CC6202-1 LA WEB DE DATOS PRIMAVERA 2016

Lecture 5: Web Ontology Language (II)

Aidan Hogan aidhog@gmail.com

PREVIOUSLY ON "LA WEB DE DATOS"

RDF: Resource Description Framework

subject	predicate	object
ex:Irelan	d ex:partOf	ex:Europe
ex:Irelan	d rdf:type	ex:Country
ex:Irelan	d ex:capital	ex:Dublin
ex:Dubli	n ex:population	1,000,000



RDF Schema ...



(an example)

A special family ...







$\leftarrow \mathsf{OWL}$

TODAY'S TOPIC ...

An ontology is just some definitions ...

```
:hasDaughter rdfs:subPropertyOf :hasChild .
:hasChild rdf:type owl:AsymmetricProperty ; owl:inverseOf :hasParent ;
  rdfs:subPropertyOf :ancestorOf .
:ancestorOf rdf:type owl:TransitiveProperty .
ex:Carmela :hasChild ex:Michael .
ex:Michael :hasDaughter ex:Mary .
ex:Mary a :Person .
:Person owl:equivalentClass
 [ owl:qualifiedCardinality 2 ;
  owl:onProperty :hasParent ;
  owl:onClass :Person ] .
```

... but what do they mean? ... and what can we do with ontologies?



MODELS ...

A model is any world that an ontology might describe

:hasDaughter rdfs:subPropertyOf :hasChild .
:hasChild rdfs:subPropertyOf :ancestorOf .
:ancestorOf rdf:type owl:TransitiveProperty .
ex:Carmela :hasChild ex:Michael .
ex:Michael :hasDaughter ex:Mary .

:hasChild

:ancestorOf



-

:hasDaughter :hasChild :ancestorOf



... not a model, since Carmela would need to be an ancestor of Mary

A model is any world that an ontology might describe

:hasDaughter rdfs:subPropertyOf :hasChild .
:hasChild rdfs:subPropertyOf :ancestorOf .
:ancestorOf rdf:type owl:TransitiveProperty .
ex:Carmela :hasChild ex:Michael .
ex:Michael :hasDaughter ex:Mary .



... a model

(leaving aside things like OWL definitions, reflexive owl:sameAs, etc.)

A <u>model</u> is any world that an ontology might describe

:hasDaughter rdfs:subPropertyOf :hasChild .
:hasChild rdfs:subPropertyOf :ancestorOf .
:ancestorOf rdf:type owl:TransitiveProperty .
ex:Carmela :hasChild ex:Michael .
ex:Michael :hasDaughter ex:Mary .





... also a model

(Under Open World Assumption, Ontology can describe part of the world)

A model is any world that an ontology might describe

:hasDaughter rdfs:subPropertyOf :hasChild .
:hasChild rdfs:subPropertyOf :ancestorOf .
:ancestorOf rdf:type owl:TransitiveProperty .
ex:Carmela :hasChild ex:Michael .
ex:Michael :hasDaughter ex:Mary .





... also a model

(Since we don't know what ex:Cake, ex:Carmela, etc., actually refer to)

Mapping of names to things part of model











ex:Carmela



ex:Mary

(Different names mapped to different things means a different model!)

ENTAILMENT ...

Ontology *O* entails *O*′ (*O* ⊨ *O*′)

```
:hasDaughter rdfs:subPropertyOf :hasChild .
:hasChild rdf:type owl:AsymmetricProperty ; owl:inverseOf
   :hasParent ;
  rdfs:subPropertyOf :ancestorOf .
:ancestorOf rdf:type owl:TransitiveProperty .
ex:Carmela :hasChild ex:Michael .
ex:Michael :hasDaughter ex:Mary .
ex:Mary a :Person .
:Person owl:equivalentClass
  [ owl:qualifiedCardinality 2 ;
    owl:onProperty :hasParent ;
    owl:onClass :Person ] .
ex:Michael :hasParent ex:Carmela .
ex:Michael :hasChild ex:Mary .
ex:Carmela :ancestorOf ex:Mary .
```

Any model of *O* is a model of *O*'

(O' forms part of the models of O)
 (O' says nothing new over O)

Entailment symbol: ⊧

$O \models O'$

Ontology Entailment

ex:Michael :hasParent ex:Carmela .
ex:Michael :hasChild ex:Mary .
ex:Carmela :ancestorOf ex:Mary .

ex:Carmela :ancestorOf ex:Mary .
ex:Mary :likes ex:Cake .

Does O entail O' ($O \models O'$)?

No! There are models of *O* that are not of *O*' ...



REASONING TASKS ...

•••

Write down entailments

```
:hasDaughter rdfs:subPropertyOf :hasChild .
:hasChild rdf:type owl:AsymmetricProperty ; owl:inverseOf
   :hasParent ;
  rdfs:subPropertyOf :ancestorOf .
:ancestorOf rdf:type owl:TransitiveProperty .
ex:Carmela :hasChild ex:Michael .
ex:Michael :hasDaughter ex:Mary .
ex:Mary a :Person .
:Person owl:equivalentClass
  [ owl:qualifiedCardinality 2 ;
    owl:onProperty :hasParent ;
    owl:onClass :Person ] .
ex:Michael :hasParent ex:Carmela .
ex:Michael :hasChild ex:Mary .
ex:Carmela :ancestorOf ex:Mary .
```

Any problems with this?

Write down entailments

```
:hasDaughter rdfs:subPropertyOf :hasChild .
:hasChild rdf:type owl:AsymmetricProperty ; owl:inverseOf
   :hasParent ;
  rdfs:subPropertyOf :ancestorOf .
:ancestorOf rdf:type owl:TransitiveProperty .
ex:Carmela :hasChild ex:Michael .
ex:Michael :hasDaughter ex:Mary .
ex:Mary a :Person .
:Person owl:equivalentClass
  [ owl:qualifiedCardinality 2 ;
    owl:onProperty :hasParent ;
    owl:onClass :Person ] .
ex:Michael :hasParent ex:Carmela .
ex:Michael :hasChild ex:Mary .
ex:Carmela :ancestorOf ex:Mary .
ex:Mary :hasParent _:parent1 . _:parent1 a :Person .
ex:Mary :hasParent _:parent2 . _:parent2 a :Person .
_:parent1 :hasParent _:parent11 . _:parent11 a :Person .
_:parent2 :hasParent _:parent12 . _:parent12 a :Person . ...
```

Write down entailments

```
:hasDaughter rdfs:subPropertyOf :hasChild .
:hasChild rdf:type owl:AsymmetricProperty ; owl:inverseOf
   :hasParent :
  rdfs:subPropertyOf :ancestorOf .
:ancestorOf rdf:type owl:TransitiveProperty .
ex:Carmela :hasChild ex:Michael .
ex:Michael :hasDaughter ex:Mary .
ex:Mary a :Person .
:Person owl:equivalentClass
  [ owl:qualifiedCardinality 2 ;
    owl:onProperty :hasParent ;
    owl:onClass :Person ] .
ex:Michael :hasParent ex:Carmela .
ex:Michael :hasChild ex:Mary .
ex:Carmela :ancestorOf ex:Mary .
:Person rdfs:subClassOf [ owl:maxQualifiedCardinality 2 ;
   owl:onProperty :hasParent ; owl:onClass :Person ] .
:Person rdfs:subClassOf [ owl:maxQualifiedCardinality 3 ;
   owl:onProperty :hasParent ; owl:onClass :Person ] . ...
```

Write down entailments

Entailments are infinite ...

... which makes it tricky to write them <u>all</u> down ...



Does *O* have a "model"?



So does *O* have a model?



YES! Ontology *O* is **Satisfiable**!

Does *O* have a "model"?



So does *O* have a model now?



YES! Ontology *O* is still **Satisfiable**!

Does *O* have a "model"?

```
:Person owl:equivalentClass
  [ owl:qualifiedCardinality 2 ; owl:onProperty :hasParent ;
    owl:onClass :Person ] .
:FirstPerson a :Person ,
  [ owl:qualifiedCardinality 2 ; owl:onProperty :hasParent ;
    owl:onClass :Ape ] .
:Ape owl:disjointWith :Person .
```

What more would we have to add to *O* to make it Unsatisfiable?



Does *O* have a "model"?



:FirstPerson a owl:Nothing

Entailment checking:

Does *O* entail *O*'?

```
:hasDaughter rdfs:subPropertyOf :hasChild .
:hasChild rdf:type owl:AsymmetricProperty ;
   owl:inverseOf :hasParent ;
   rdfs:subPropertyOf :ancestorOf .
:ancestorOf rdf:type owl:TransitiveProperty .
ex:Carmela :hasChild ex:Michael .
ex:Michael :hasDaughter ex:Mary .
ex:Michael :hasParent ex:Carmela .
ex:Michael :hasChild ex:Mary .
ex:Carmela :ancestorOf ex:Mary .
```

Alternatively: Are all models of *O* models of *O*' too?

REASONING ...

How can we perform reasoning?

Does Ontology *O* entail *O*'?

:hasDaughter rdfs:subPropertyOf :hasChild .
ex:Michael :hasDaughter ex:Mary .

ex:Michael :hasChild ex:Mary .

Could instead ask: is "O U ¬O'" unsatisfiable?

:hasDaughter rdfs:subPropertyOf :hasChild .
ex:Michael :hasDaughter ex:Mary .
[] owl:sourceIndividual ex:Michael ;
 owl:assertionProperty :hasChild ;
 owl:targetIndividual ex:Mary .

Can reduce entailment to unsatisfiability!

So how do we test unsatisfiability then?

NOTHING IS IMPOSSIBLE THE WORD ITSELF SAYS I'M POSSIBLE Auchey Hepburn

(Audrey was clearly not a Computer Scientist)

UNDECIDABILITY ...

A simple Java program ...

```
public class Collatz {
    public static void collatz(int n) {
                                                               In: 6
        StdOut.print(n + " ");
                                                                    3
        if (n == 1) return;
                                                                   10
        else if (n % 2 == 0) collatz(n / 2);
        else collatz(3*n + 1);
                                                                    5
    }
                                                                   16
                                                                    8
    public static void main(String[] args) {
                                                                    4
        int N = Integer.parseInt(args[0]);
                                                                    2
        collatz(N);
        StdOut.println();
                                                                    1 (end)
    }
}
```

Does this Java program terminate on <u>all possible (positive) inputs</u>?

A simple Java program ...

```
public class Collatz {
    public static void collatz(int n) {
        StdOut.print(n + " ");
        if (n == 1) return;
        else if (n % 2 == 0) collatz(n / 2);
        else collatz(3*n + 1);
    }
    public static void main(String[] args) {
        int N = Integer.parseInt(args[0]);
        collatz(N);
        StdOut.println();
    }
}
```

- Collatz conjecture: an unsolved problem in mathematics
- If we knew that this program terminates or does not terminate on all natural numbers (not just ints), this problem would be solved!

Halting Problem

- Input: a program and an input to that program
- Output:
 - -true if the program halts on that input
 - false otherwise
- UNDECIDABLE: a general algorithm to solve the Halting Problem does not exist!
 - It will not halt for <u>all</u> program—input pairs!
 - It may halt for <u>some</u> program-input pairs

A quick sketch of Halting Problem proof





Problem: Consecutive '1's in π

- Input: A natural number *n*
- Output:
 - true if π contains *n* consecutive '1's
 - false otherwise

Is this problem **DECIDABLE** or **UNDECIDABLE**?

... i.e., does there exist a program that halts (with the correct answer) for all *n*?

What if we knew the maximum sequence of consecutive '1's in π ?

if (n ≤ MAX) return true; else return false;

there must exist a MAX sequence of consecutive '1's in π (even if it's ∞)
 ∴ there must exist a correct program that halts (even if we don't know its details)
 ∴ problem is DECIDABLE!

Problem: Collatz Halting Problem

- Input: [none]
- Output:
 - true if Collatz program halts on all inputs
 - false otherwise

Is this problem **DECIDABLE** or **UNDECIDABLE**?

(P1) return true;

(P2) return false;

• either (P1) or (P2) must be correct

∴ there must exist a correct program that halts (even if we don't know it)
∴ problem is **DECIDABLE**!

Halting Problem UNDECIDABLE in the general case (for <u>all</u> programs and inputs)

Domino Tiling Problem



Is this problem DECIDABLE or UNDECIDABLE?

- Input: A set of Dominos (like D)
- Output:
 - true if there exists a valid infinite tiling (like t)
 - false otherwise

Can reduce from Halting to Tiling



It has been shown that there exists a program that can reduce any Halting problem instance into a Domino Tiling problem instance

Now is the Domino Tiling problem **DECIDABLE** or **UNDECIDABLE**?

```
halts(p,in) {
    D = reduce(p,in);
    return hasTiling(D);
}
```

reduce(p,in) is DECIDABLE
 ∴ if hasTiling(D) were DECIDABLE, then halts(p,in) would be DECIDABLE
 but halts(p,in) is UNDECIDABLE
 ∴ hasTiling(D) must be UNDECIDABLE!

If we have a decidable reduction from an UNDECIDABLE problem A to another problem B, then B must be UNDECIDABLE

Reduce from Tiling to OWL entailment?



Does D have an infinite tiling?

Does OWL ontology O entail O'?

How can we encode a Domino Tiling question into an OWL ontology entailment question?

If we could do this, we could save ourselves trying to code a program to implement OWL entailment since we would know it was **UNDECIDABLE**

Some Description Logic symbols

- ⊑: sub-class/-property
- ≡: equivalent class/property
- 🛛: union
- Intersection
- T: top (class of everything)
- L: bottom (empty class)
- ∃: exists (someValuesFrom/hasValue)
- ∀: for all (allValuesFrom)
- -: not (complement, negation)
- – (superscript minus): inverse property
- {}: enumeration (owl:oneOf)
- Self, Trans, Dom, etc.: where symbols not available
- •: property chain
- C(x): class membership
- P(x,y): a triple (x,P,y)

- 1. Each tile must be a domino
 - Define dominos as a class *D*, a union of classes for each domino type:

$$D \equiv D_1 \sqcup D_2 \sqcup \ldots \sqcup D_{k-1} \sqcup D_k$$

Now what else do we need to encode?



- 1. Each tile must be a domino
 - Define dominos as a class *D*, a union of classes for each domino type:

$$D \equiv D_1 \sqcup D_2 \sqcup \ldots \sqcup D_{k-1} \sqcup D_k$$

2. Each tile can only be one domino type

How can we encode this in OWL?



- 1. Each tile must be a domino
 - Define dominos as a class *D*, a union of classes for each domino type:

 $D \equiv D_1 \sqcup D_2 \sqcup \ldots \sqcup D_{k-1} \sqcup D_k$

- 2. Each tile can only be one domino type
 - Define dominos types as pairwise disjoint:

$$D_i \sqcap D_j \sqsubseteq \bot (\text{for } 1 \le i < j \le k)$$

What else do we need to encode?



- 1. Each tile must be a domino
 - Define dominos as a class *D*, a union of classes for each domino type:

 $D \equiv D_1 \sqcup D_2 \sqcup \ldots \sqcup D_{k-1} \sqcup D_k$

2. Each tile can only be one domino type

- Define dominos types as pairwise disjoint:

$$D_i \sqcap D_j \sqsubseteq \bot (\text{for } 1 \le i < j \le k)$$

3. Each tile must have a tile to the right and above

How can we encode this?



- 1. Each tile must be a domino
 - Define dominos as a class *D*, a union of classes for each domino type:

 $D \equiv D_1 \sqcup D_2 \sqcup \ldots \sqcup D_{k-1} \sqcup D_k$

2. Each tile can only be one domino type

- Define dominos types as pairwise disjoint:

$$D_i \sqcap D_j \sqsubseteq \bot (\text{for } 1 \le i < j \le k)$$

- 3. Each tile must have a tile to the right and above
 - Define that a domino has some values from domino for right/above:

 $D \sqsubseteq (\exists r.D) \sqcap (\exists a.D)$

Are we there yet?



4. Tiles to the right and tiles above must match colour

- Define that a domino has all values from matching tiles right/above:

$$D_1 \sqsubseteq \forall r. (\bigsqcup_{D' \in R(D_1)} D') \sqcap \forall a. (\bigsqcup_{D' \in A(D_1)} D')$$

$$D_k \sqsubseteq \forall r. (\bigsqcup_{D' \in R(D_k)} D') \sqcap \forall a. (\bigsqcup_{D' \in A(D_k)} D')$$

Where:

 $R(D_i)$ denotes all domino types that can be to the right of D_i $A(D_i)$ denotes all domino types that can be above D_i

Are we there yet?



What condition are we missing?

- So far we could still have trees as models
- Need to state:

Tile above and to the right = Tile to the right and above



- 5. Tile right then above = Tile above then right
 - Define *d*iagonal tile using two property chains (above-right/right-above)
 - Declare functional (a tile can only have one such diagonal tile)

$$d \sqsubseteq a \circ r, \quad d \sqsubseteq r \circ a, \quad \mathsf{Func}(d)$$



What's the entailment question?

$$D \equiv D_1 \sqcup D_2 \sqcup \ldots \sqcup D_{k-1} \sqcup D_k$$
$$D_i \sqcap D_j \sqsubseteq \bot (\text{for } 1 \le i < j \le k)$$
$$D \sqsubseteq (\exists r.D) \sqcap (\exists a.D)$$
$$D_1 \sqsubseteq \forall r.(\bigsqcup_{D' \in R(D_1)} D') \sqcap \forall a.(\bigsqcup_{D' \in A(D_1)} D')$$
$$\ldots$$
$$D_k \sqsubseteq \forall r.(\bigsqcup_{D' \in R(D_k)} D') \sqcap \forall a.(\bigsqcup_{D' \in A(D_k)} D')$$
$$d \sqsubseteq a \circ r, \quad d \sqsubseteq r \circ a, \quad \mathsf{Func}(d)$$

???

Goal: Ontology *O* entails *O*' if and only if *D* has an infinite tiling

What's the entailment question?

$$D \equiv D_1 \sqcup D_2 \sqcup \ldots \sqcup D_{k-1} \sqcup D_k$$
$$D_i \sqcap D_j \sqsubseteq \bot (\text{for } 1 \le i < j \le k)$$
$$D \sqsubseteq (\exists r.D) \sqcap (\exists a.D)$$
$$D_1 \sqsubseteq \forall r.(\bigsqcup_{D' \in R(D_1)} D') \sqcap \forall a.(\bigsqcup_{D' \in A(D_1)} D')$$
$$\ldots$$
$$D_k \sqsubseteq \forall r.(\bigsqcup_{D' \in R(D_k)} D') \sqcap \forall a.(\bigsqcup_{D' \in A(D_k)} D')$$
$$d \sqsubseteq a \circ r, \quad d \sqsubseteq r \circ a, \quad \mathsf{Func}(d)$$

 $D \equiv \bot$

Goal: Ontology *O* entails *O'* if and only if *D* has an infinite tiling

If *D* has <u>any</u> member (a "tile"), it must have an infinite tiling! If *D* has no member, it must not have an infinite tiling.

Could also use satisfiability

$$D \equiv D_1 \sqcup D_2 \sqcup \ldots \sqcup D_{k-1} \sqcup D_k$$
$$D_i \sqcap D_j \sqsubseteq \bot (\text{for } 1 \le i < j \le k)$$
$$D \sqsubseteq (\exists r.D) \sqcap (\exists a.D)$$
$$D_1 \sqsubseteq \forall r. (\bigsqcup_{D' \in R(D_1)} D') \sqcap \forall a. (\bigsqcup_{D' \in A(D_1)} D')$$
$$\ldots$$
$$D_k \sqsubseteq \forall r. (\bigsqcup_{D' \in R(D_k)} D') \sqcap \forall a. (\bigsqcup_{D' \in A(D_k)} D')$$
$$d \sqsubseteq a \circ r, \quad d \sqsubseteq r \circ a, \quad \mathsf{Func}(d)$$
$$D(x)$$

Ontology *O* is satisfiable if and only if *D* has an infinite tiling Here, **x** is an arbitrary fresh term

OWL satisfiability/entailment is powerful





OWL satisfiability/entailment also **undecidable**! Otherwise could be used to solve Domino Tiling problem and the Halting problem ...

... and (given enough time), the Collatz conjecture ...

... and a bunch of other stuff



PRACTICAL REASONING

Options ...

Well great. What are we supposed to do now?

- Accept incomplete reasoners that halt
 - You may not get all the entailments ... so what entailments do you get?
- Accept complete reasoners that may not halt
 - Java is a language that lets you write programmes that may not halt
- Restrict OWL so we can't solve Halting/Tiling
 - Main problem tackled in Description Logics field: find decidable sublanguages of OWL without turning off too many features (and allowing efficient algorithms)

More next week ...

RECAP



OWL Definitions

• Models

- A world that the ontology could be true for
 - ... and a mapping from the terms of the ontology to that world
- Not necessarily the real world (just a consistent world)
- Can be much larger than the ontology describes
- The more detailed the ontology, the fewer its models

• Entailment

- An ontology O entails another ontology O' if any model of O is a model of O'
 - ... in which case O' adds no new information to O
 - ... in which case O' follow from O

OWL Reasoning Tasks

- Materialisation
 - Write down all the entailments from O'
 - Unfortunately, they are infinite
- Satisfiability
 - Does an ontology have any model?
 - If not, it is inconsistent/unsatisfiable
 - Unfortunately, satisfiability is undecidable
- Entailment (checking)
 - Does an ontology O entail another ontology O'?
 - If so, O' follows as a consequence of O
 - Can be reduced to satisfiability
 - Unfortunately, entailment checking is also undecidable

