Reasoning with OWL

Olivier Dameron¹, Alan Rector², Nick Drummond², Matthew Horridge²

1) EA 3888, Université de Rennes1, France
2) The University of Manchester, United Kingdom
Credits:

• Matthew Horridge, Holger Knublauch et al.
  A Practical guide to building OWL ontologies using the Protégé-OWL plugin and CO-ODE tools

• Natasha Noy, Alan Rector
  W3C “Semantic Web Best Practice” Working Group
Objective

Acquiring an in-depth understanding of the OWL-DL semantics in order to perform advanced reasoning tasks

• We will rely on the pizza example for:
  • a better formalization of the domain knowledge
  • leveraging OWL-DL reasoning capabilities for an easier curation of the ontology
  • an overview of some good practice
Outline

- OWL semantics
- Open world assumption
- Reasoning with individuals
Getting started

1. Getting Protégé
   - version 4.0
   - http://protege.stanford.edu

2. Getting some documentation
   - http://protege.stanford.edu/doc/users.html
   - OWL Tutorial: http://www.co-ode.org
   - Wiki: http://protege.cim3.net/cgi-bin/wiki.pl
   - Mailing lists
1. Use the protege2007owlTutorial-01.owl ontology from:

2. Launch Protégé

3. Select “Open OWL ontology”

4. Retrieve your local copy of the ontology
OWL Semantics
(*the theoretical part*)
Individuals

- Atoms
- Individuals have an identity and can be counted
- They are fundamental for understanding the semantics of DL...
- ... but you hardly use them when building ontologies
Properties

- A property = binary relationships btw individuals
- Domain, Range
  - Used as axioms (e.g. hasTopping and ice creams)
- Subproperties
- Characteristics
  - Transitive: e.g. hasPart, hasAncestor...
  - Symmetric: e.g. isSiblingOf...
  - Functional: e.g. hasSSN, hasMother...
NB Transitive - the ingredients of ingredients are ingredients of the whole.
Functional Properties

- Functional property: each element of the domain can have 0 or 1 image in the range
  - ex: `hasBiologicalMother`, `isToppingOf`, `isBaseOf`, ...

- If a property is functional, then its inverse is inverse functional
  - ex: `hasTopping`

- A property can be both functional and inverse-functional
  - ex: `hasSSN`, `hasBase`
  - not all do! -> `hasBiologicalMother`
Any given instance of topping should only be added to a single pizza (no cheap half-measures on our pizzas).
Note that hasTopping is inverse functional because isToppingOf is functional@en
Classes

- A class is a set of individuals
  - **Special classes:**
    - top ($\top$) = owl:Thing i.e. set of all the individuals
    - bottom ($\bot$) = empty set
  - **Can be combined using set operators**
    - subset (subsumption)
    - disjoint sets
    - union
    - intersection
    - complement
By default, any individual MAY be an instance of any classes => partial overlap of classes is assumed
$A \sqsubseteq B : \text{all the instances of } A \text{ are instances of } B$ (A is subClass of B)
Classes

• **Cumulative approach:** combine classes
  • using set operators (union, intersection, complement)
  • express constraints
  • define complex concepts

• **Intensional approach:** describe the characteristics of a class and the system will automatically:
  • recognize that an individual is an instance of it
  • recognize that it is a subclass or a superclass of another class
Combining Classes
Objective

- Combine classes using the OR, AND and NOT operators
- Refer to the semantics of these operators (and avoid some basic mistakes)

=> find out which pizza are:
  - Cheesy and vegetarian
  - Cheesy or vegetarian
  - Vegetarian and not vegetarian
Prerequisite

- Pizza
  - VegetarianPizza
  - CheesyPizza
  - NamedPizza
    - MargheritaPizza
    - AmericanPizza
    - CaprinaPizza

Don't worry about the toppings, this is the next step!
• **Create** CheesyAndVegetarianPizza as a subclass of Pizza
  • so far, except for the name, we have not provided any meaning
  • we have not exploited the cumulative approach

• **Add the necessary condition:**
  VegetarianPizza \( \cap \) CheesyPizza

• **Classify**
Superclasses

- Pizza
- VegetarianPizza

Class Description: CheesyAndVegetarianPizza

- Equivalent classes
- Superclasses:
  - Pizza
  - VegetarianPizza and CheesyPizza
- Inherited anonymous classes:
  - hasBase some PizzaBase
  - Pizza that hasTopping only VegetarianTopping
  - Pizza that hasTopping some CheeseTopping
- Instances
- Disjoint classes

Asserted class hierarchy inferred class hierarchy
is equivalent to:

... but the reasoning would have been trivial :-}
A ∩ B = set of indiv. instances of A and of B
AND (Intersection)

Ex: VegetarianPizza \( \cap \) CheesyPizza
Create CheesyOrVegetarianPizza as a subclass of Pizza

Add the necessary condition:
VegetarianPizza \cup CheesyPizza

Classify :-(

OR (Union)
Superclasses:
- Pizza
- VegetarianPizza
  - or CheeseyPizza
CheesyPizza and VegetarianPizza are not recognized as subclasses of CheesyOrVegetarianPizza.
OR (Union)

\[ A \cup B = \text{set of indiv. instances of } A \text{ or of } B \]
OR (Union)

Ex: VegetarianPizza $\cup$ CheesyPizza
OR (Union)

CheesyOrVegetarianPizza

A \cup B

CheesyOrVeggie \sqsubset VegetarianPizza \cup CheesyPizza
There could be instances of CheesyPizza (red dot) that are not instances of CheesyOrVegetarianPizza...

... therefore, CheesyPizza is not subclass of CheesyOrVeggie
• Now, use a definition for CheesyOrVegetarianPizza
(tip: right-click is your friend)
your friend is here!
Equivalent classes

VegetarianPizza
or CheesyPizza
Examples

• Declare MargheritaPizza to be a VegetarianPizza
• Declare AmericanPizza to be a CheesyPizza
• Declare CaprinaPizza to be both CheesyPizza and VegetarianPizza
• Classify
  • :-)  
  • why isn't CaprinaPizza classified as expected?
Class Description: American

- **CheesyPizza**
- **CheesyAndVegetarianPizza**
- **MeatyPizza**
- **NamedPizza**
  - **American**
  - **AmericanHot**
  - **Cajun**
  - **Capricciosa**
  - **Caprina**
  - **Fiorentina**
  - **FourSeasons**
  - **FruttiDiMare**
  - **Giardiniera**
  - **LaReine**
  - **Margherita**
  - **Mushroom**
  - **Napoleotana**
  - **Parmense**
  - **PolloAdAstra**
  - **PrinceCarlo**
  - **QuattroFormaggi**
  - **Rosa**
  - **Siciliana**
  - **SloppyGiuseppe**
  - **Soho**
  - **Veneziana**
  - **ThinAndCrispyPizza**
  - **VegetarianPizza**

Inferred class hierarchy: American

- **CheesyPizza**
- **MeatyPizza**
- **NamedPizza**
  - **American**
  - **AmericanHot**
  - **Cajun**
  - **Capricciosa**
  - **Caprina**
  - **Fiorentina**
  - **FourSeasons**
  - **FruttiDiMare**
  - **Giardiniera**
  - **LaReine**
  - **Margherita**
  - **Mushroom**
  - **Napoleotana**
  - **Parmense**
  - **PolloAdAstra**
  - **PrinceCarlo**
  - **QuattroFormaggi**
  - **Rosa**
  - **Siciliana**
  - **SloppyGiuseppe**
  - **Soho**
  - **Veneziana**
  - **ThinAndCrispyPizza**
  - **VegetarianPizza**

Instances:

- **hasBase some PizzaBase**
- **Pizza**
  - **that hasTopping some MeatTopping**
AmericanPizza and CaprinaPizza are recognised as cheesey.
MargheritaPizza and CaprinaPizza are recognised as veggie.
... but Caprina is not recognised as CheesyAndVeggie.
AND (Intersection)

Ex: VegetarianPizza ∩ CheesyPizza
Equivalent classes

VegetarianPizza and CheeseyPizza
NEGATION (Complement)

Ex: \( \neg \) VegetarianPizza
NEGATION (Complement)

• Create VegetarianTopping as a subclass of PizzaTopping

• A Vegetarian topping is
  • a topping
  • neither a meat topping, nor a fish topping
VegetableTopping, FruitTopping, ... are not recognised as VegetarianToppings
NEGATION (Complement)

• Create VegetarianTopping as a subclass of PizzaTopping
• A Vegetarian topping is neither a meat topping, nor a fish topping
• Classify
• Why do we have to provide a Necessary and Sufficient definition?
Equivalent classes

PizzaTopping
and not MeatTopping
and not FishTopping
Inferred class hierarchy: VegetarianTopping

- Thing
  - Nothing
  - DomainConcept
  - Food
    - Pizza
    - PizzaBase
    - PizzaTopping
      - FishTopping
      - MeatTopping
      - VegetarianTopping
        - VegetarianToppingNick
          - CheeseTopping
          - FruitTopping
          - HerbSpiceTopping
          - NutTopping
          - SauceTopping
          - VegetableTopping

Class Description:
- Equivalent classes

- PizzaTopping
  and not MeatTopping
  and not FishTopping

- Superclasses
  - PizzaTopping

- Inherited anonymous classes

- Instances

- Disjoint classes

NEGATION (Complement)

- A Vegetarian topping is neither a meat topping, nor a fish topping
- Why do we have to provide a Necessary and Sufficient definition?
  - it ensures that all the instances of PizzaTopping that are neither instances of MeatTopping nor of FishTopping are inferred to be instances of VegetarianTopping
NEGATION (Complement)

- Create VegetarianTopping as a subclass of PizzaTopping
- A Vegetarian topping is neither a meat topping, nor a fish topping
- Why do we have to provide a N&S definition?
- Create NonVegetarianTopping
- Classify
Equivalent classes

PizzaTopping and not VegetarianTopping
NEGATION (Complement)

- Note that the reasoner found out that CheeseTopping and VegetableTopping are subclasses of VegetarianTopping whereas the definition of VegetarianTopping does not mention CheeseTopping nor VegetableTopping (intentionality)
Expressing constraints
Objective

Application of the intensional approach: leverage the expressivity of the OWL-DL language for a precise representation of the classes' features

- We will describe the pizza ingredients and use the reasoner to find out which ones are cheesy and/or vegetarian
Getting in sync!

If you need to catch-up, the ontology at this point is protege2007owlTutorial-02.owl

from:

Constraints

1. Quantifier restriction (at least one, all of)
   • How to represent the fact that every pizza must have \emph{at least} a topping?
   • How to represent the fact that \emph{all} the ingredients of a vegetarian pizza must be vegetarians?

2. Cardinality restrictions
   • How to represent that a Hand must have 5 fingers as parts?

3. hasValue restrictions
   • How to define the value of a relation for a class?
Principles

• A restriction describes an anonymous class composed of all the individuals that satisfy the restriction
  • e.g. all the individuals that have (amongst other things) mozzarella as topping

• This anonymous class is used as a superclass of the (named) class we want to express a constraint on
  • e.g. MargheritaPizza
 Existential restriction

• $(\exists \text{ hasTopping Mozzarella}) : $set of the individuals being linked to at least one instance of Mozzarella through the $\text{hasTopping}$ property
  • They can be linked to multiple instances of Mozzarella
  • They can also be linked to instances of other classes (provided domain and range integrity)

• $\text{Margherita} \subseteq (\exists \text{ hasTopping Mozzarella})$
Existential restriction

• $(\exists \: hasTopping \: Mozzarella) : \text{set of the individuals being linked to at least one instance of Mozzarella through the } hasTopping \text{ property}$
  • They can be linked to multiple instances of Mozzarella
  • They can also be linked to instances of other classes (provided domain and range integrity)

• Margherita $\subseteq (\exists \: hasTopping \: Mozzarella)$
  • Other pizze can also have Mozzarella!
Complete the ontology

- Define CheesyPizza as a pizza having at least one cheese topping
- Remove the fact that AmericanPizza and CaprinaPizza are subclasses of CheesyPizza!
Pizza
that hasTopping some CheeseTopping
Class Annotations: CheezyPizza

Annotation:

comment

Any pizza that has at least 1 cheese topping @en

Class Description: CheezyPizza

Equivalent classes

- Pizza
- hasTopping some CheeseTopping

Superclasses

- Pizza
- CheezyOrVegetarianPizza

Inherited anonymous classes

- hasBase some PizzaBase
  - VegetarianPizza
    or CheezyPizza

Instances

Disjoint classes
Universal restriction

• $(\forall \text{hasTopping VegetarianTopping}) : \text{set of all the individuals only linked to instances of VegetarianTopping through the hasTopping property}$

• Warning: also includes all the individuals linked to nothing through the hasTopping property
Universal restriction

- $(\forall \text{hasTopping VegetarianTopping})$
- Remove the fact that MargheritaPizza and CaprinaPizza are subclasses of VegetarianPizza
- Define VegetarianPizza as any pizza for which all the toppings are vegetarian toppings
- Classify :-(
Pizza that hasTopping only VegetarianTopping
Any pizza that only has vegetarian toppings or no toppings is a VegetarianPizzaEquiv1. Should be inferred to be equivalent to VegetarianPizzaEquiv2. Not equivalent to VegetarianPizza because PizzaTopping is not covering en.

```
:-(
```
A pizza that only has vegetarian toppings or no toppings is a VegetarianPizzaEquiv1. Should be inferred to be equivalent to VegetarianPizzaEquiv2. Not equivalent to VegetarianPizza because PizzaTopping is not covering.
Universal restriction

- Why Margherita and Caprina pizze were not recognised as vegetarian pizze? (even though the vegetarian toppings were correctly recognised)
- ... find out in a few slides
Cardinality restriction

- **PizzaWithTwoToppings**
  - Pizza \( \sqcap (\text{hasTopping} = 2) \)
- **PizzaWithFiveOrMoreToppings**
  - Pizza \( \sqcap (\text{hasTopping} \geq 5) \)
- **PizzaWithThreeOrLessToppings**
  - Pizza \( \sqcap (\text{hasTopping} \leq 3) \)
- **Warning:** This is NOT qualified cardinality restr.
Equivalent classes

Pizza

that hasTopping exactly 2 Thing
Equivalent classes

Pizza
that hasTopping max 3 Thing
Inferred class hierarchy: PizzaWithFiveOrMoreToppings

- Thing
- Nothing
- DomainConcept
- Food
  - Pizza
    - CheesyOrVegetarianPizza
    - MeatyPizza
    - NonmeatPizza
    - PizzaWithFiveOrMoreToppings
      - AmericanHot
      - Cajun
      - Capricciosa
      - Fiorentina
      - FourSeasons
      - Giardiniera
      - LaReine
      - Napoletana
      - Parmigiana
      - PolloAdAstra
      - PrinceCarlo
      - Siciliana
      - SloppyGuiseppe
      - Soho
      - Veneziana
  - PizzaWithFiveOrLessToppings
    - ThinAndCrispyPizza
  - PizzaBase
  - PizzaTopping

Class Annotations: PizzaWithFiveOrMoreToppings
- Annotations:

:-)

Class Description: PizzaWithFiveOrMoreToppings
- Equivalent classes:
  - Pizza
  - that has Topping min 5 Thing

Superclasses:
- Pizza

Inherited anonymous classes:
- has base some PizzaBase

Instances:

Disjoint classes:
PizzaWithTwoToppings is correctly recognized as a subclass of PizzaWithThreeOrLessToppings...

... but MargheritaPizza is not recognized as a PizzaWithTwoToppings (hint...)
Open world assumption
Open VS Closed World Reasoning

- Remember a few slides ago ???
- MargheritaPizza ⊆ (∃ has\text{Topping} Mozzarella) \cap (∃ has\text{Topping} Tomato)
- VegetarianPizza = Pizza \cap (∀ has\text{Topping VegetarianTop.})
- Tomato and Mozzarella ARE Vegetarian toppings
- So, why isn't Margherita classified under VegetarianPizza ?
Open VS Closed World Reasoning

- Remember a few slides ago ???
- MargheritaPizza ⊨ (∃ hasTopping Mozzarella) ⊨ (∃ hasTopping Tomato)
- VegetarianPizza = Pizza ⊨ (∀ hasTopping VegetarianTop.)
- Tomato and Mozzarella ARE Vegetarian toppings
- Because some Margheritas may have other toppings (e.g. HotSpicedBeefTopping)!
Open VS Closed World Reasoning

- Closed-World reasoning
  - Negation as failure
  - Anything that cannot be found is false
  - Reasoning about this world

- Open-World reasoning
  - Negation as contradiction
  - Anything might be true unless it can be proven false
  - Reasoning about any world consistent with the model
Margherita pizzas only have Tomato and Mozzarella for topping

- MargheritaPizza ⊑ (∃ hasTopping Mozzarella) ⊓ (∃ hasTopping Tomato) ⊓ ?????
Need for closure

Margherita pizzas **only** have Tomato and Mozzarella for topping

- MargheritaPizza ⊆ (∃ hasTopping Mozzarella) ⊓ (∃ hasTopping Tomato) ⊓ (∀ hasTopping ???)
Margherita pizzas **only** have Tomato and Mozzarella for topping

- MargheritaPizza ⊆ (∃ hasTopping Mozzarella) ∩ (∃ hasTopping Tomato) ∩ (∀ hasTopping (Mozzarella ⊆ Tomato))
Margherita pizzas *only* have Tomato and Mozzarella for topping

- MargheritaPizza ⊆ (∃ hasTopping Mozzarella) \(∩\) (∃ hasTopping Tomato) \(∩\) (∀ hasTopping (Mozzarella \(∪\) Tomato))

- The universal constraint (∀) alone is not enough! **We need both ∃ and ∀ constraints**
Margherita pizzas **only** have Tomato and Mozzarella for topping

- MargheritaPizza ⊆ (∃ hasTopping Mozzarella) ∩ (∃ hasTopping Tomato) ∩ (∀ hasTopping (Mozzarella △ Tomato))

- Same principle for all the other pizze!
Inferred class hierarchy: VegetarianPizza

- Thing
  - Nothing
  - DomainConcept
    - Food
      - Pizza
        - CheeseOrVegetarianPizza
          - CheesePizza
          - VegetarianPizza
            - CheesyAndVegetarianPizza
              - Caprina
              - Margherita
            - CheesyPizza
            - MeatyPizza
            - NamedPizza
            - PizzaWithFiveOrMoreToppings
            - PizzaWithThreeOrLessToppings
            - ThinAndCrispyPizza
          - PizzaBase
          - PizzaTopping
    - CheeseOrVegetarianPizza
      - Caprina
      - Margherita

Class Annotations:
- Annotations:
  - comment
    - Any pizza

Class Description:
- Pizza
  - hasTopping only VegetarianTopping
  - has some PizzaBase
    - VegetarianPizza or CheesePizza

Superclasses:
- CheeseOrVegetarianPizza

Inherited Anonymous Classes:
- hasBase some PizzaBase
  - VegetarianPizza
  - CheesePizza

Instances:
- Disjoint Classes
Getting in sync!

If you need to catch-up, the ontology at this point is protege2007owlTutorial-03.owl from:

More fun with closure and defined classes

- Before we added the closures, why wasn't AmericanPizza recognised as a subclass of MargheritaPizza?
Need for closure

Mozzarella + Tomato

Mozzarella + Tomato + Pepperoni
Need for closure

Mozzarella + Tomato

Mozzarella + Tomato + Pepperoni

MargheritaPizza

AmericanPizza
Margherita pizzas only have Tomato and Mozzarella for topping

- MargheritaPizza = (∃ hasTopping Mozzarella) ∩ (∃ hasTopping Tomato) ∩ (∀ hasTopping (Mozzarella ⊕ Tomato))
Getting in sync!

If you need to catch-up, the ontology at this point is protege2007owlTutorial-04.owl from:

More fun with cardinality

• Why isn't MargheritaPizza classified under PizzaWithTwoToppings?
More fun with cardinality

- Why isn't MargheritaPizza classified under PizzaWithTwoToppings?

- Hints:
  - Why isn't it classified under PizzaWithThreeOrLessToppings?
  - Why isn't it even classified under PizzaWithFiveOrMoreToppings? ... do Margherita pizze have exactly 4 toppings?
More fun with cardinality

• Why isn't MargheritaPizza classified under PizzaWithTwoToppings?

• Still... the open-world assumption: imagine one instance of MargheritaPizza having as topping:
  • one instance of MozzarellaTopping
  • one other instance of MozzarellaTopping
  • one instance of TomatoTopping
Class Description: Margherita2

Equivalent classes:

- hasTopping some MozzarellaTopping
  and hasTopping some TomatoTopping
  and hasTopping exactly 2 Thing

Superclasses:

- NamedPizza

Inherited anonymous classes:

- hasBase some PizzaBase

Instances:

Disjoint classes:

 asserted: 11 inferred: 1
hasValue restriction

- So far, we have been narrowing the range of relationship
  - create the class Person
  - create the relation hasPizzaMaker: Pizza -> Person
  - create ItalianPerson as a subclass of Person
  - define GenuinePizza = (∃ hasPizzaMaker ItalianPers.)
Pizza that hasPizzaMaker some ItalianPerson
hasValue restriction

- So far, we have been narrowing the range of relationship

- We may also want to restrict it to a precise value (and not to a set of values)
  - create olivier as an instance of Person
  - define OliviersPizza = (hasPizzaMaker ø olivier)
hasValue restriction

- Create luigi as an instance of ItalianPerson
- Create LuigisPizza
- Classify :-}
ItalianPerson and FrenchPerson are not disjoint

Because there is no Unique Name Assumption, luigi and olivier could be the same person

Use the owl:differentFrom and owl:allDifferent constructs (in the OWL menu)!
Reasoning makes life easier :-) 

- Supports queries such as:
  - What are the vegetarian pizzas?
  - What are the cheesy pizzas?
  - What are the non-cheesy pizzas?
  - What are the cheesy vegetarian pizzas?
- ... it allows you to take advantage of the knowledge you put into your ontology
OWL and beyond...

OWL 1.1
Qualified Cardinality Restriction

- **OWL 1.0 Cardinality restrictions:**
  - PizzaWithTwoToppings
  - PizzaWithFiveOrMoreToppings
  - PizzaWithThreeToppingsOrLess

- **OWL 1.1 Qualified cardinality restrictions**
  - PizzaWithThreeCheese
  - PizzaWithAtLeastTwoCheese
  - PizzaWithAtLeastTwoCheeseAnd
Pizza

that hasTopping exactly 3 CheeseTopping
Pizza that hasTopping min 2 CheeseTopping
Additional features for properties

- Reflexivity
  - e.g. \textit{knows, isGreaterOrEqualTo}
  \[
  \forall a \in X, \ aRa
  \]

- Irreflexivity
  - e.g. \textit{isMotherOf, isGreaterThan}
  \[
  \forall a \in X, \neg (aRa)
  \]

- Antisymmetry
  - e.g. \textit{isAncestorOf, isGreaterOrEqualTo}
  \[
  \forall a, b \in X, \ aRb \land bRa \implies a = b
  \]
Character:
- Functional
- Inverse functional
- Transitive
- Symmetric
- Antisymmetric
- Reflexive
- Irreflexive

Description: hasPizzaMaker
- Equivalent object properties
- Super properties
- Inverse properties
- Disjoint properties

Property chains
Property chains

• Allow to describe (simple) composition of relations

• e.g.:
  if $X$ eats $Y$ and $Y$ hasIngredient $Z$
  then $X$ eats $Z$

• Notation: $fog(x) = f(g(x))$
Summary
Summary

1. Compositional approach
2. Intensional description
3. Reasoning
   - classification
   - open-world assumption
   - inconsistency