber:

```
:Barbers rdfs:subClassOf :People ;
         owl:equivalentClass [
             rdf:type owl:Class ;
             owl:oneOf ( :theBarber )
         ] .
```

OWL2 Example: Russell's Paradox

- Definition of people shaving themselves:

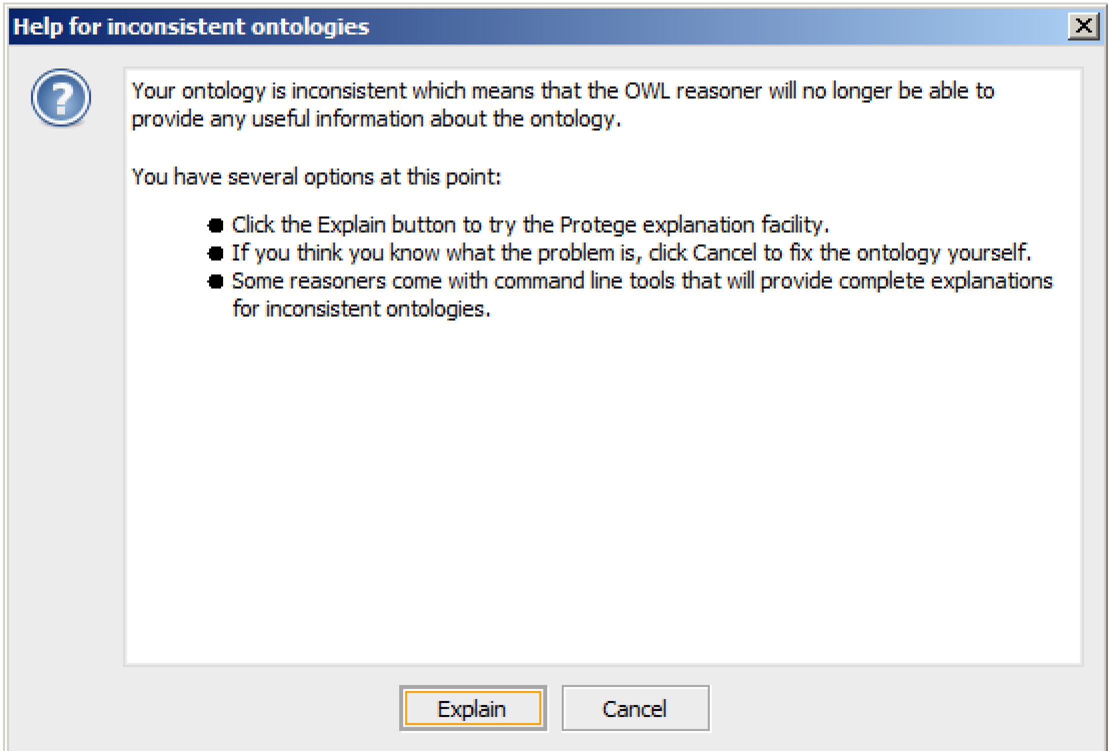
```
:PeopleWhoShaveThemselves owl:equivalentClass [
  rdf:type owl:Class ;
  owl:intersectionOf
  ( :People
    [
      a owl:Restriction ;
      owl:onProperty :shavedBy ;
      owl:hasSelf "true"^^xsd:boolean
    ]
   )
] .
```

OWL2 Example: Russell's Paradox

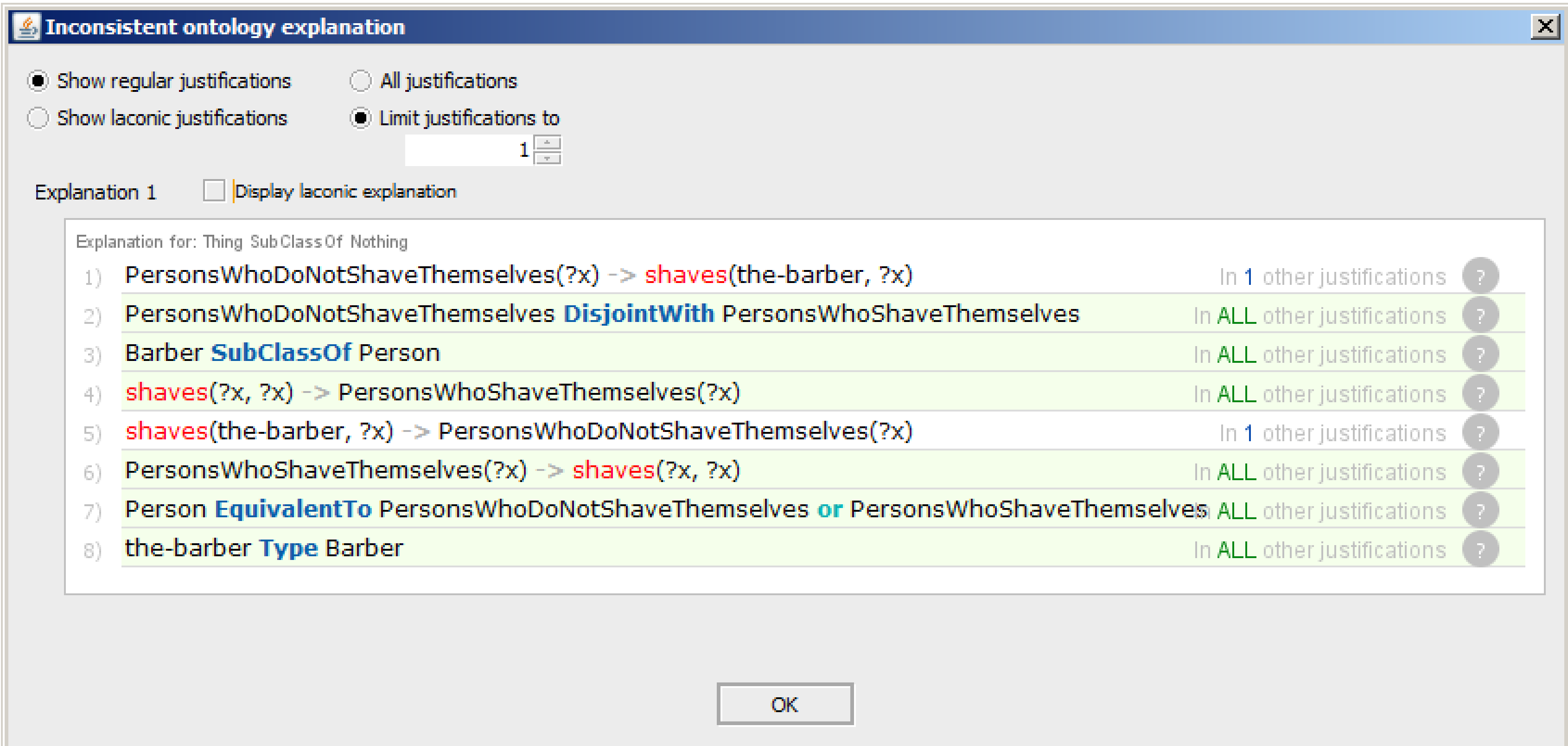
- Definition of people who do not shave themselves:

```
:PeopleWhoDoNotShaveThemselves owl:equivalentClass [
   a owl:Class ;
   owl:intersectionOf (
    :People
    [ a owl:Restriction
      owl:onProperty :shavedBy ;
      owl:allValuesFrom :Barbers
    ]
   )
] .
```

OWL2 Example: Russell's Paradox



OWL2 Example: Russell's Paradox



Reasoning in OWL DL

- We have seen reasoning for RDFS
  - Forward chaining algorithm
  - Derive axioms from other axioms

- Reasoning for OWL DL is more difficult
  - Forward chaining may have scalability issues
  - Conjunction (e.g., unionOf) is not supported by forward chaining
  - Different approach: Tableau Reasoning
  - Underlying idea: find contradictions in ontology
    - i.e., both a statement and its opposite can be derived from the ontology

Typical Reasoning Tasks

- What do we want to know from a reasoner?
  - Subclass relations
    - e.g., Are all birds flying animals?
  - Equivalent classes
    - e.g., Are all birds flying animals *and vice versa*?
  - Disjoint classes
    - e.g., Are there animals that are mammals and birds at the same time?
  - Class consistency
    - e.g., Can there be mammals that lay eggs?
  - Class instantiation
    - e.g., Is Flipper a dolphin?
  - Class enumeration
    - e.g., List all dolphins

Example: A Simple Contradiction

- Given:

```
:Human a owl:Class .
:Animal a owl:Class .
:Human owl:disjointWith :Animal .

:Jimmy a :Animal .
:Jimmy a :Human .
```

Example: A Simple Contradiction

- We can derive:
  - $:Human \cap :Animal = \emptyset$

```
owl:Nothing owl:intersectionOf (:Human :Animal) .
```

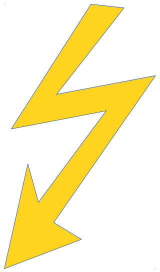
  - $:Jimmy \in (:Human \cap :Animal)$

```
:Jimmy a [ a owl:Class; owl:intersectionOf
                                (:Human :Animal)] .
```

- i.e.:
  - $:Jimmy \in \emptyset$

```
:Jimmy a owl:Nothing .
```

  - That means: the instance must not exist
  - but it does



Reasoning Tasks Revisited

- Subclass Relations

  Student $\subseteq$ Person $\Leftrightarrow$ „Every student is a person“

- Proof method: Reductio ad absurdum
  - "Invent" an instance i
  - Define Student(i) and $\neg$Person(i)
  - Check for contradictions
    - If there is one: Student $\subseteq$ Person has to hold
    - If there is none: Student $\subseteq$ Person cannot be derived
      - Note: it may still hold!

Example: Subclass Relations

- Ontology:

```
:Student owl:subClassOf :UniversityMember .
:UniversityMember owl:subClassOf :Person .
```

- Invented instance:

```
:i a :Student .
```

```
:i a [ owl:complementOf :Person ] .
```

- We have

```
:i a :Student .
:Student owl:subClassOf :UniversityMember .
```

  Thus

```
:i a :UniversityMember .
```

- And from

```
:UniversityMember owl:subClassOf :Person .
```

- We further derive that

```
:i a Person .
```

Example: Subclass Relations

- Now, we have

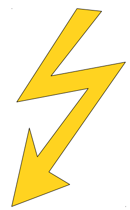
```
:i a :Person .
:i a [ owl:complementOf :Person ] .
```

i.e.,

```
:i a [ owl:intersectionOf (:Person
                           [ owl:complementOf :Person
])] .
```

- from which we derive

```
:i a owl:Nothing .
```



Reasoning Tasks Revisited

- Class equivalence
  - Person $\equiv$ Human
- Split into
  - Person $\subseteq$ Human and
  - Human $\subseteq$ Person
- i.e., show subclass relation twice
  - We have seen that

- Class disjointness
  - Are C and D disjoint?
  - "Invent" an instance i
  - Define C(i) and D(i)
    - We have done set (the Jimmy example)

Class Consistency

- Can a class have instances?
  - e.g., married bachelors

```
:Bachelor owl:subClassOf :Man .
:Bachelor owl:subClassOf
  [ a owl:Restriction;
    owl:onProperty :marriedTo;
    owl:cardinality 0 ] .
:MarriedPerson owl:subClassOf [
    a owl:Restriction;
    owl:onProperty :marriedTo;
    owl:cardinality 1 ] .

:MarriedBachelor owl:intersectionOf
  (:Bachelor :MarriedPerson) .
```

- Now: invent an instance of the class
  - And check for contradictions

Reasoning Tasks Revisited

- Class Instantiation
  - Is Flipper a dolphin?
- Check:
  - define ¬Dolphin(Flipper)
  - Check for contradiction
- Class enumeration
  - Repeat class instantiation for all known instances

Typical Reasoning Tasks Revisited

- What do we want to know from a reasoner?
  - Subclass relations
    - e.g., Are all birds flying animals?
  - Equivalent classes
    - e.g., Are all birds flying animals *and vice versa*?
  - Disjoint classes
    - e.g., Are there animals that are mammals and birds at the same time?
  - Class consistency
    - e.g., Can there be mammals that lay eggs?
  - Class instantiation
    - e.g., Is Flipper a dolphin?
  - Class enumeration
    - e.g., List all dolphins

Typical Reasoning Tasks Revisited

- We have seen
  - All reasoning tasks can be reduced to the same basic tasks
  - i.e., consistency checking
- This means: for building a reasoner that can solve those tasks,
  - We only need a reasoner capable of consistency checking

Ontologies in Description Logics Notation

- Classes and Instances
  - $C(x)$ ↔ `x a C .`
  - $R(x,y)$ ↔ `x R y .`
  - $C \sqsubseteq D$ ↔ `C rdfs:subClassOf D`
  - $C \equiv D$ ↔ `C owl:equivalentClass D`
  - $C \sqsubseteq \neg D$ ↔ `C owl:disjointWith D`
  - $C \equiv \neg D$ ↔ `C owl:complementOf D`
  - $C \equiv D \sqcap E$ ↔ `C owl:intersectionOf (D E) .`
  - $C \equiv D \sqcup E$ ↔ `C owl:unionOf (D E) .`
  - $\top$ ↔ `owl:Thing`
  - $\bot$ ↔ `owl:Nothing`

Ontologies in Description Logics Notation

- Domains, ranges, and restrictions
  - $\exists R.\top \sqsubseteq C$ ↔ `R rdfs:domain C .`
  - $\forall R.C$ ↔ `R rdfs:range C .`
  - $C \sqsubseteq \forall R.D$ ↔
    ```
    C owl:subClassOf
      [ a owl:Restriction;
        owl:onProperty R;
        owl:allValuesFrom D ] .
    ```
  - $C \sqsubseteq \exists R.D$ ↔
    ```
    C owl:subClassOf
      [ a owl:Restriction;
        owl:onProperty R;
        owl:someValuesFrom D ] .
    ```
  - $C \sqsubseteq \geq nR$ ↔
    ```
    C owl:subClassOf
      [ a owl:Restriction;
        owl:onProperty R;
        owl:minCardinality n ] .
    ```

Global Statements in Description Logic

- So far, we have seen mostly statements about single classes
  - e.g., $C \sqsubseteq D$

- In Description Logics, we can also make global statements
  - e.g., $D \sqcup E$
  - This means: every single instance is a member of D or E (or both)
- Those global statements are heavily used in the reasoning process

Negation Normal Form (NNF)

- Transforming ontologies to Negation Normal Form:
  - ⊑ und ≡ are not used
  - Negation only for atomic classes and axioms

- A simplified notation of ontologies
- Used by tableau reasoners

Negation Normal Form (NNF)

- Eliminating ⊑:
  - Replace C ⊑ D by ¬C ⊔ D
  - Note: this is a shorthand notation for ∀x: ¬C(x) ∨ D(x)

- Why does this hold?
  - C ⊑ D is equivalent to C(x) → D(x)

C(x)	D(x)	C(x) → D(x)	¬C(x) ∨ D(x)
true	true	true	true
true	false	false	false
false	true	true	true
false	false	true	true

Negation Normal Form (NNF)

- Eliminating ≡:
    - Replace $C \equiv D$ by $C \sqsubseteq D$ and $D \sqsubseteq C$
    - Proceed as before
- i.e.: $C \equiv D$ becomes

  $C \sqsubseteq D$

  $D \sqsubseteq C$

  - and thus

    $\neg C \sqcup D$

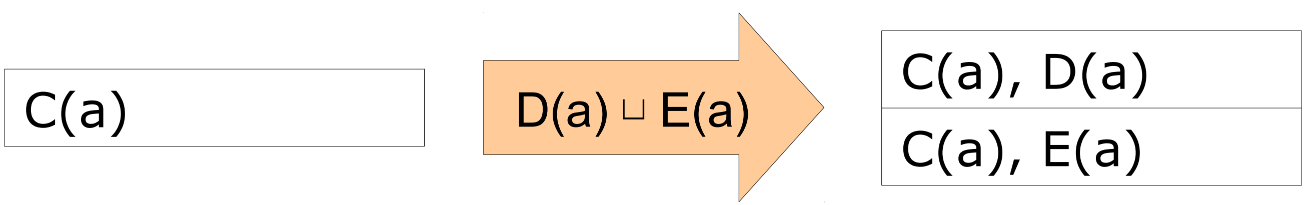
    $\neg D \sqcup C$

Negation Normal Form (NNF)

- Further transformation rules
  - NNF(C) = C (for atomic C)
  - NNF(¬C) = ¬C (for atomic C)
  - NNF(¬ ¬C) = C
  - NNF(C ⊔ D) = NNF(C) ⊔ NNF(D)
  - NNF(C ⊓ D) = NNF(C) ⊓ NNF(D)
  - NNF(¬(C ⊓ D)) = NNF(¬C) ⊔ NNF(¬D)
  - NNF(¬(C ⊔ D)) = NNF(¬C) ⊓ NNF(¬D)
  - NNF(∀R.C) = ∀R.NNF(C)
  - NNF(∃R.C) = ∃R.NNF(C)
  - NNF(¬∀R.C) = ∃R.NNF(¬C)
  - NNF(¬∃R.C) = ∀R.NNF(¬C)

The Basic Tableau Algorithm

- Tableau: Collection of derived axioms
  - Is subsequently extended
  - As for forward chaining
- In case of conjunction
  - Split the tableau



When is an Ontology Free of Contradictions?

- Tableau is continuously extended and split
- Free of contradictions if...
  - No further axioms can be created
  - At least one partial tableau is free of contradictions
  - A partial tableau has a contradiction if it contains both an axiom and its negation
    - e.g.. Person(Peter) und ¬Person(Peter)
    - The partial tableau is then called *closed*

The Basic Tableau Algorithm

- Given: an ontology O in NNF

  While not all partial tableaus are closed

      * Choose a non-closed partial tableau T and an A $\in$ O $\cup$ T
        If A is not contained in T
            If A is an atomic statement
                add A to T
                back to *
            If A is a non-atomic statement
                Choose an individual i $\in$ O $\cup$ T
                Add A(i) to T
                back to *
        else
            Extend the tableau with consequences from A
            back to *

The Basic Tableau Algorithm

- Extending a tableau with consequences

Nr	Axiom	Action
1	C(a)	Add C(a)
2	R(a,b)	Add R(a,b)
3	C	Choose an individual a, add C(a)
4	(C ⊓ D)(a)	Add C(a) and D(a)
5	(C ⊔ D)(a)	Split tableau into T1 and T2. Add C(a) to T1, D(a) to T2
6	(∃R.C)(a)	Add R(a,b) and C(b) for a *new* Individual b
7	(∀R.C)(a)	For all b with R(a,b) ϵ T: add C(b)

A Simple Example

- Given the following ontology:

  :Animal owl:unionOf (:Mammal :Bird :Fish :Insect :Reptile) .
  :Animal owl:disjointWith :Human .
  :Seth a :Human .
  :Seth a :Insect .

- Is this knowledge base consistent?

A Simple Example

- Given the following ontology:

  :Animal owl:unionOf (:Mammal :Bird :Fish :Insect :Reptile) .
  :Animal owl:disjointWith :Human .
  :Seth a :Human .
  :Seth a :Insect .

  - The same ontology in DL-NNF:

    $\neg$Animal $\sqcup$ $\neg$Human
    Animal $\sqcup$ ($\neg$Mammal $\sqcap$ $\neg$Bird $\sqcap$ $\neg$Fish $\sqcap$ $\neg$Insect $\sqcap$ $\neg$Reptile)
    $\neg$Animal $\sqcup$ (Mammal $\sqcup$ Bird $\sqcup$ Fish $\sqcup$ Insect $\sqcup$ Reptile)
    Human(Seth)
    Insect(Seth)

- Let's try how reasoning works now!

A Simple Example

Human(Seth), Insect(Seth)

Nr	Axiom	Action
1	C(a)	Add C(a)

A Simple Example

Human(Seth), Insect(Seth),
(¬Animal ⊔ ¬Human)(Seth)

Nr	Axiom	Action
3	C	Choose an individual a, add C(a)

A Simple Example

Human(Seth), Insect(Seth), ¬Animal(Seth)
Human(Seth), Insect(Seth), ¬Human(Seth)

Nr	Axiom	Action
5	(C ⊔ D)(a)	Split the tableau into T1 and T2. Add C(a) to T1, D(a) to T2

A Simple Example

Human(Seth), Insect(Seth), $\neg$Animal(Seth) Animal $\sqcup$ ($\neg$Mammal $\sqcap$ $\neg$Bird $\sqcap$ $\neg$Fish $\sqcap$ $\neg$Insect)(Seth)
Human(Seth), Insect(Seth), $\neg$Human(Seth)

Nr	Axiom	Action
3	C	Choose an individual a, add C(a)

A Simple Example

Human(Seth), Insect(Seth), ¬Animal(Seth) Animal(Seth)
Human(Seth), Insect(Seth), ¬Animal(Seth) (¬Mammal ⊓ ¬Bird ⊓ ¬Fish ⊓ ¬Insect ⊓ ¬Reptile)(Seth)
Human(Seth), Insect(Seth), ¬Human(Seth)

Nr	Axiom	Action
5	(C ⊔ D)(a)	Split the tableau into T1 and T2. Add C(a) to T1, D(a) to T2

A Simple Example

Human(Seth), Insect(Seth), ¬Animal(Seth) Animal(Seth)
Human(Seth), Insect(Seth), ¬Animal(Seth) (¬Mammal ⊓ ¬Bird ⊓ ¬Fish ⊓ ¬Insect ⊓ ¬Reptile)(Seth) ¬Mammal(Seth) ¬Bird(Seth) ¬Fish(Seth) ¬Insect(Seth) ¬Reptile(Seth)
Human(Seth), Insect(Seth), ¬Human(Seth)

Nr	Axiom	Action
4	(C ⊓ D)(a)	Add C(a) and D(a)

Another Example

- Again, a simple ontology:

```
:Woman rdfs:subClassOf :Person .
:Man rdfs:subClassOf :Person .
:hasChild rdfs:domain :Person .
:hasChild rdfs:range :Person .
:Peter :hasChild :Julia .
:Julia a :Woman .
:Peter a :Man .
```

Another Example

- in DL NNF:

  $\neg$Man $\sqcup$ Person
  $\neg$Woman $\sqcup$ Person
  $\neg\exists$hasChild.T $\sqcup$ Person
  $\forall$hasChild.Person
  hasChild(Peter,Julia)
  Woman(Julia)
  Man(Peter)

Another Example

hasChild(Peter,Julia)

Nr	Axiom	Action
2	R(a,b)	Add R(a,b)

Another Example

hasChild(Peter,Julia), Woman(Julia)

Nr	Axiom	Action
1	C(a)	Add C(a)

Another Example

hasChild(Peter,Julia), Woman(Julia),
$(\neg\exists hasChild.\top \sqcup Person)(Peter)$

Nr	Axiom	Action
3	C	Choose an individual a, add C(a)

Another Example

hasChild(Peter,Julia), Woman(Julia), (¬∃hasChild.T ⊔ Person)(Peter), ¬∃hasChild.T(Peter)
hasChild(Peter,Julia), Woman(Julia), (¬∃hasChild.T)(Peter), Person(Peter)

Nr	Axiom	Action
5	(C ⊔ D)(a)	Split the tableau into T1 and T2. Add C(a) to T1, D(a) to T2

Another Example

hasChild(Peter,Julia), Woman(Julia), (¬∃hasChild.T ⊔ Person)(Peter), ¬∃hasChild.T(Peter)
hasChild(Peter,Julia), Woman(Julia), (¬∃hasChild.T)(Peter), Person(Peter), ¬hasChild(Peter,b0),T(b0)

Nr	Axiom	Action
6	(∃R.C)(a)	Add R(a,b) und C(b) for a *new* Individual b

Another Example

hasChild(Peter,Julia), Woman(Julia), (¬∃hasChild.T ⊔ Person)(Peter), ¬∃hasChild.T(Peter)
hasChild(Peter,Julia), Woman(Julia), (¬∃hasChild.T)(Peter), Person(Peter), ¬hasChild(Peter,b0),T(b0), ¬hasChild(Peter,b1),T(b1), ...

Nr	Axiom	Action
6	(∃R.C)(a)	Add R(a,b) und C(b) for a *new* Individual b

Introducing Rule Blocking

- Observation
  - The tableau algorithm does not necessarily terminate
  - We can add arbitrarily many new axioms

Nr	Axiom	Action
6	(∃R.C)(a)	Add R(a,b) und C(b) for a *new* Individual b

- Idea: avoid rule 6 if no new information is created
  - i.e., if we already created one instance b_a for instance a, then block using rule 6 for a.

Tableau Algorithm with Rule Blocking

- Given: an ontology O in NNF

  While not all partial tableaus are closed
  and further axioms can be created

  * Choose a non-closed partial tableau T and a non-blocked A ϵ O ∪T
    If A is not contained in T
      If A is an atomic statement
        add A to T
        back to *
      If A is a non-atomic statement
        Choose an individual i ϵ O ∪T
        Add A(i) to T
        back to *
    else
      Extend the tableau with consequences from A
      If rule 6 was used, block A for T
      back to *

Tableau Algorithm: Wrap Up

- An algorithm for description logic based ontologies
  - works for OWL Lite and DL
- We have seen examples for some OWL expressions
  - Other OWL DL expressions can be “translated” to DL as well
  - And they come with their own expansion rules
  - Reasoning may become more difficult
    - e.g., dynamic blocking and unblocking

Optimizing Tableau Reasoners

- Given: an ontology O in NNF

  While not all partial tableaus are closed
  and further axioms can be created

  * Choose a non-closed partial tableau T and a non-blocked A ϵ O $\cup$T
  If A is not contained in T
  If A is an atomic statement
  add A to T
  back to *
  If A is a non-atomic statement
  Choose an individual i ϵ O $\cup$T
  Add A(i) to T
  back to *
  else
  Extend the tableau with consequences from A
  If rule 6 was used, block A for T
  back to *

OWL Lite vs DL Revisited

- Recap: OWL Lite has some restrictions
  - Those are meant to allow for faster reasoning
- Restrictions only with cardinalities 0 and 1
  - Higher cardinalities make blocking more complex
- unionOf, disjointWith, complementOf, closed classes, ...
  - they all introduce more disjunctions
  - i.e., more splitting operations

Complexity of Ontologies

- Reasoning is usually expensive
- Reasoning performance depends on ontology complexity
  - Rule of thumb: the more complexity, the more costly
- Most useful ontologies are in OWL DL
  - But there are differences
  - In detail: complexity classes

Simple Ontologies: ALC

- ALC: Attribute Language with Complement

- Allowed:
  - subClassOf, equivalentClass
  - unionOf, complementOf, disjointWith
  - Restrictions: allValuesFrom, someValuesFrom
  - domain, range
  - Definition of individuals

SHIQ, SHOIN & co

- Complexity classes are noted as letter sequences
- Using
  - S = ALC plus transitive properties (basis for most ontologies)
  - H = Property hierarchies (subPropertyOf)
  - O = closed classes (oneOf)
  - I = inverse properties (inversePropertyOf)
  - N = numeric restrictions (min/maxCardinality)
  - F = functional properties
  - Q = qualified numerical restrictions (OWL2)
  - (D) = Usage of datatype properties

Some Tableau Reasoners

- Fact
  - University of Manchester, free
  - SHIQ
- Fact++/JFact
  - Extension of Fact, free
  - SHOIQ(and a little D), OWL-DL + OWL2
- Pellet
  - Clark & Parsia, free for academic use
  - SHOIN(D), OWL-DL + OWL2
- RacerPro
  - Racer Systems, commercial
  - SHIQ(D)

Sudoku Revisited

- Recap: we used a closed class
  - Plus some disjointness
- Resulting complexity: SO
- Which reasoners do support that?
  - Fact: SHIQ :-(
  - RacerPro: SHIQ(D) :-(
  - Pellet: SHOIN(D) :-)
  - HermiT: SHOIQ :-)

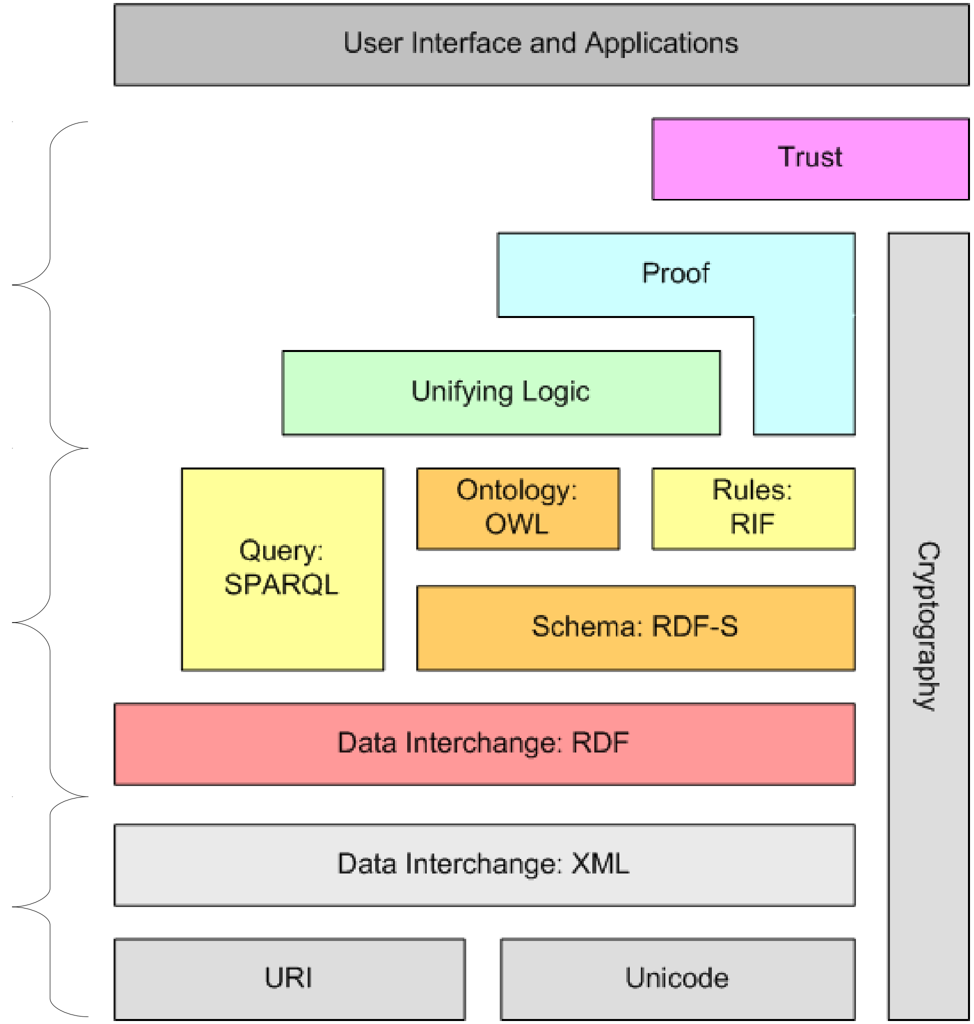
5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Rules: Beyond OWL

here be dragons...

Semantic Web Technologies (This lecture)

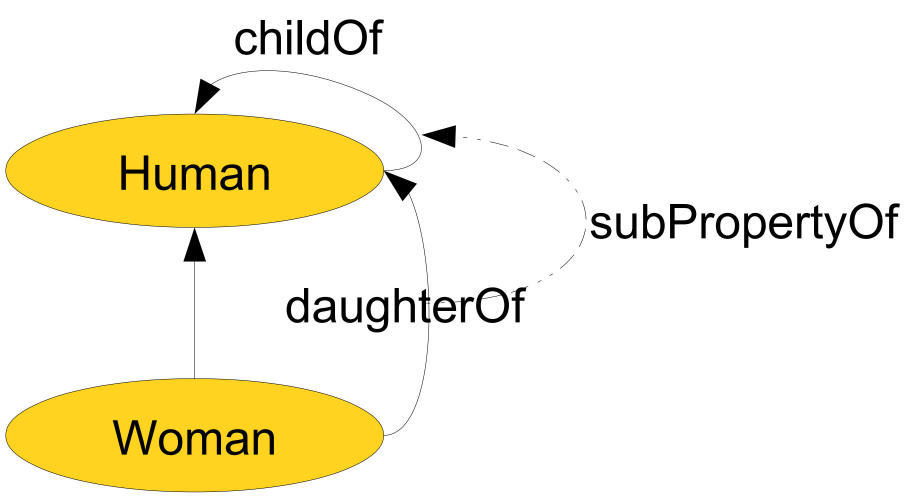
Technical Foundations



Berners-Lee (2009): *Semantic Web and Linked Data*
http://www.w3.org/2009/Talks/0120-campus-party-tbl/

Limitations of OWL

- Some things are hard or impossible to express in OWL
- Example:
  - If A is a woman and the child of B
    then A is the daughter of B



Limitations of OWL

- Let's try this in OWL:

```
:Woman rdfs:subClassOf :Human .
:childOf     a owl:ObjectProperty ;
             rdfs:domain :Human ;
             rdfs:range :Human .
:daughterOf a owl:ObjectProperty ;
             rdfs:subPropertyOf :childOf ;
             rdfs:domain :Woman .
```

Limitations of OWL

- What can a reasoner conclude with this ontology?
- Example:

```
:Julia :daughterOf :Peter .
→ :Julia a :Woman .
```

- What we would like to have instead:

```
:Julia :childOf :Peter .
:Julia a :Woman .
→ :Julia :daughterOf :Peter .
```

Limitations of OWL

- What we would like to have:

  daughterOf(X,Y) $\leftarrow$ childOf(X,Y) $\wedge$Woman(X) .

- Rules are flexible
- There are rules in the Semantic Web, e.g.
  - Semantic Web Rule Language (SWRL)
  - Rule Interchange Format (RIF)
  - See lecture in two weeks
- Some reasoners do (partly) support rules

Wrap Up

- OWL comes in many flavours
  - OWL Lite, OWL DL, OWL Full
  - Detailed complexity classes of OWL DL
  - Additions and profiles from OWL2
  - However, there are still some things that cannot be expressed...
- Reasoning is typically done using the Tableau algorithm

Questions?