MANCHESTER 1824

The University of Manchester

A Heuristic Approach to Explain the Inconsistency in OWL Ontologies

Hai Wang, Matthew Horridge, Alan Rector,

Nick Drummond, Julian Seidenberg

8/17/2005

MANCHESTER 1824

The University of Manchester

Introduction

- **OWL IS COMING!!**
- Debugging OWL is very difficult even for experts.
  - Inferences can be indirect and non-local.
  - Multiple expressions for the same notion.
  - Inconsistence propagates.
  - The internal representation is very different from User's Ontology for modern tableaux reasoners .
  - The more powerful the reasoner, the more likely it is to make non-obvious inferences
- A Heuristic Approach to debugging OWL (DL)

What is OWL?

- The latest standard in ontology languages.
- W3C recommendation.
- Based on RDF and DAML+OIL
- Has formal mathematical foundations in Description Logics.
- It allows us to use a reasoner to check the ontology.
- Three Components of an OWL Ontology: Classes, Properties and Individuals.

OWL Classes

- ► OWL is an ontology language that is primarily designed to describe and define classes. Classes are therefore the basic building blocks of an OWL ontology.
- ► Six main ways of describing classes

  The simplest of these is a **Named Class**. The other

  types are: **Intersection classes**, **Union classes**, **Complement classes**, **Restrictions**, **Enumerated classes**.

OWL Classes examples

- Restrictions
  - Restrictions describe a class of individuals based on the type and possibly number of relationships that they participate in.
  - For example:
    - Existential Restrictions
      - An existential restriction describes the class of individuals that have at least one kind of relationship along a specified property to an individual that is a member of a specified class.
      - restriction(hasFatContent someValuesFrom FatContent)
    - Universal Restrictions
      - A *Universal* restriction describes the class of individuals that for a given property, all the individuals must be members of a specified class.
      - restriction(hasTopping allValuesFrom Vegetable)

Properties

- There are two main categories of properties: **Object** properties and **datatype** properties.
- **Object** properties link individuals to individuals.
- **Datatype** properties link individuals to datatype values (e.g. integers, floats, strings).
- Properties can have as specified domain and range.
- Properties can have certain characteristics, i.e., **Functional**, **Inverse functional**, **Symmetric**, **Transitive**.

OWL Web Ontology Language Reference

http://www.w3.org/TR/owl-ref/

Unsatisfiable OWL Classes

► An OWL class is deemed to be Unsatisfiable if, because of its description, it cannot possibly have any instances.

```
DisjointClasses(Meat,Vegetable)
Class(MeatyVegetable partial Meat,
          Vegetable)
```

A Heuristic Approach to Ontology Debugging

- The heuristics are based on courses about OWL that are presented at The University of Manchester.
  - The common made mistakes have been identified.
  - The DL-reasoner has been treated as a “black box”.
- It is a uncompleted solution, but can handle the majority cases.

The Debugging Process

Debuggger

Check if OWL class is inconsistent

Identify the *unsatisfiable core*

Generate the *debugging super conditions*

Determine the *most general conflict*

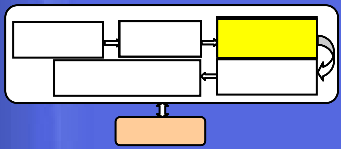
Analyse the *most general conflict* and generate explanation

OWL-DL Reasoner

The Determination of the *Unsatisfiable Core*

- Three kinds of axioms define an OWL named class -- the *basic debugging necessary conditions (BDNC)*
  - Subclass axioms (rdfs:subClassOf)
  - Equivalent class axioms (owl:equivalentClass)
  - Disjoint axioms (owl:disjointWith)
- The *unsatisfiable core* is the smallest *unsatisfiable* subset of BDNC.
  1. $UC(C) \subseteq BDNC(C)$
  2. Intersection of UC(C) is unsatisfiable.
  3. For every set of class descriptions CD:

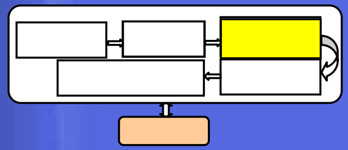
     $CD \subset UC(C) \Rightarrow$ Intersection of CD is satisfiable $\vee$ $CD=\emptyset$



The Generation of the *Debugging Super Conditions*

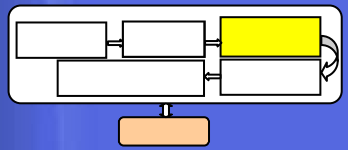
- The *unsatisfiable core merely identify the local axioms resulting in the inconsistency*.
- Actual cuase of the inconsistency may be defined somewhere else.

```
Class(PizzaTopping)
Class(PizzaBase)
DisjointClasses(PizzaTopping, PizzaBase)
ObjectProperty(hasFatContent domain(PizzaTopping))
Class(DeepPanBase partial PizzaBase
   restriction(hasFatContent someValuesFrom FatContent)
   …..))
```



The Generation of the *Debugging Super Conditions*

- The debugging process ‘collects' distributed axioms.
- Maps them into local axioms i.e. sets of necessary conditions.
- The ultimate set of ‘local' conditions is referred to as the *debugging super conditions*.



Debugging Super Condition Generation Rules (Examples)

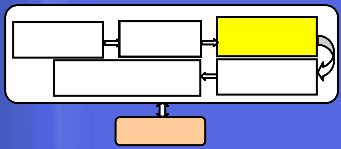
- Domain Rule
  - **IF** $\exists S.C_1 \in DSC(C)$ or

    $\geqslant n\ S \in DSC(C)$ or

    $= n\ S \in DSC(C)$,

    where $n>1$, and $DOM(S)= C_2$

    THEN $C_2 \in DSC(C)$
- For the **DeepPanBase** example, the class PizzaTopping is added to set of debugging super condition.



Debugging Super Conditions Generating Rules (Details in paper)

Named class rule:

(1) if $C_1 \in DSC(C) \wedge C_1 \sqsubseteq C_2$, where C_1 is a named OWL class
then $C_2 \in DSC(C)$

(2) if $C_1 \in DSC(C)$ and $Disj(C_1, C_2)$, where C_1 and C_2 are named OWL classes then $\neg C_2 \in DSC(C)$

Complement class rule:

(1) if $\neg C_1 \in DSC(C)$, where C_1 is a named OWL class
then if $C_2 \sqsubseteq C_1$, then $\neg C_2 \in DSC(C)$
if $C_1 \equiv C_2$, then $\neg C_2 \in DSC(C)$

(2) if $\neg C_1 \in DSC(C)$, where C_1 is an anonymous OWL class
then $NORM(C_1) \in DSC(C)$

Domain/Range rule:

(1) if $\exists S.C_1 \in DSC(C) \vee \geq n S \in DSC(C) \vee = n S \in DSC(C)$,
where $n > 0$, and $DOM(S) = C_2$
then $C_2 \in DSC(C)$

(2) if $\exists S.C_1 \in DSC(C) \vee \geq n S \in DSC(C) \vee = n S \in DSC(C)$,
and where $n > 0$, $INV(S) = S_1$ and $RAN(S_1) = C_2$
then $C_2 \in DSC(C)$

(3) if $\exists S.C_1 \in DSC(C) \vee \geq n S \in DSC(C) \vee = n S \in DSC(C)$,
where $n > 0$, and $RAN(S) = C_2$
then $\forall S.C_2 \in DSC(C)$

Functional / Inverse functional property

(1) if $\exists S.C_1 \in DSC(C)$ or $\geq n S \in DSC(C)$ or $= n S \in DSC(C)$,
where $n > 0$ and S *is functional*
then $\leq 1 S \in DSC(C)$

(2) if $\exists S.C_1 \in DSC(C)$ or $\geq n S \in DSC(C)$ or $= n S \in DSC(C)$,
where $n > 0$
and $INV(S) = S_1$, S_1 *is inverse functional*
then $\leq 1 S \in DSC(C)$

Inverse Rule

if $\exists S.C_1 \in DSC(C)$ and $INV(S) = S_1$,
and $C_2 \sqsupseteq C_1$ and $C_2 \sqsubseteq \forall S_1 C_3$
then $C_3 \in DSC(C)$

Symmetric Rule

if $\exists S.C_1 \in DSC(C)$ and S is a symmetric property,
and $C_2 \sqsupseteq C_1$ and $C_2 \sqsubseteq \forall S C_3$
then $C_3 \in DSC(C)$

Transitive Rule

if $\forall S.C_1 \in DSC(C)$ and S is a transitive property,
then $\forall S\ \forall S.C_1 \in DSC(C)$

Intersection Rule

if $C \wedge C_1 \in DSC(C)$,
then $C \in DSC(C)$ and $C_1 \in DSC(C)$

Subproperty Rule

(1) if $\forall S.C_1 \in DSC(C)$ and $S_1 \sqsubset S$, then $\forall S_1.C_1 \in DSC(C)$

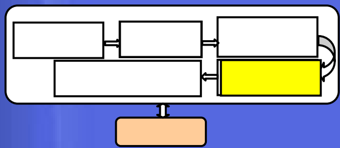
(2) if $\leq nS \in DSC(C)$ and $S_1 \sqsubset S$, then $\leq nS_1.C \in DSC(C)$

(3) if $\exists S.C_1 \in DSC(C)$ and $S_1 \sqsupset S$, then $\exists S_1.C_1 \in DSC(C)$

(4) if $\geq nS \in DSC(C)$ and $S_1 \sqsupset S$, then $\geq nS \in DSC(C)$

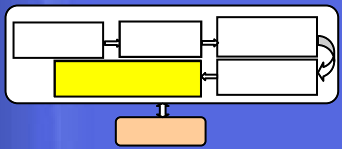
Other inference Rule

if C_1 can be inferred by any subset of $UC(C)$, where C is a named class
$C_1 \in DSC(C)$



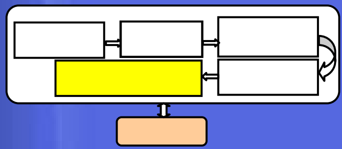
Determine The Most General Conflict

- Based on one simple observation.
  - If an OWL class **C** is disjointed with another class **D**, then any subclass of **C** is disjointed with **D** as well.
  - If an OWL class has both **C**, **D** and Csub as necessary conditions, it is more sensible to analyze the conflict between the class **C** and **D** rather than **D** and Csub.
- Most General Conflict (MGC)
  1. $MGC(C) \subseteq DSC(C)$
  2. Intersection of MGC(C) is unsatisfiable
  3. $\forall C_1, C_2 : MGC(C)$, such that $C_1 \sqsubset C_2 \Rightarrow C_1 = C_2$
  4. $\nexists C_1 : DSC(C) - MGC(C)$, such that $\exists C_2 : MGC(C)$ such that $C_2 \sqsubset C_1$ and Intersection of $MGC(C) \cup \{C_1\} - \{C_2\}$ is unsatisfiable.



Most General Conflict Analysis

- Most inconsistencies can be boiled down into a small number of ‘error patterns’.
- Determine which of the above cases led to an inconsistency,
- Use provenance information to trace where the problem come from.

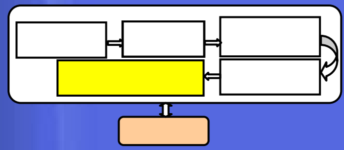


Error patterns

- The inconsistence is from some local definition:
  1. Having both a class and its complement class as super conditions.

     E.g.: MeatyVegetable $\sqsubset$ Vegetable,

     MeatyVegetable $\sqsubset$ not Vegetable



Error patterns

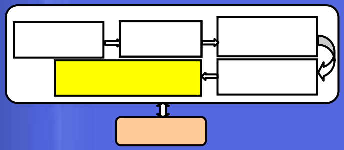
► The inconsistence is from some local definition:

1. Having both a class and its complement class as super conditions.
2. Having both universal and existential restrictions that act along the same property, whilst the filler classes are disjoint.

E.g.: $\text{VegetarianPizza} \sqsubset \forall\ \text{hasTopping Vegetable},$

$\text{VegetarianPizza} \sqsubset \exists\ \text{hasTopping Meat},$

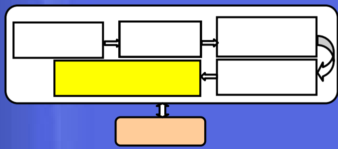
$\text{Vegetable} \sqcap \text{Meat} = \varnothing$



Error patterns

- The inconsistence is from some local definition:
    1. Having both a class and its complement class as super conditions.
    2. Having both universal and existential restrictions that act along the same property, whilst the filler classes are disjoint.
    3. Having a super condition that is asserted to be disjoint with ***owl:Thing***.

        E.g.: MyPizza $\sqsubset$ - owl:Thing



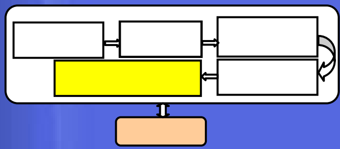
Error patterns

- The inconsistence is from some local definition:
  1. Having both a class and its complement class as super conditions.
  2. Having both universal and existential restrictions that act along the same property, whilst the filler classes are disjoint.
  3. Having a super condition that is asserted to be disjoint with ***owl:Thing***.
  4. Having a super condition that is an existential restriction that has a filler which is disjoint with the range of the restricted property.

E.g.: $\text{IceCreamPizza} \sqsubset \exists\ \text{hasTopping IceCream}$,

$\text{Ran(hasTopping)} = \text{PizzaTopping}$,

$\text{Food} \sqcap \text{IceCream} = \varnothing$



Error patterns

► The inconsistence is from some local definition:

1. Having both a class and its complement class as super conditions.
2. Having both universal and existential restrictions that act along the same property, whilst the filler classes are disjoint.
3. Having a super condition that is asserted to be disjoint with ***owl:Thing***.
4. Having a super condition that is an existential restriction that has a filler which is disjoint with the range of the restricted property.
5. Having an universal restriction with ***owl:Nothing*** as the filler and a *must existing restriction* along property relationships.

E.g.: Bread $\sqsubset \forall$ hasTopping owl:Nothing,

Bread $\sqsubset \exists$ hasTopping Meat

Error patterns

► The inconsistence is from some local definition:

1. Having both a class and its complement class as super conditions.
2. Having both universal and existential restrictions that act along the same property, and the filler classes are disjoint.
3. Having been asserted to be disjoint with ***owl:Thing***.
4. Having an existential restriction that has a filler which is disjoint with the range of the restricted property.
5. Having an universal restriction with ***owl:Nothing*** as the filler and a *must existing restriction* (existential/MinCard/Card) along property relationships.
6. Having ***n*** existential restrictions that act along a given property with disjoint fillers, whilst there is a `less then ***n*** restriction' along the property.

E.g.: $\text{BoringPizza} \sqsubset < \text{hasTopping } 2$,

$\text{BoringPizza} \sqsubset \exists \text{ hasTopping Meat}$,

$\text{BoringPizza} \sqsubset \exists \text{ hasTopping Vegetable}$,

$\text{Meat} \sqcap \text{Vegetable} = \varnothing$

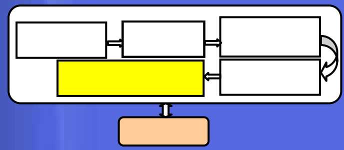
Error patterns

► The inconsistence is from some local definition:

1. Having both a class and its complement class as super conditions.
2. Having both universal and existential restrictions that act along the same property, whilst the filler classes are disjoint.
3. Having a super condition that is asserted to be disjoint with ***owl:Thing***.
4. Having a super condition that is an existential restriction that has a filler which is disjoint with the range of the restricted property.
5. Having an universal restriction with ***owl:Nothing*** as the filler and a *must existing restriction* along property relationships.
6. Having super conditions of ***n*** existential restrictions that act along a given property with disjoint fillers, whilst there is a \`less then ***n*** restriction' along the property.
7. Having super conditions containing conflicting cardinality restrictions.

E.g.: BoringFancyPizza $\sqsubset$ < hasTopping 2,

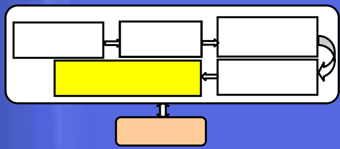
BoringFancyPizza $\sqsubset$ > hasTopping 2



Error patterns (2)

- The inconsistence is propagated from other source:
  1. Having a super condition that is an existential restriction that has an inconsistent filler.

     E.g.: MeatyVegetablePizza $\sqsubset \exists$ hasTopping MeatyVegetable



Error patterns (2)

- The inconsistence is propagated from other source:
  1. Having a super condition that is an existential restriction that has an inconsistent filler.
  2. Having a super condition that is a hasValue restriction that has an individual that is asserted to be a member of an inconsistent class.

E.g.: MeatyVegetablePizza $\sqsubset$ hasValue hasTopping aMeatyVegetable

aMeatyVegetable $\in$ MeatyVegetable

Conclusions

- ► A heuristic approach to ontology debugging.
- ► Using DL Reasoner, treating the reasoner as a 'black box'.
- ► Useful for beginners constructing small ontologies, through to domain experts and ontology engineers working with large complex ontologies,

Thank You!