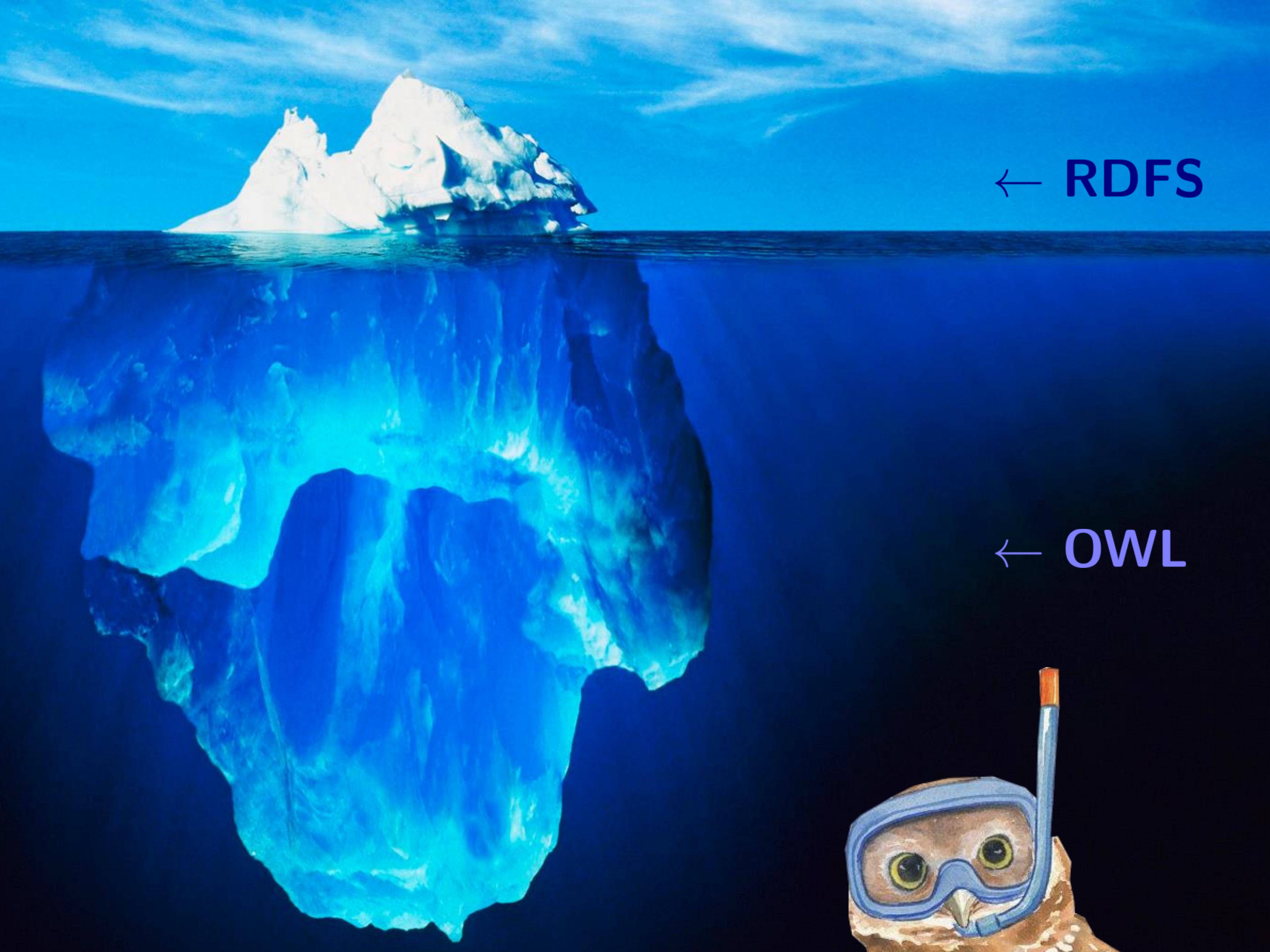
CC7220-1
La Web de Datos
Primavera 2022

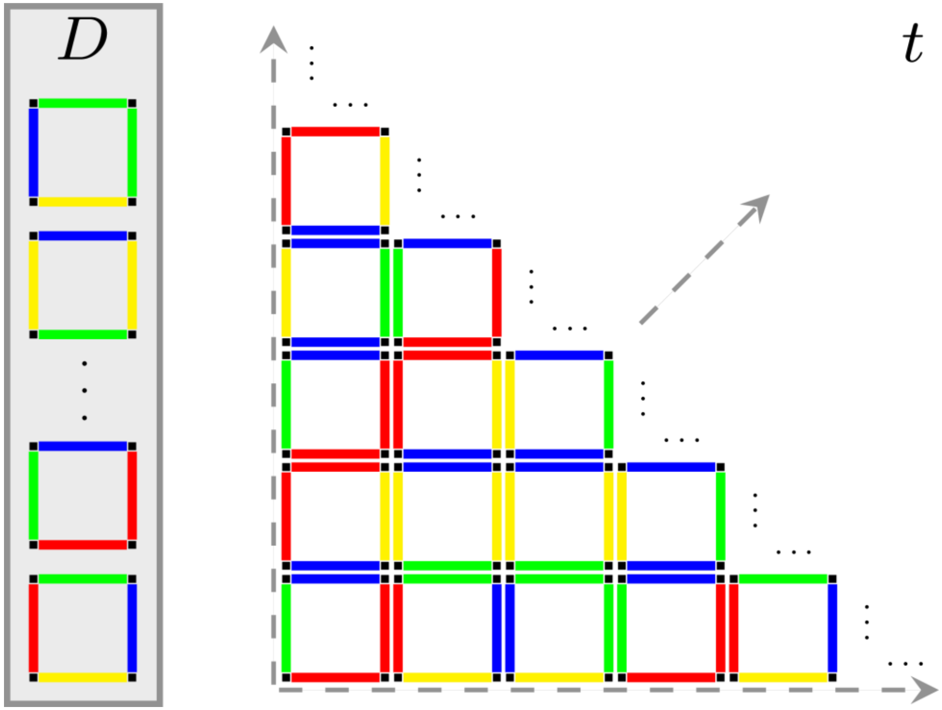
Lecture 6: Web Ontology Language (OWL) [III]

Aidan Hogan
aidhog@gmail.com

Last time …


← RDFS
← OWL

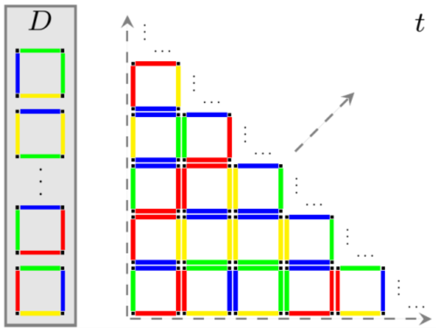
Domino Tiling Problem (Undecidable!)



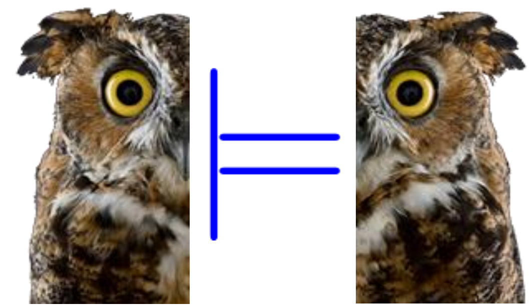
- Input: A set of Dominos (like *D*)
- Output:
  - `true` if there exists a valid infinite tiling (like *t*)
  - `false` otherwise

TODAY'S TOPIC

Reduce from Tiling to OWL entailment?



Reduction



Does D have an infinite tiling?

Does OWL ontology O entail O'?

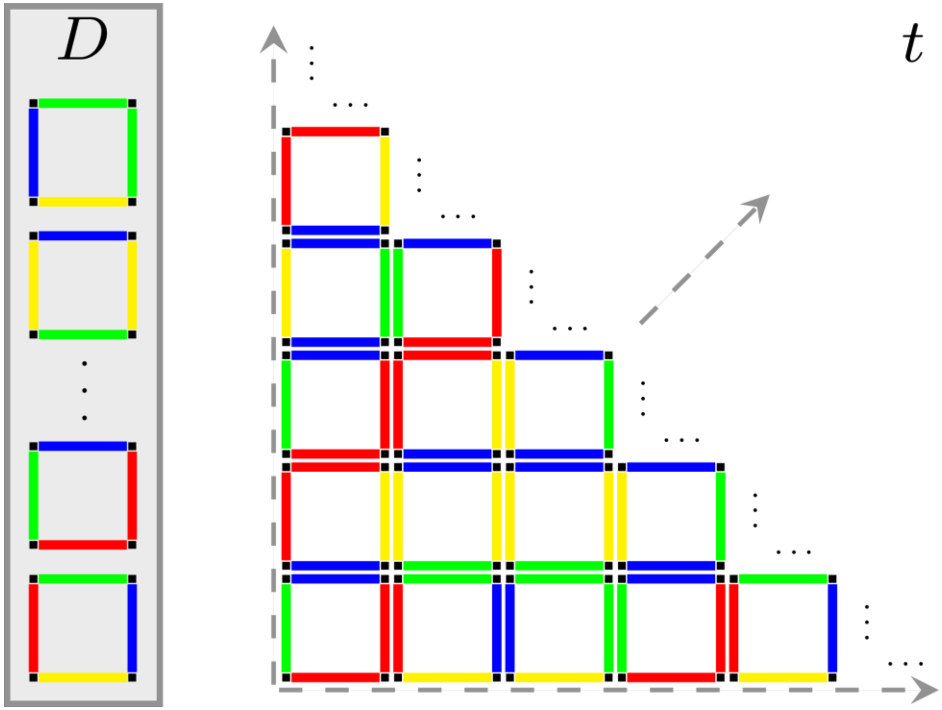
How can we encode a Domino Tiling question into an OWL ontology entailment question?

Based on talk/proof by Uli Sattler:
http://www.cs.man.ac.uk/~sattler/teaching/COMP61132-slides5.pdf

Some Description Logic symbols

- $\sqsubseteq$: sub-class/-property
- $\equiv$: equivalent class/property
- $\sqcup$: union
- $\sqcap$: intersection
- $\top$: top (class of everything)
- $\bot$: bottom (empty class)
- $\exists$: exists (`someValuesFrom`/`hasValue`)
- $\forall$: for all (`allValuesFrom`)
- $\neg$: not (complement, negation)
- $-$ (superscript minus): inverse property
- {}: enumeration (`owl:oneOf`)
- Self, Trans, Dom, etc.: where symbols not available
- $\circ$: property chain
- C(x): class membership
- P(x,y): a triple `(x,P,y)`

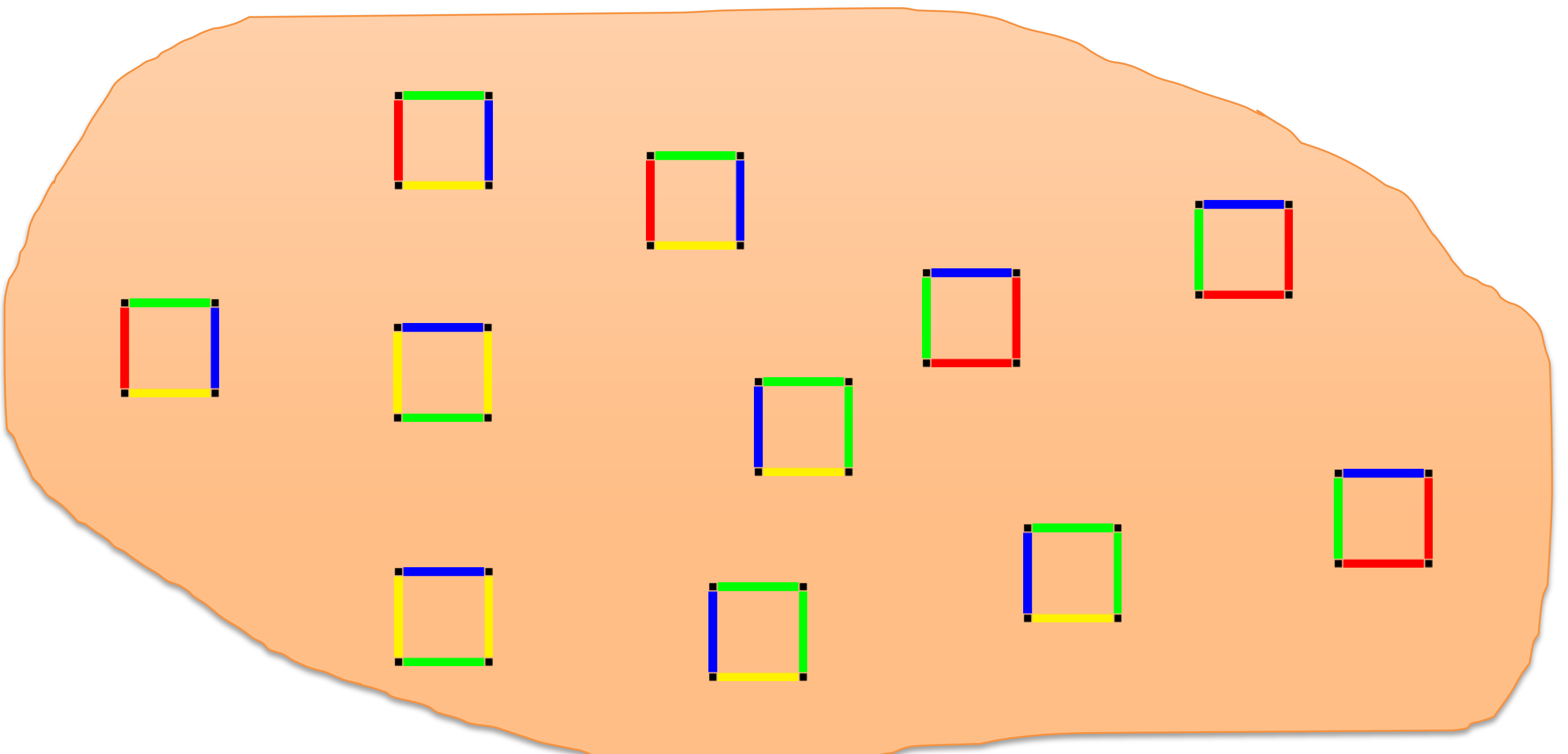
Domino Tiling Problem (Undecidable!)



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

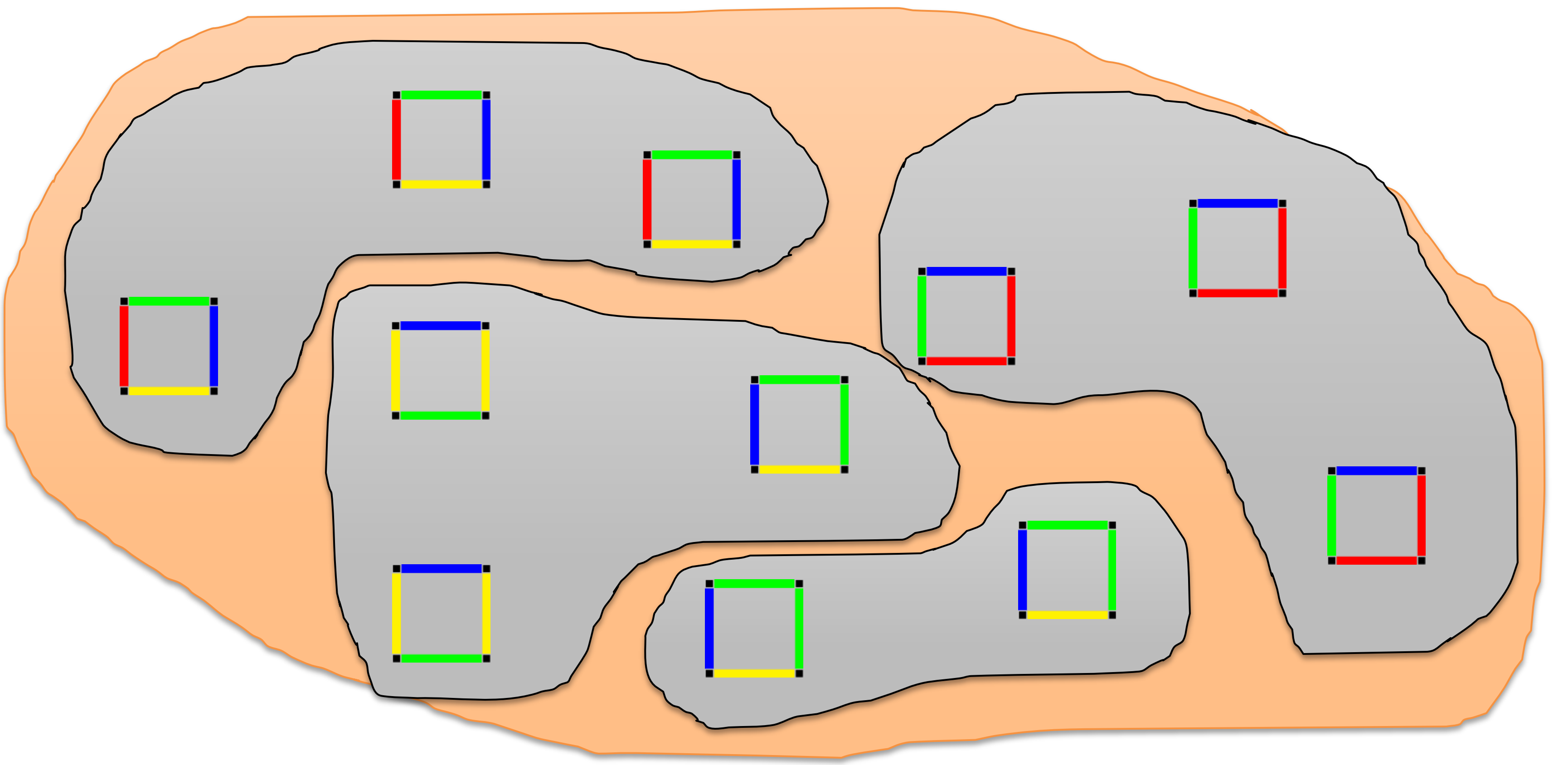
Domino Tiling Problem: Some terminology

- Tile: A Piece



Domino Tiling Problem: Some terminology

- Tile: A Piece
- Domino: Group of Tiles of same colour

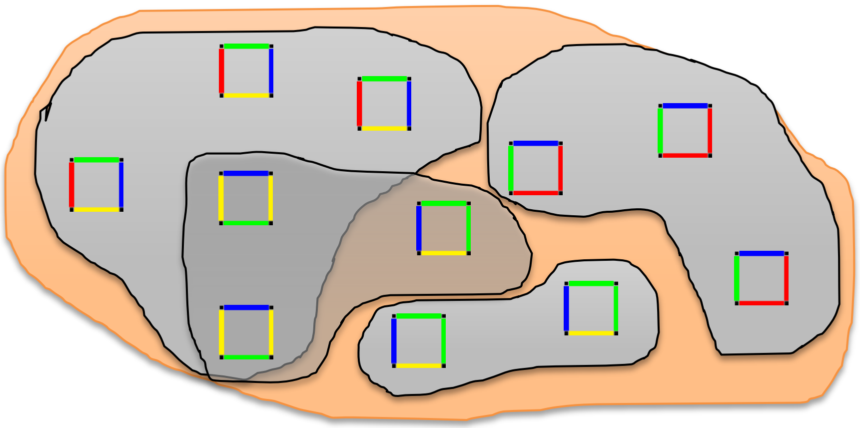


Can reduce from OWL entailment to Tiling

1. Each tile must have a domino type
   - Define tiles as a class *T*, a union of classes for each domino type:

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

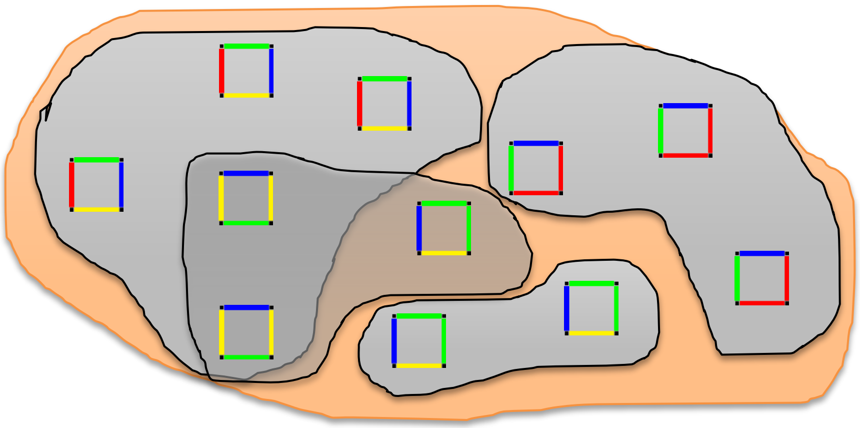
```
:T owl:equivalentClass
  [ owl:unionOf ( :D1 ... :Dk ) ] .
```



Can reduce from Tiling to Owl Entailment

1. Each tile must have a domino type

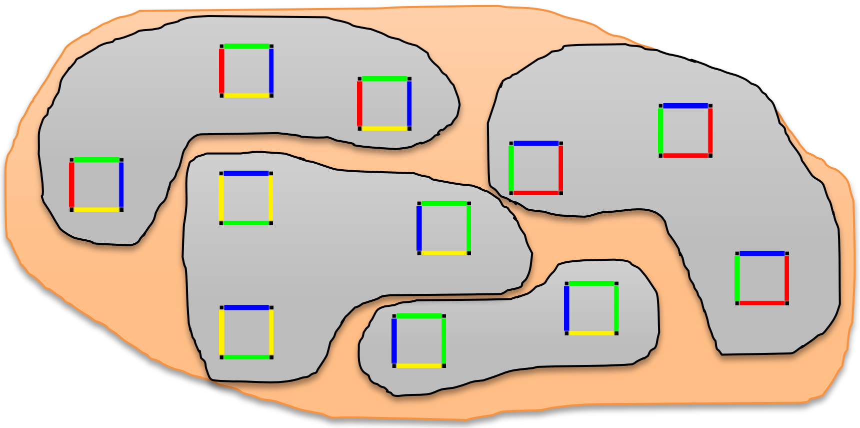
Now what else do we need to encode?



Can reduce from Tiling to OWL Entailment

1. Each tile must have a domino type
2. Each tile can only be one domino type

How can we encode this in OWL?

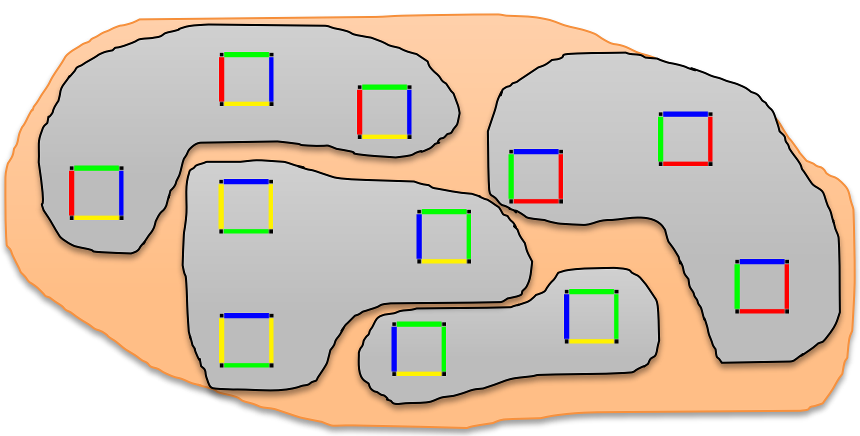


Can reduce from Tiling to OWL Entailment

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

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

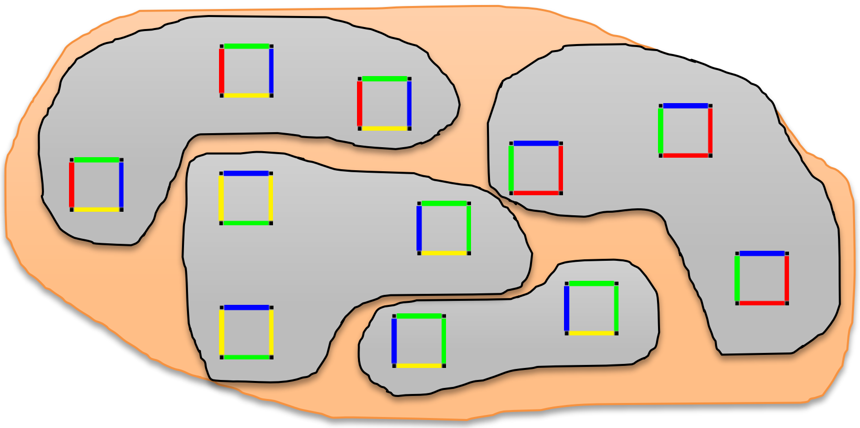
```
:T owl:equivalentClass    #covers 1 and 2!!
  [ owl:disjointUnionOf ( :D1 ... :Dk ) ] .
```



Can reduce from Tiling to Owl Entailment

1. Each tile must have a domino type
2. Each tile can only be one domino type

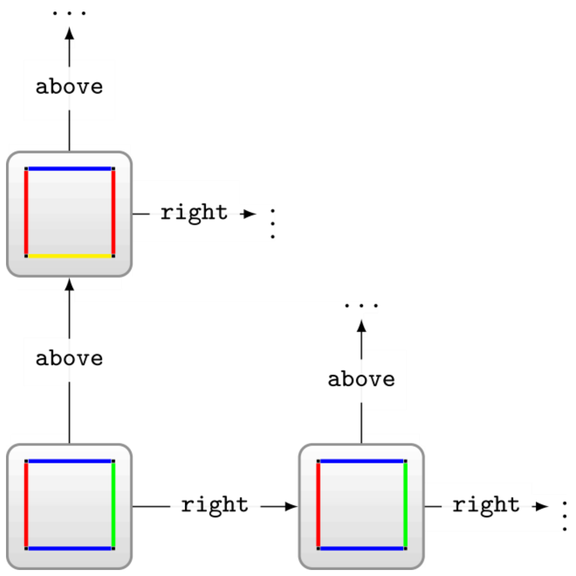
Now what else do we need to encode?



Can reduce from Tiling to OWL Entailment

1. Each tile must have a domino type
2. Each tile can only be one domino type
3. Each tile must have a tile to the right and above

How can we encode this in OWL?

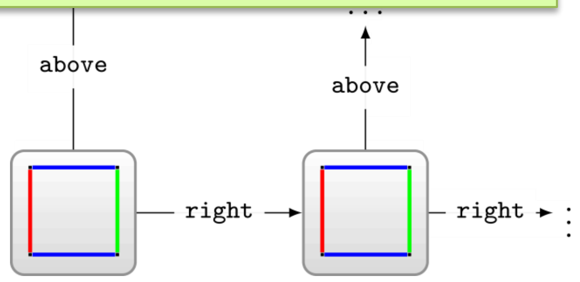


Can reduce from Tiling to OWL Entailment

1. Each tile must have a domino type
2. Each tile can only be one domino type
3. Each tile must have a tile to the right and above
   - Define that a tile has some values from tile for *r*ight/*a*bove:

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

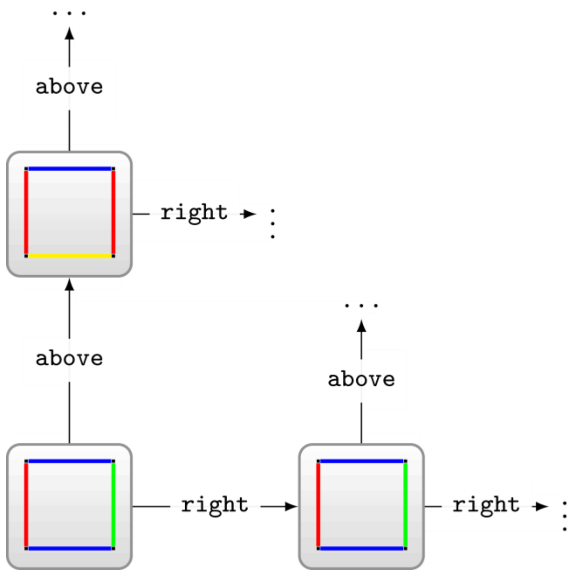
```
:T rdfs:subClassOf
  [ owl:intersectionOf (
      [ owl:someValuesFrom :T ; owl:onProperty :r ]
      [ owl:someValuesFrom :T ; owl:onProperty :a ]
    ) ] .
```



Can reduce from Tiling to Owl Entailment

1. Each tile must have a domino type
2. Each tile can only be one domino type
3. Each tile must have a tile to the right and above

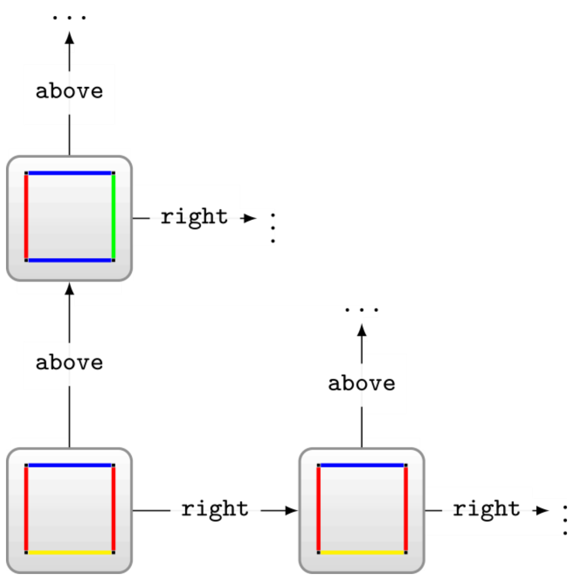
Are we there yet?



Can reduce from Tiling to OWL Entailment

1. Each tile must have a domino type
2. Each tile can only be one domino type
3. Each tile must have a tile to the right and above
4. Tiles to the right and tiles above must match colour

How can we encode this in OWL?



Can reduce from Tiling to OWL Entailment

1. Each tile must have a domino type
2. Each tile can only be one domino type
3. Each tile must have a tile to the right and above
4. Tiles to the right and tiles above must match colour

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

$$\ldots$$

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

Where:
$R(D_i)$ denotes all dominos that can be to the right of D_i
$A(D_i)$ denotes all dominos that can be above D_i

Can reduce from Tiling to Owl Entailment

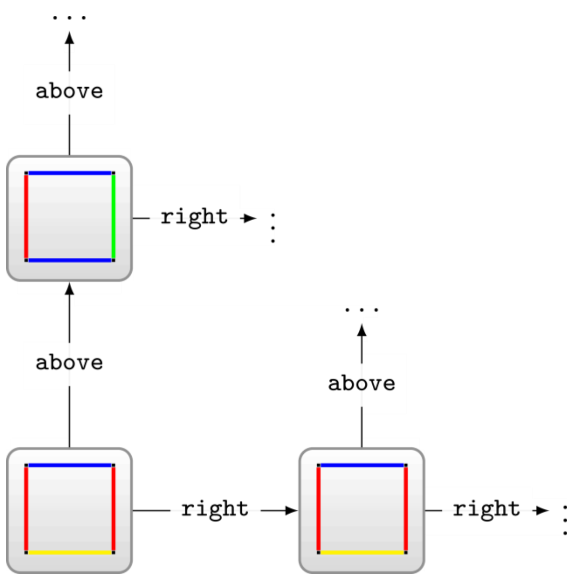
1. Each tile must have a domino type
2. Each tile can only be one domino type
3. Each tile must have a tile to the right and above
4. Tiles to the right and tiles above must match colour

```
# for 1 >= n >= k
:Dn owl:equivalentClass
  [ owl:intersectionOf
     ( [ a owl:Restriction ;
         owl:allValuesFrom [ owl:unionOf DnA ] ;
         owl:onProperty :above ]
       [ a owl:Restriction ;
         owl:allValuesFrom [ owl:unionOf DnR ] ;
         owl:onProperty :right ] ) ] .
  # DnA : list of dominos that can go above :Dn
  # DnR : list of dominos that can go right of :Dn
```

Can reduce from Tiling to OWL Entailment

1. Each tile must have a domino type
2. Each tile can only be one domino type
3. Each tile must have a tile to the right and above
4. Tiles to the right and tiles above must match colour

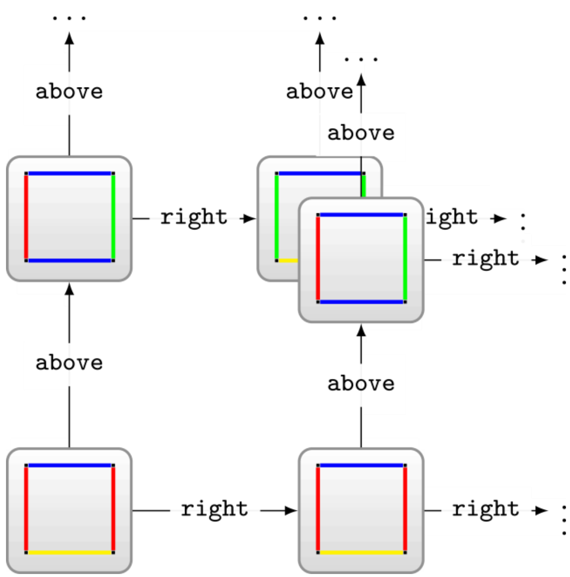
Are we there yet?



Can reduce from Tiling to Owl Entailment

1. Each tile must have a domino type
2. Each tile can only be one domino type
3. Each tile must have a tile to the right and above
4. Tiles to the right and tiles above must match colour

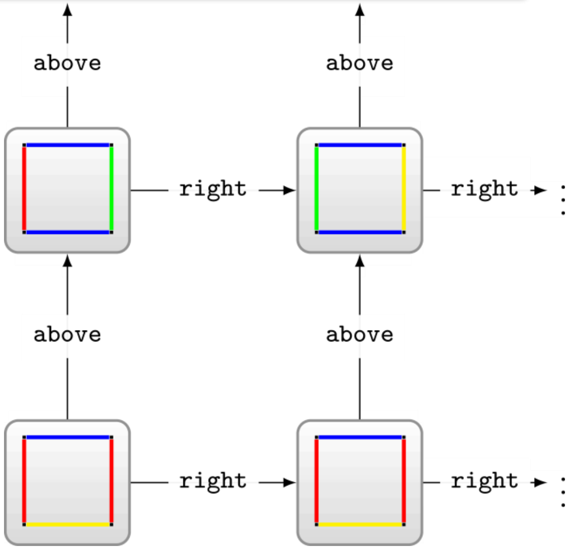
Are we there yet?



Can reduce from Tiling to OWL Entailment

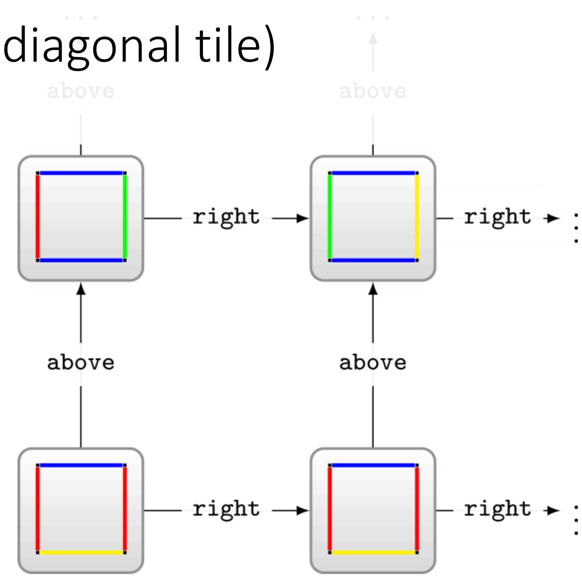
1. Each tile must have a domino type
2. Each tile can only be one domino type
3. Each tile must have a tile to the right and above
4. Tiles to the right and tiles above must match colour
5. Tile right then above = Tile above then right

How can we encode this in OWL?



Can reduce from Tiling to OWL Entailment

1. Each tile must have a domino type
2. Each tile can only be one domino type
3. Each tile must have a tile to the right and above
4. Tiles to the right and tiles above must match colour
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)

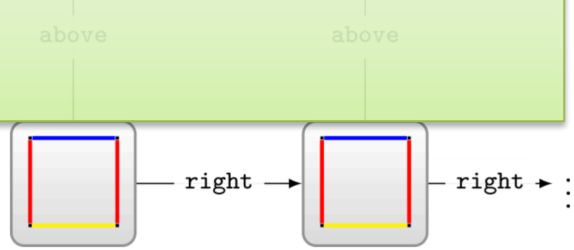


Can reduce from Tiling to Owl Entailment

1. Each tile must have a domino type
2. Each tile can only be one domino type
3. Each tile must have a tile to the right and above
4. Tiles to the right and tiles above must match colour
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)

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

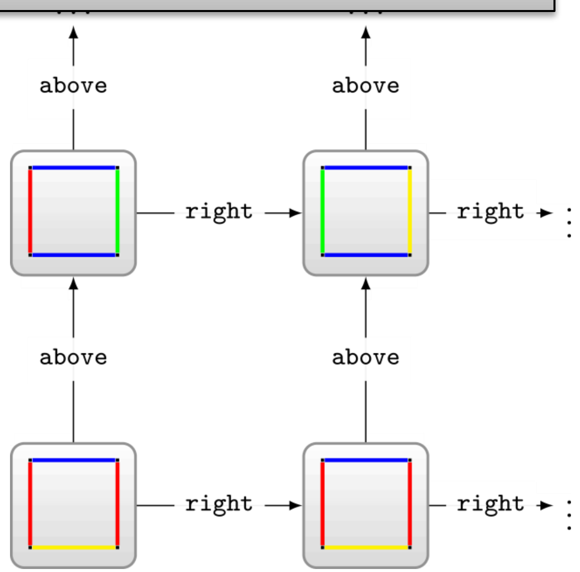
```
:d owl:propertyChainAxiom ( :a :r ) .
:d owl:propertyChainAxiom ( :r :a ) .
:d a owl:FunctionalProperty .
```



Can reduce from Tiling to Owl Entailment

1. Each tile must have a domino type
2. Each tile can only be one domino type
3. Each tile must have a tile to the right and above
4. Tiles to the right and tiles above must match colour
5. Tile right then above = Tile above then right

Are we there yet?



Can reduce from Tiling to Owl Entailment

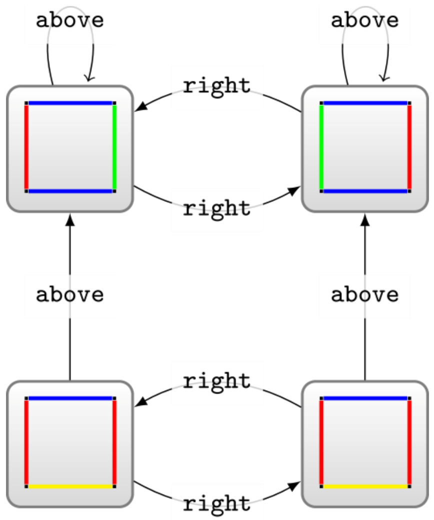
1. Each tile must have a domino type
2. Each tile can only be one domino type
3. Each tile must have a tile to the right and above
4. Tiles to the right and tiles above must match colour
5. Tile right then above = Tile above then right

Are we there yet?

We could have "cyclic" models:

We could remove such models by defining a sub-property of right/above to be transitive and asymmetric ...

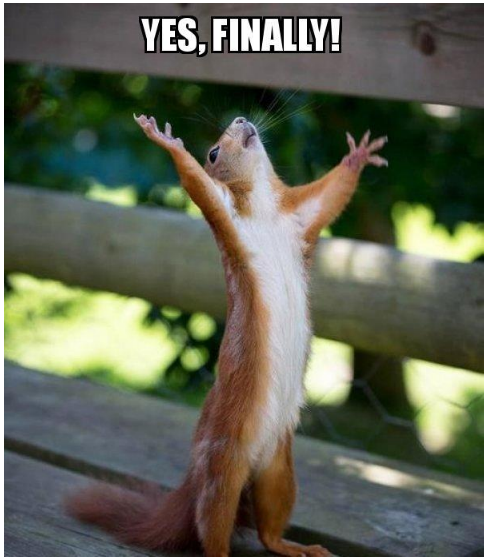
But actually such "cyclic" models can be "unravelled" into valid domino tilings so we don't need to!

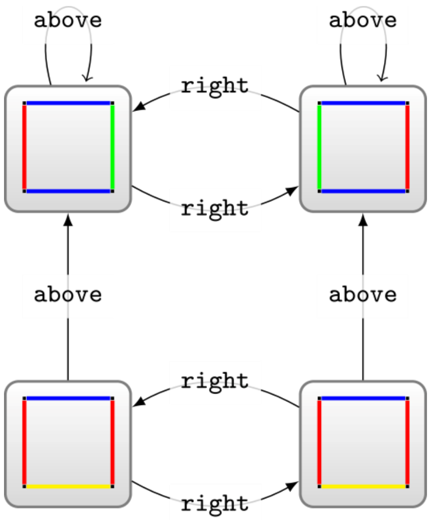


Can reduce from Tiling to Owl Entailment

1. Each tile must have a domino type
2. Each tile can only be one domino type
3. Each tile must have a tile to the right and above
4. Tiles to the right and tiles above must match colour
5. Tile right then above = Tile above then right

Are we there yet?





But what's the entailment question?

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

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

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

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

$$\ldots$$

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

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

???

What should we put in *O′*?

Goal: Ontology *O* entails *O′* if and only if *D* has no infinite tiling

But what's the entailment question?

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

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

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

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

$$\ldots$$

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

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

$$T \equiv \bot$$

Goal: Ontology *O* entails *O'* if and only if *D* has no infinite tiling

If *T* can have <u>any</u> member (a "tile"), it must have an infinite tiling!

If *T* can have <u>no</u> member, it must not have an infinite tiling.

COULD ALSO USE SATISFIABILITY ...

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

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

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

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

$$\ldots$$

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

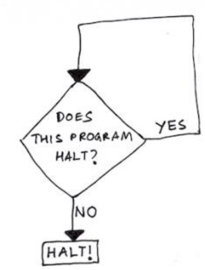
$$a \circ r \sqsubseteq d, \quad r \circ a \sqsubseteq d, \quad \mathsf{Func}(d)$$

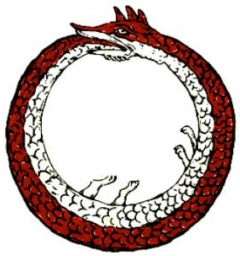
$$T(x)$$

Goal: Ontology *O* is satisfiable if and only if *D* has an infinite tiling

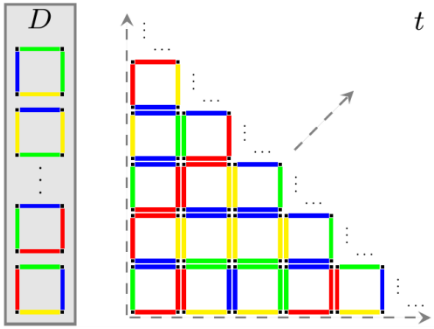
Here, *x* is an arbitrary fresh term

OWL entailment/satisfiability is undecidable!

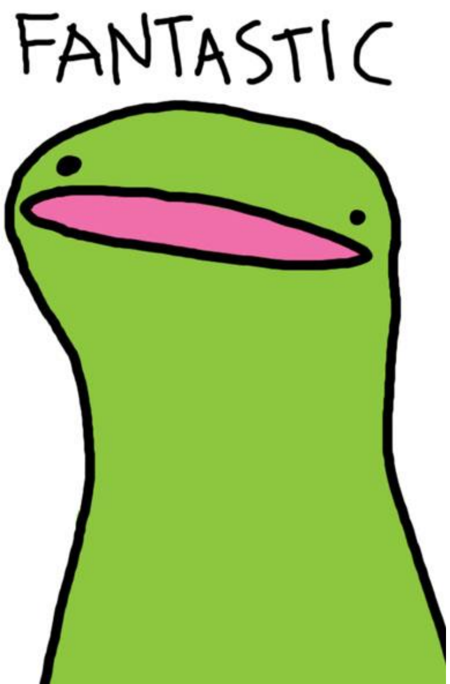


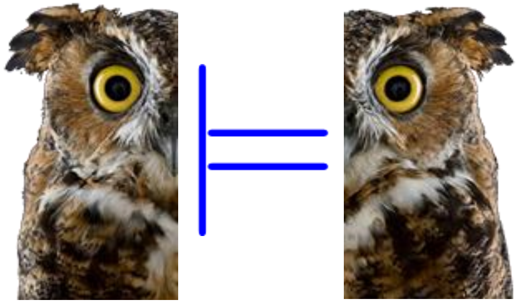


Reduction



Reduction





Not just OWL is undecidable ...

Knowledge representation:

Tell machines stuff about the world in a formalism they can (deductively) reason over using automated methods.

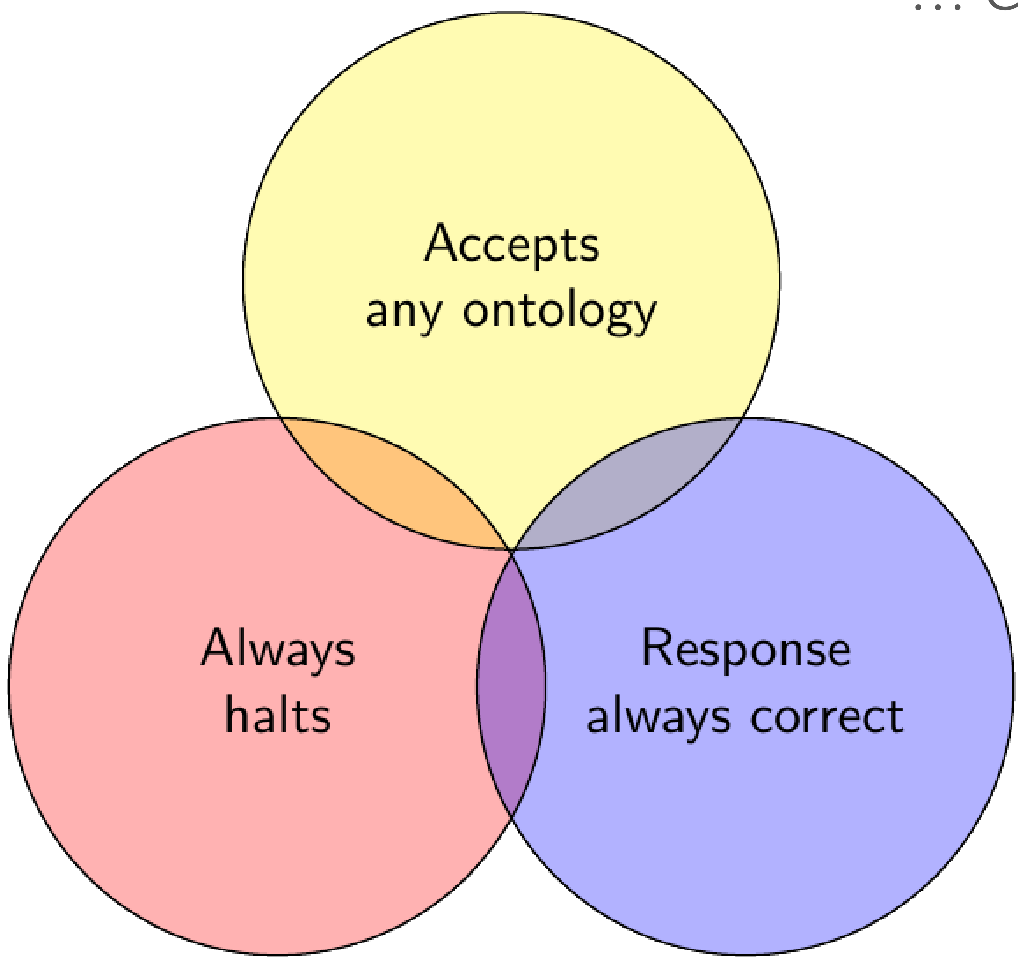
But if we tell them everything ...

Reasoning becomes undecidable!

OWL entailment/satisfiability is undecidable …

So what are we supposed to do now?

… CHOOSE TWO



OWL entailment/satisfiability is undecidable …

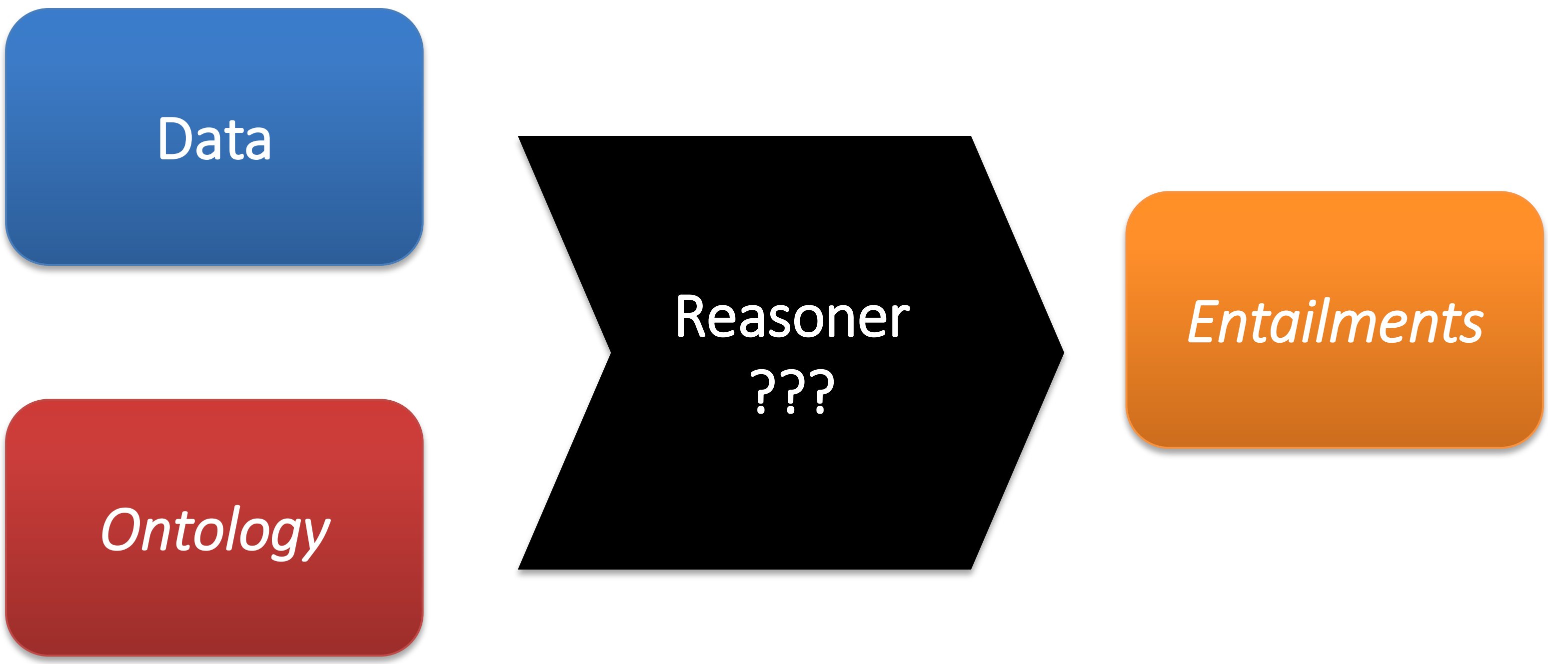
So what are we supposed to do now?

- Accept incomplete reasoners that halt
  - Complete language, incomplete reasoning, halts
- Accept complete reasoners that may not halt
  - Complete language, complete reasoning, may not halt
- Restrict OWL so reasoning becomes decidable
  - Restricted language, complete reasoning, halts

Incomplete reasoners

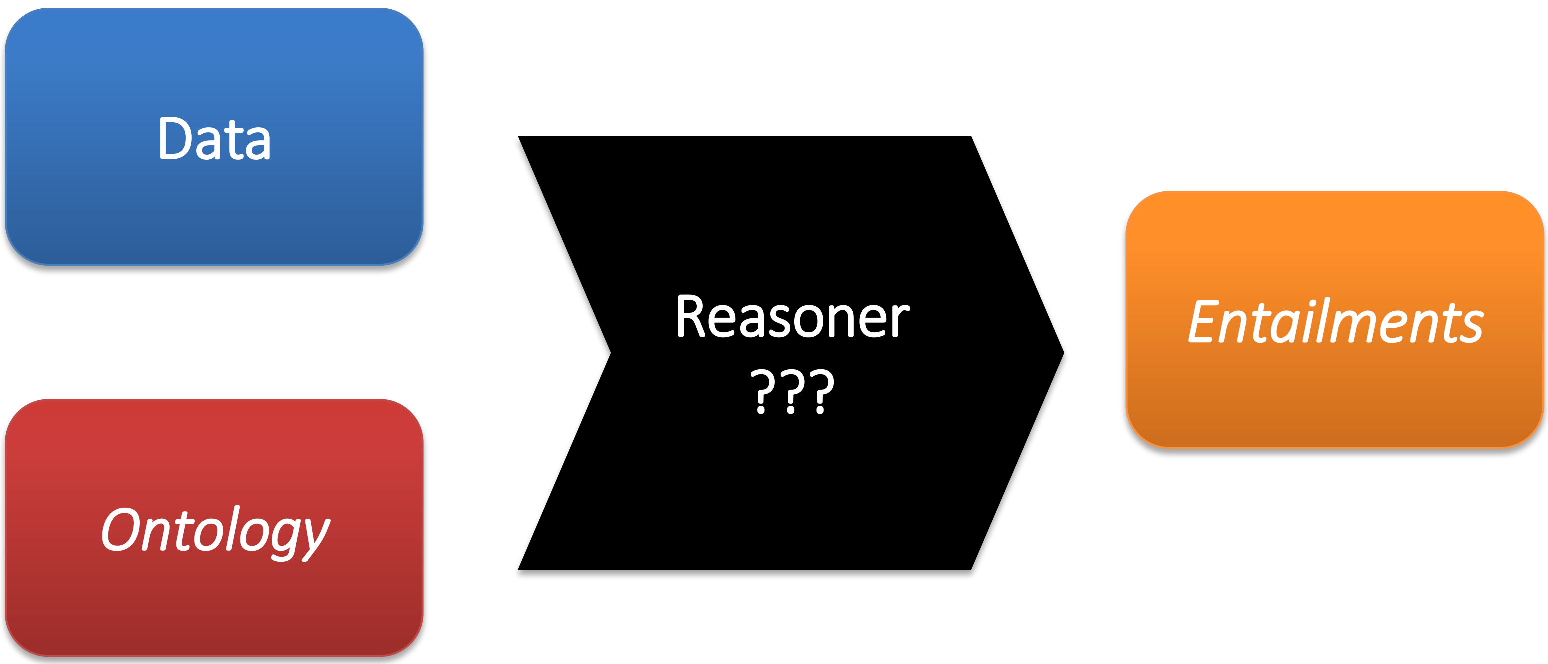
that halt

In the labs …



But what is the reasoner actually doing?

In the labs …



But what is the reasoner actually doing?

Incomplete materialisation using rules.

RECALL RULES FOR RDFS ...

ID	if G matches	then G RDFS$_D$-entails
rdfD1	?x ?p ?l . (?l a literal with datatype IRI dt(?l) $\in D$)	?x ?p _:b . _:b a dt(?l) .
rdfD2	?x ?p ?y .	?p a rdf:Property .
rdfs1	?u $\in D$	?u a rdfs:Datatype .
rdfs2	?p rdfs:domain ?c . ?x ?p ?y .	?x a ?c .
rdfs3	?p rdfs:range ?c . ?x ?p ?y .	?y a ?c .
rdfs4a	?x ?p ?y .	?x a rdfs:Resource .
rdfs4b	?x ?p ?y .	?y a rdfs:Resource .
rdfs5	?p rdfs:subPropertyOf ?q . ?x ?p ?y .	?x ?q ?y .
rdfs6	?p a rdf:Property .	?p rdfs:subPropertyOf ?p .
rdfs7	?p rdfs:subPropertyOf ?q . ?q rdfs:subPropertyOf ?r .	?p rdfs:subPropertyOf ?r .
rdfs8	?c a rdfs:Class .	?c rdfs:subClassOf rdfs:Resource .
rdfs9	?c rdfs:subClassOf ?d . ?x a ?c .	?x a ?d .
rdfs10	?c a rdfs:Class .	?c rdfs:subClassOf ?c .
rdfs11	?c rdfs:subClassOf ?d . ?d rdfs:subClassOf ?e .	?c rdfs:subClassOf ?e .
rdfs12	?p a rdfs:ContainerMembershipProperty .	?p rdfs:subPropertyOf rdfs:member .
rdfs13	?d a rdfs:Datatype .	?d rdfs:subClassOf rdf:Literal .

Now we need rules to cover (some of) OWL ...

Standard set of OWL rules: OWL 2 RL/RDF

https://www.w3.org/TR/owl2-profiles/#Reasoning_in_OWL_2_RL_and_RDF_Graphs_using_Rules

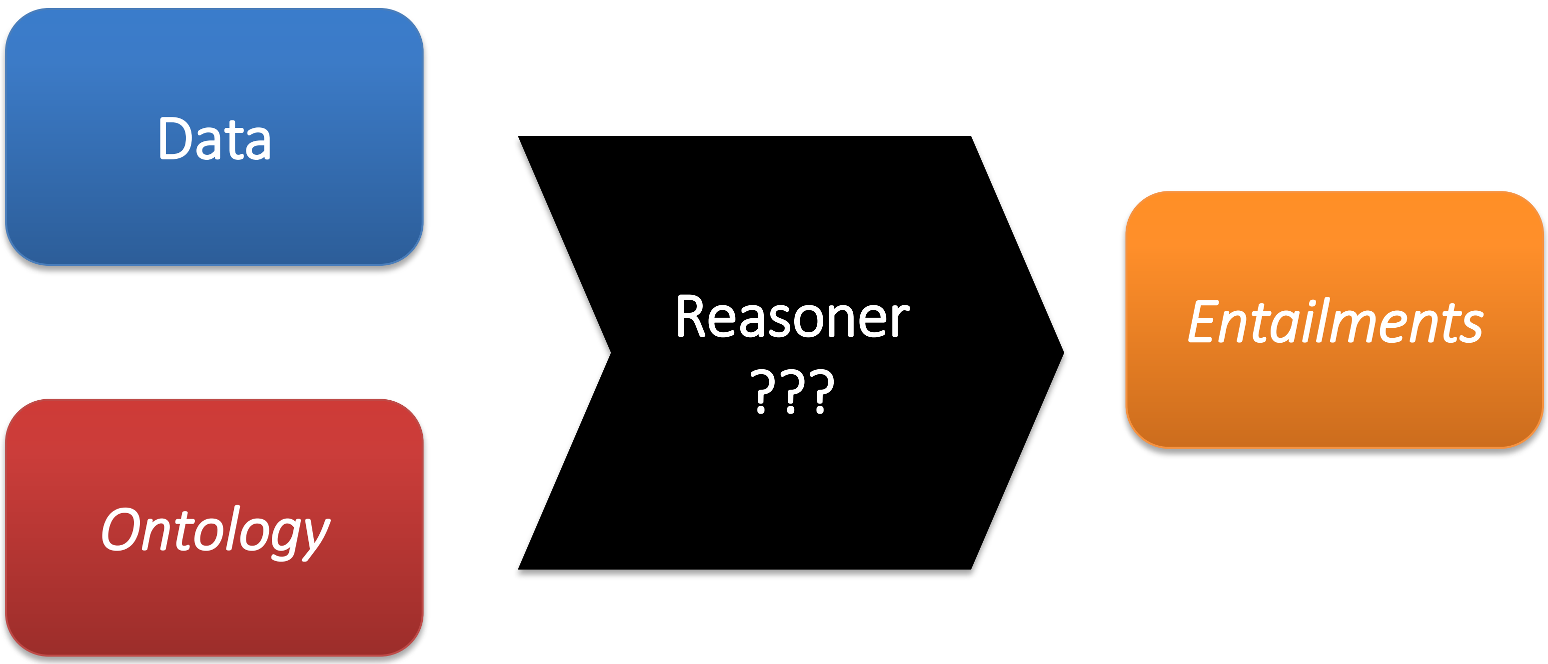
W3C Recommendation

OWL 2 Web Ontology Language Profiles (Second Edition)

W3C Recommendation 11 December 2012

This version:
http://www.w3.org/TR/2012/REC-owl2-profiles-20121211/
Latest version (series 2):
http://www.w3.org/TR/owl2-profiles/
Latest Recommendation:
http://www.w3.org/TR/owl-profiles
Previous version:
http://www.w3.org/TR/2012/PER-owl2-profiles-20121018/

In the labs ...



But what is the reasoner actually doing?

Incomplete materialisation using RDFS & OWL 2 RL/RDF rules.

OWL 2 RL/RDF rule examples: Equality

ID	if G matches	then G OWL-entails
EQ-REF	?s ?p ?o .	?s owl:sameAs ?s . ?p owl:sameAs ?p . ?o owl:sameAs ?o .
EQ-SYM	?x owl:sameAs ?y .	?y owl:sameAs ?x .
EQ-TRANS	?x owl:sameAs ?y . ?y owl:sameAs ?z .	?x owl:sameAs ?z .
EQ-REP-S	?s owl:sameAs ?s′ . ?s ?p ?o .	???
EQ-REP-P	?p owl:sameAs ?p′ . ?s ?p ?o .	?s ?p′ ?o .
EQ-REP-O	?o owl:sameAs ?o′ . ?s ?p ?o .	?s ?p ?o′ .
EQ-DIFF1	?x owl:sameAs ?y ; owl:differentFrom ?y .	???
...	...	...

OWL 2 RL/RDF rule examples: Equality

ID	if G matches	then G OWL-entails

EQ-REF	`?s ?p ?o .`	`?s owl:sameAs ?s .`
		`?p owl:sameAs ?p .`
		`?o owl:sameAs ?o .`

EQ-SYM	`?x owl:sameAs ?y .`	`?y owl:sameAs ?x .`
EQ-TRANS	`?x owl:sameAs ?y . ?y owl:sameAs ?z .`	`?x owl:sameAs ?z .`
EQ-REP-S	`?s owl:sameAs ?s′ . ?s ?p ?o .`	`?s′ ?p ?o .`
EQ-REP-P	`?p owl:sameAs ?p′ . ?s ?p ?o .`	`?s ?p′ ?o .`
EQ-REP-O	`?o owl:sameAs ?o′ . ?s ?p ?o .`	`?s ?p ?o′ .`
EQ-DIFF1	`?x owl:sameAs ?y ; owl:differentFrom ?y .`	FALSE
...	...	...

OWL 2 RL/RDF rule examples: Properties

ID	if G matches	then G OWL-entails
PRP-DOM	?p rdfs:domain ?c . ?x ?p ?y .	?x a ?c .
PRP-RNG	?p rdfs:range ?c . ?x ?p ?y .	?y a ?c .
PRP-FP	?p a owl:FunctionalProperty . ?x ?p ?y_1 . ?x ?p ?y_2 .	???
PRP-IFP	?p a owl:InverseFunctionalProperty . ?x_1 ?p ?y . ?x_2 ?p ?y .	?x_1 owl:sameAs ?x_2 .
PRP-IRP	?p a owl:IrreflexiveProperty . ?x ?p ?x .	FALSE
PRP-SYMP	?p a owl:SymmetricProperty . ?x ?p ?y .	?y ?p ?x .
PRP-ASYP	?p a owl:AsymmetricProperty . ?x ?p ?y . ?y ?p ?x .	FALSE
PRP-TRP	?p a owl:TransitiveProperty . ?x ?p ?y . ?y ?p ?z .	???
PRP-SPO1	?p_1 rdfs:subPropertyOf ?p_2 . ?x ?p_1 ?y .	?x ?p_1 ?z .
PRP-SPO2	?p owl:propertyChainAxiom (?p_1 ... ?p_n) . ?a_1 ?p_1 ?a_2 . ... ?a_n ?p_n ?a_{n+1} .	?a_1 ?p ?a_{n+1} .
PRP-EQP1	?p_1 owl:equivalentProperty ?p_2 . ?x ?p_1 ?y .	?x ?p_2 ?y .
PRP-EQP2	?p_1 owl:equivalentProperty ?p_2 . ?x ?p_2 ?y .	?x ?p_1 ?y .
PRP-PDW	?p_1 owl:propertyDisjointWith ?p_2 . ?x ?p_1 ?y . ?x ?p_2 ?y .	FALSE
PRP-INV1	?p_1 owl:inverseOf ?p_2 . ?x ?p_1 ?y .	???
PRP-INV2	?p_1 owl:inverseOf ?p_2 . ?y ?p_2 ?x .	?x ?p_1 ?y .
PRP-KEY	?c owl:hasKey (?p_1 ... ?p_n) . ?x a ?c ; ?p_1 ?z_1 ; ... ; ?p_n ?z_n . ?y a ?c ; ?p_1 ?z_1 ; ... ; ?p_n ?z_n .	?x owl:sameAs ?y .
...	...	...

OWL 2 RL/RDF rule examples: Properties

ID	if G matches	then G OWL-entails
PRP-DOM	`?p rdfs:domain ?c . ?x ?p ?y .`	`?x a ?c .`
PRP-RNG	`?p rdfs:range ?c . ?x ?p ?y .`	`?y a ?c .`
PRP-FP	`?p a owl:FunctionalProperty . ?x ?p ?y`$_1$ `. ?x ?p ?y`$_2$ `.`	`?y`$_1$ `owl:sameAs ?y`$_2$ `.`
PRP-IFP	`?p a owl:InverseFunctionalProperty . ?x`$_1$ `?p ?y . ?x`$_2$ `?p ?y .`	`?x`$_1$ `owl:sameAs ?x`$_2$ `.`
PRP-IRP	`?p a owl:IrreflexiveProperty . ?x ?p ?x .`	FALSE
PRP-SYMP	`?p a owl:SymmetricProperty . ?x ?p ?y .`	`?y ?p ?x .`
PRP-ASYP	`?p a owl:AsymmetricProperty . ?x ?p ?y . ?y ?p ?x .`	FALSE
PRP-TRP	`?p a owl:TransitiveProperty . ?x ?p ?y . ?y ?p ?z .`	`?x ?p ?z .`
PRP-SPO1	`?p`$_1$ `rdfs:subPropertyOf ?p`$_2$ `. ?x ?p`$_1$ `?y .`	`?x ?p`$_1$ `?z .`
PRP-SPO2	`?p owl:propertyChainAxiom (?p`$_1$ `... ?p`$_n$`) .` `?a`$_1$ `?p`$_1$ `?a`$_2$ `. ... ?a`$_n$ `?p`$_n$ `?a`$_{n+1}$ `.`	`?a`$_1$ `?p ?a`$_{n+1}$ `.`
PRP-EQP1	`?p`$_1$ `owl:equivalentProperty ?p`$_2$ `. ?x ?p`$_1$ `?y .`	`?x ?p`$_2$ `?y .`
PRP-EQP2	`?p`$_1$ `owl:equivalentProperty ?p`$_2$ `. ?x ?p`$_2$ `?y .`	`?x ?p`$_1$ `?y .`
PRP-PDW	`?p`$_1$ `owl:propertyDisjointWith ?p`$_2$ `. ?x ?p`$_1$ `?y . ?x ?p`$_2$ `?y .`	FALSE
PRP-INV1	`?p`$_1$ `owl:inverseOf ?p`$_2$ `. ?x ?p`$_1$ `?y .`	`?y ?p`$_2$ `?x .`
PRP-INV2	`?p`$_1$ `owl:inverseOf ?p`$_2$ `. ?y ?p`$_2$ `?x .`	`?x ?p`$_1$ `?y .`

PRP-KEY	`?c owl:hasKey (?p`$_1$ `... ?p`$_n$`) .`
	`?x a ?c ; ?p`$_1$ `?z`$_1$ `; ... ; ?p`$_n$ `?z`$_n$ `.`
	`?y a ?c ; ?p`$_1$ `?z`$_1$ `; ... ; ?p`$_n$ `?z`$_n$ `.`
		`?x owl:sameAs ?y .`

...	...	...

OWL 2 RL/RDF RULE EXAMPLES: CLASSES

ID	if G matches	then G OWL-entails
CAX-SCO	?c_1 rdfs:subClassOf ?c_2 . ?x a ?c_1 .	?x a ?c_2 .
CAX-EQC1	?c_1 owl:equivalentClass ?c_2 . ?x a ?c_1 .	?x a ?c_2 .
CAX-EQC2	?c_1 owl:equivalentClass ?c_2 . ?x a ?c_2 .	?x a ?c_1 .
CAX-DW	?c_1 owl:disjointWith ?c_2 . ?x a ?c_1 , ?c_2 .	FALSE
CLS-INT1	?c owl:intersectionOf (?c_1 ... ?c_n) . ?y a ?c_1 , ..., ?c_n .	???
CLS-INT2	?c owl:intersectionOf (?c_1 ... ?c_n) . ?y a ?c .	?y a ?c_1 , ..., ?c_n .
CLS-UNI	?c owl:unionOf (?c_1 ... ?c_n) . ?y a ?c_i . $(1 \leq i \leq n)$	???
CLS-COM	?c_1 owl:complementOf ?c2 . ?x a ?c_1 , ?c_2 .	FALSE
CLS-SVF1	?x owl:someValuesFrom ?y ; owl:onProperty ?p . ?u ?p ?v . ?v a ?y .	?u a ?x .
CLS-SVF2	?x owl:someValuesFrom owl:Thing ; owl:onProperty ?p . ?u ?p ?v .	?u a ?x .
CLS-AVF	?x owl:allValuesFrom ?y ; owl:onProperty ?p . ?u ?p ?v ; a ?x .	???
CLS-HV1	?x owl:hasValue ?y ; owl:onProperty ?p . ?u a ?x .	?u ?p ?y .
CLS-HV2	?x owl:hasValue ?y ; owl:onProperty ?p . ?u ?p ?y .	?u a ?x .
CLS-MAXC1	?x owl:maxCardinality 0 ; owl:onProperty ?p . ?u a ?x ; ?p ?y .	FALSE
CLS-MAXC2	?x owl:maxCardinality 1 ; owl:onProperty ?p . ?u a ?x ; ?p ?y_1 , ?y_2 .	?y_1 owl:sameAs ?y_2 .
CLS-MAXQC1	?x owl:maxQualifiedCardinality 0 ; owl:onProperty ?p ; owl:onClass ?c . ?u a ?x ; ?p ?y . ?y a ?c .	FALSE
CLS-MAXQC3	?x owl:maxQualifiedCardinality 1 ; owl:onProperty ?p ; owl:onClass ?c . ?u a ?x ; ?p ?y_1 , ?y_2 . ?y_1 a ?c . ?y_2 a ?c .	?y_1 owl:sameAs ?y_2 .
CLS-OO	?c owl:oneOf (?y_1 ... ?y_n) .	?y_1 a ?c . ... ?y_n a ?c .
...	...	...

OWL 2 RL/RDF RULE EXAMPLES: CLASSES

ID	if G matches	then G OWL-entails
CAX-SCO	?c_1 rdfs:subClassOf ?c_2 . ?x a ?c_1 .	?x a ?c_2 .
CAX-EQC1	?c_1 owl:equivalentClass ?c_2 . ?x a ?c_1 .	?x a ?c_2 .
CAX-EQC2	?c_1 owl:equivalentClass ?c_2 . ?x a ?c_2 .	?x a ?c_1 .
CAX-DW	?c_1 owl:disjointWith ?c_2 . ?x a ?c_1 , ?c_2 .	FALSE
CLS-INT1	?c owl:intersectionOf (?c_1 ... ?c_n) . ?y a ?c_1 , ..., ?c_n .	?y a ?c .
CLS-INT2	?c owl:intersectionOf (?c_1 ... ?c_n) . ?y a ?c .	?y a ?c_1 , ..., ?c_n .
CLS-UNI	?c owl:unionOf (?c_1 ... ?c_n) . ?y a ?c_i . $(1 \leq i \leq n)$	?y a ?c .
CLS-COM	?c_1 owl:complementOf ?c2 . ?x a ?c_1 , ?c_2 .	FALSE
CLS-SVF1	?x owl:someValuesFrom ?y ; owl:onProperty ?p . ?u ?p ?v . ?v a ?y .	?u a ?x .
CLS-SVF2	?x owl:someValuesFrom owl:Thing ; owl:onProperty ?p . ?u ?p ?v .	?u a ?x .
CLS-AVF	?x owl:allValuesFrom ?y ; owl:onProperty ?p . ?u ?p ?v ; a ?x .	?v a ?y .
CLS-HV1	?x owl:hasValue ?y ; owl:onProperty ?p . ?u a ?x .	?u ?p ?y .
CLS-HV2	?x owl:hasValue ?y ; owl:onProperty ?p . ?u ?p ?y .	?u a ?x .
CLS-MAXC1	?x owl:maxCardinality 0 ; owl:onProperty ?p . ?u a ?x ; ?p ?y .	FALSE
CLS-MAXC2	?x owl:maxCardinality 1 ; owl:onProperty ?p . ?u a ?x ; ?p ?y_1 , ?y_2 .	?y_1 owl:sameAs ?y_2 .
CLS-MAXQC1	?x owl:maxQualifiedCardinality 0 ; owl:onProperty ?p ; owl:onClass ?c . ?u a ?x ; ?p ?y . ?y a ?c .	FALSE
CLS-MAXQC3	?x owl:maxQualifiedCardinality 1 ; owl:onProperty ?p ; owl:onClass ?c . ?u a ?x ; ?p ?y_1 , ?y_2 . ?y_1 a ?c . ?y_2 a ?c .	?y_1 owl:sameAs ?y_2 .
CLS-OO	?c owl:oneOf (?y_1 ... ?y_n) .	?y_1 a ?c . ... ?y_n a ?c .
...	...	...

OWL 2 RL/RDF RULE EXAMPLES: SCHEMA

ID	if G matches	then G OWL-entails
SCM-SCO	?c$_1$ rdfs:subClassOf ?c$_2$. ?c$_2$ rdfs:subClassOf ?c$_3$.	???
SCM-EQC1	?c$_1$ owl:equivalentClass ?c$_2$.	?c$_1$ rdfs:subClassOf ?c$_2$. ?c$_2$ rdfs:subClassOf ?c$_1$.
SCM-EQC2	?c$_1$ rdfs:subClassOf ?c$_2$. ?c$_2$ rdfs:subClassOf ?c$_1$.	???
SCM-SPO	?p$_1$ rdfs:subPropertyOf ?p$_2$. ?p$_2$ rdfs:subPropertyOf ?p$_3$.	?p$_1$ rdfs:subPropertyOf ?p$_3$.
SCM-EQP1	?p$_1$ owl:equivalentProperty ?p$_2$.	?p$_1$ rdfs:subPropertyOf ?p$_2$. ?p$_2$ rdfs:subPropertyOf ?p$_1$.
SCM-EQP2	?p$_1$ rdfs:subPropertyOf ?p$_2$. ?p$_2$ rdfs:subPropertyOf ?p$_1$.	?p$_1$ owl:equivalentProperty ?p$_2$.
SCM-DOM1	?p rdfs:domain ?c$_1$. ?c$_1$ rdfs:subClassOf ?c$_2$.	???
SCM-DOM2	?p$_2$ rdfs:domain ?c . ?p$_1$ rdfs:subPropertyOf ?p$_2$.	?p$_1$ rdfs:range ?c .
SCM-RNG1	?p rdfs:range ?c$_1$. ?c$_1$ rdfs:subClassOf ?c$_2$.	?p rdfs:domain ?c$_2$.
SCM-RNG2	?p$_2$ rdfs:range ?c . ?p$_1$ rdfs:subPropertyOf ?p$_2$.	?p$_1$ rdfs:range ?c .
SCM-INT	?c owl:intersectionOf (?c$_1$ ... ?c$_n$) .	?c rdfs:subClassOf ?c$_1$, ..., ?c$_n$.
SCM-UNI	?c owl:unionOf (?c$_1$ ... ?c$_n$) .	?c$_1$ rdfs:subClassOf ?c . ... ?c$_n$ rdfs:subClassOf ?c .
...	...	...

OWL 2 RL/RDF rule examples: Missing

ID	if G matches	then G OWL-entails
—	?p a owl:ReflexiveProperty . ?x a owl:Thing .	???
…	…	…
—	?x owl:hasSelf true ; owl:onProperty ?p . ?u a ?x .	???
—	?x owl:hasSelf true ; owl:onProperty ?p . ?u ?p ?u .	???
…	…	…
—	?x owl:inverseOf ?x .	???
—	?x owl:inverseOf ?y . ?y owl:inverseOf ?z .	???
—	?x owl:inverseOf ?y . ?y a owl:FunctionalProperty .	???
—	?x owl:complementOf ?y , ?z .	???
…	…	…

OWL 2 RL/RDF rule examples: Missing

ID	if G matches	then G OWL-entails
—	`?p a owl:ReflexiveProperty . ?x a owl:Thing .`	`?x ?p ?x .`
…	…	…
—	`?x owl:hasSelf true ; owl:onProperty ?p . ?u a ?x .`	`?u ?p ?u .`
—	`?x owl:hasSelf true ; owl:onProperty ?p . ?u ?p ?u .`	`?u a ?x .`
…	…	…
—	`?x owl:inverseOf ?x .`	`?x a owl:SymmetricProperty .`
—	`?x owl:inverseOf ?y . ?y owl:inverseOf ?z .`	`?x owl:equivalentProperty ?z .`
—	`?x owl:inverseOf ?y . ?y a owl:FunctionalProperty .`	`?x a owl:InverseFunctionalProperty .`
—	`?x owl:complementOf ?y , ?z .`	`?y owl:equivalentClass ?z .`
…	…	…

Full list of OWL 2 RL/RDF rules (or see the book)

https://www.w3.org/TR/owl2-profiles/#Reasoning_in_OWL_2_RL_and_RDF_Graphs_using_Rules

W3C Recommendation

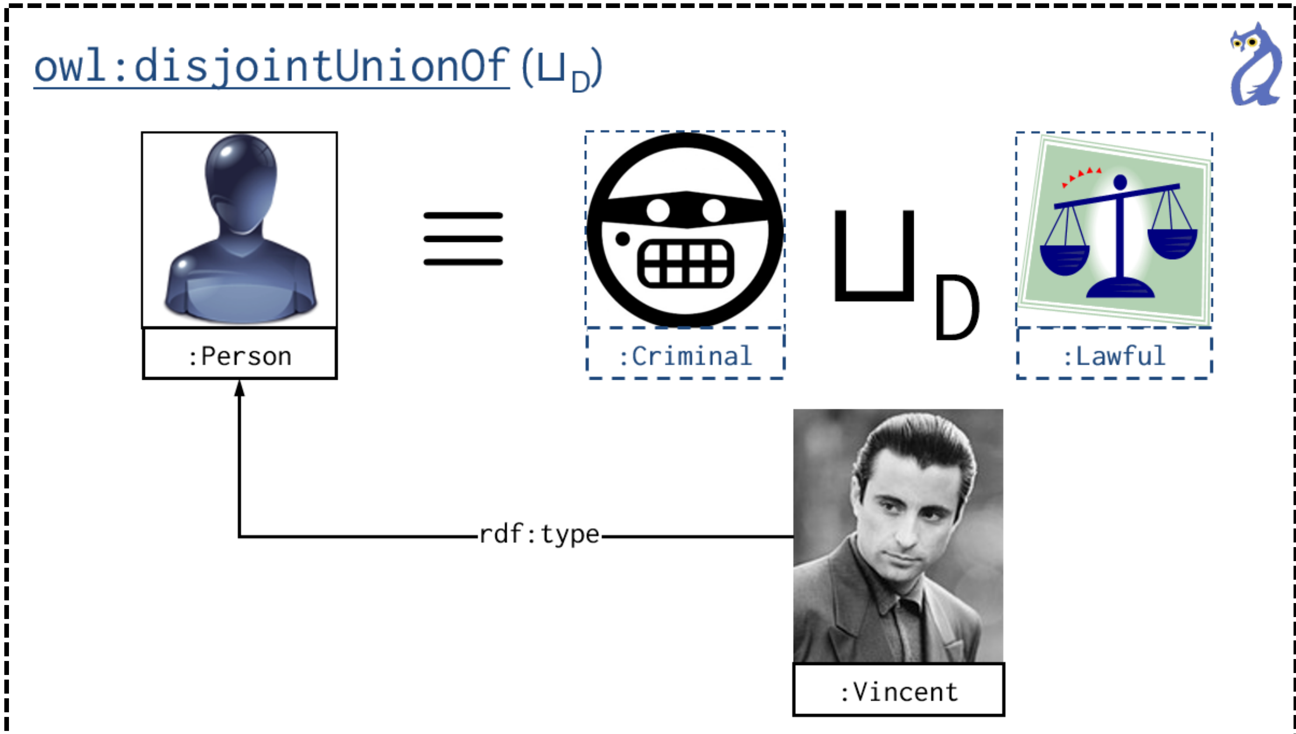
W3C®

OWL 2 Web Ontology Language Profiles (Second Edition)

W3C Recommendation 11 December 2012

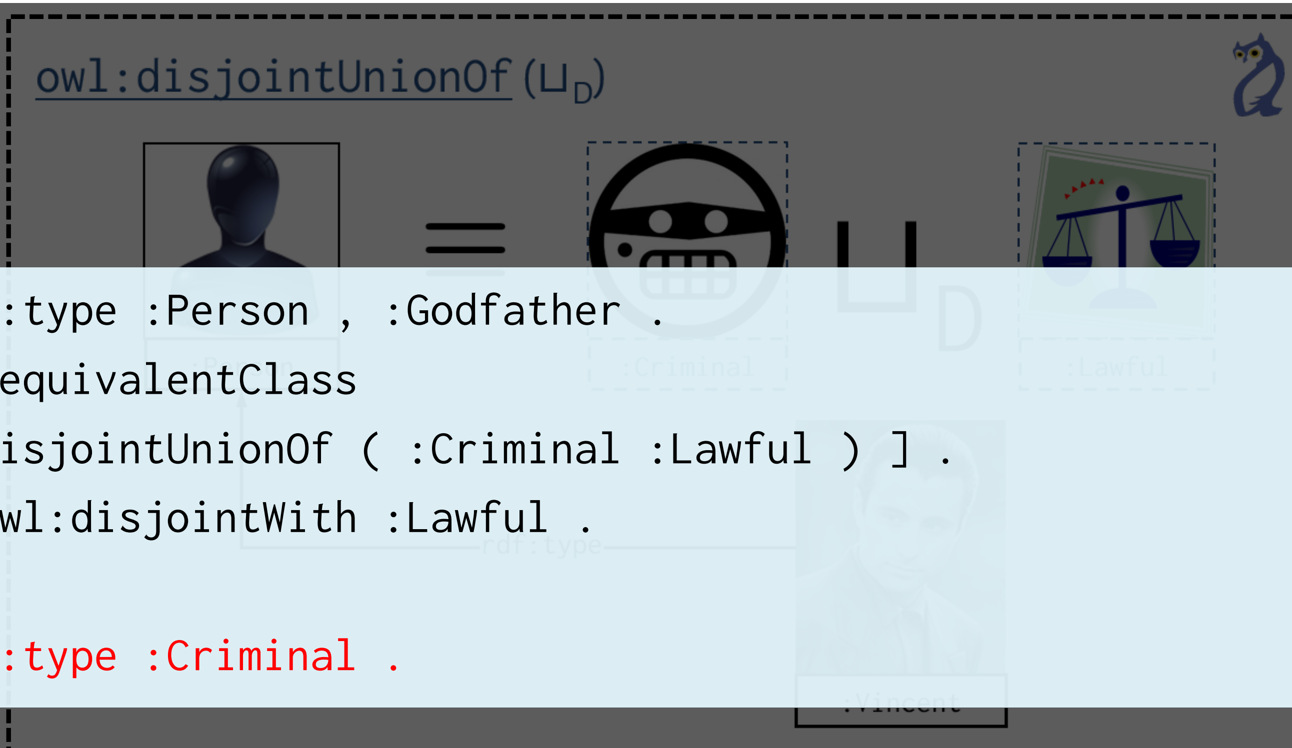
This version:
 http://www.w3.org/TR/2012/REC-owl2-profiles-20121211/
Latest version (series 2):
 http://www.w3.org/TR/owl2-profiles/
Latest Recommendation:
 http://www.w3.org/TR/owl-profiles
Previous version:
 http://www.w3.org/TR/2012/PER-owl2-profiles-20121018/

How is OWL2RL/RDF incomplete?




```
:Vincent rdf:type :Person .
:Person owl:equivalentClass
     [ owl:disjointUnionOf ( :Criminal :Lawful ) ]
```

How is OWL2RL/RDF incomplete? Disjunction

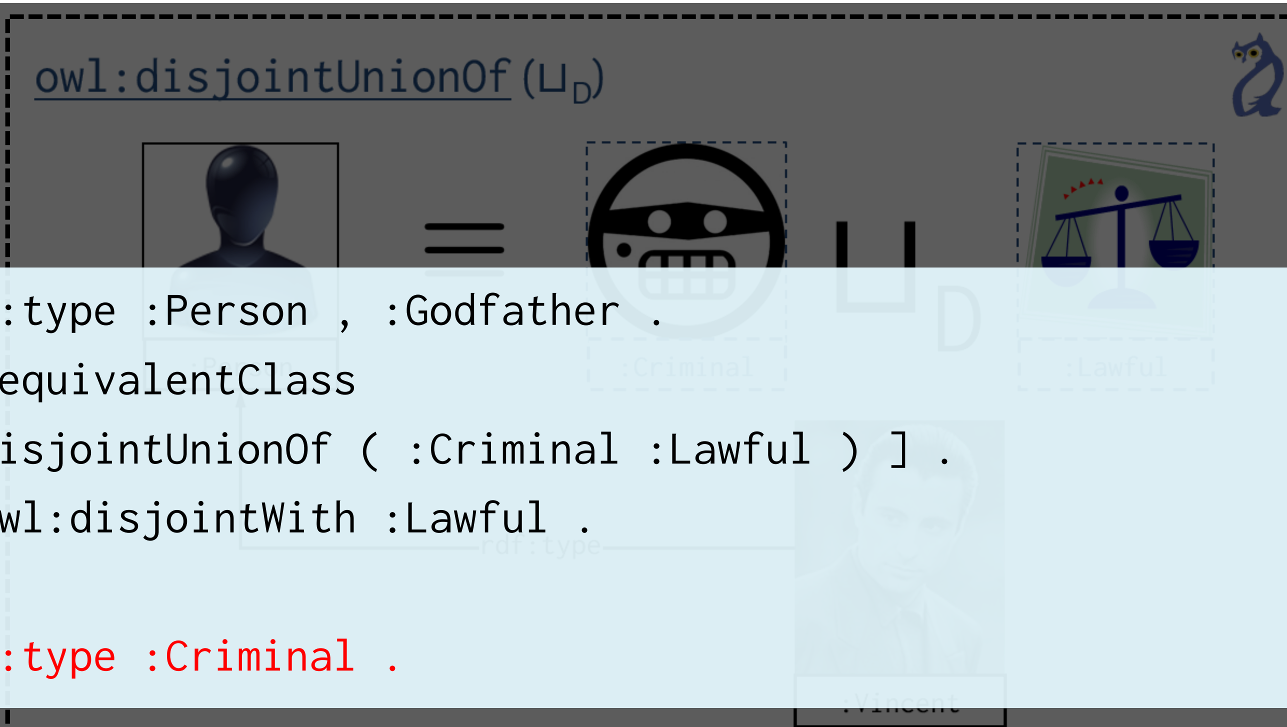



```
:Vincent rdf:type :Person , :Godfather .
:Person owl:equivalentClass
     [ owl:disjointUnionOf ( :Criminal :Lawful ) ] .
:Godfather owl:disjointWith :Lawful .
⇒
:Vincent rdf:type :Criminal .
```

```
:Vincent rdf:type :Person .
:Person owl:equivalentClass
     [ owl:disjointUnionOf ( :Criminal :Lawful ) ]
```

OWL 2 RL/RDF rules will miss this valid inference ...
Misses the information that Vincent is Criminal **OR** Lawful!

How is OWL2RL/RDF incomplete? Negation




```
:Vincent rdf:type :Person , :Godfather .
:Person owl:equivalentClass
     [ owl:disjointUnionOf ( :Criminal :Lawful ) ] .
:Godfather owl:disjointWith :Lawful .
⇒
:Vincent rdf:type :Criminal .
```

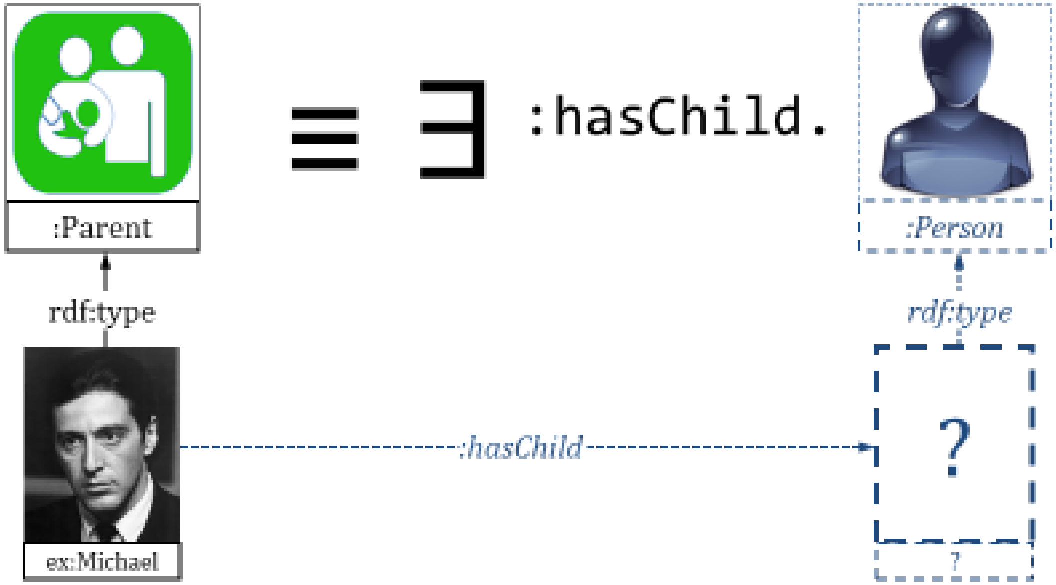
```
:Vincent rdf:type :Person .
:Person owl:equivalentClass
     [ owl:disjointUnionOf ( :Criminal :Lawful ) ]
```

OWL 2 RL/RDF rules will miss this valid inference ...

Misses the information that Vincent is **NOT** Lawful!

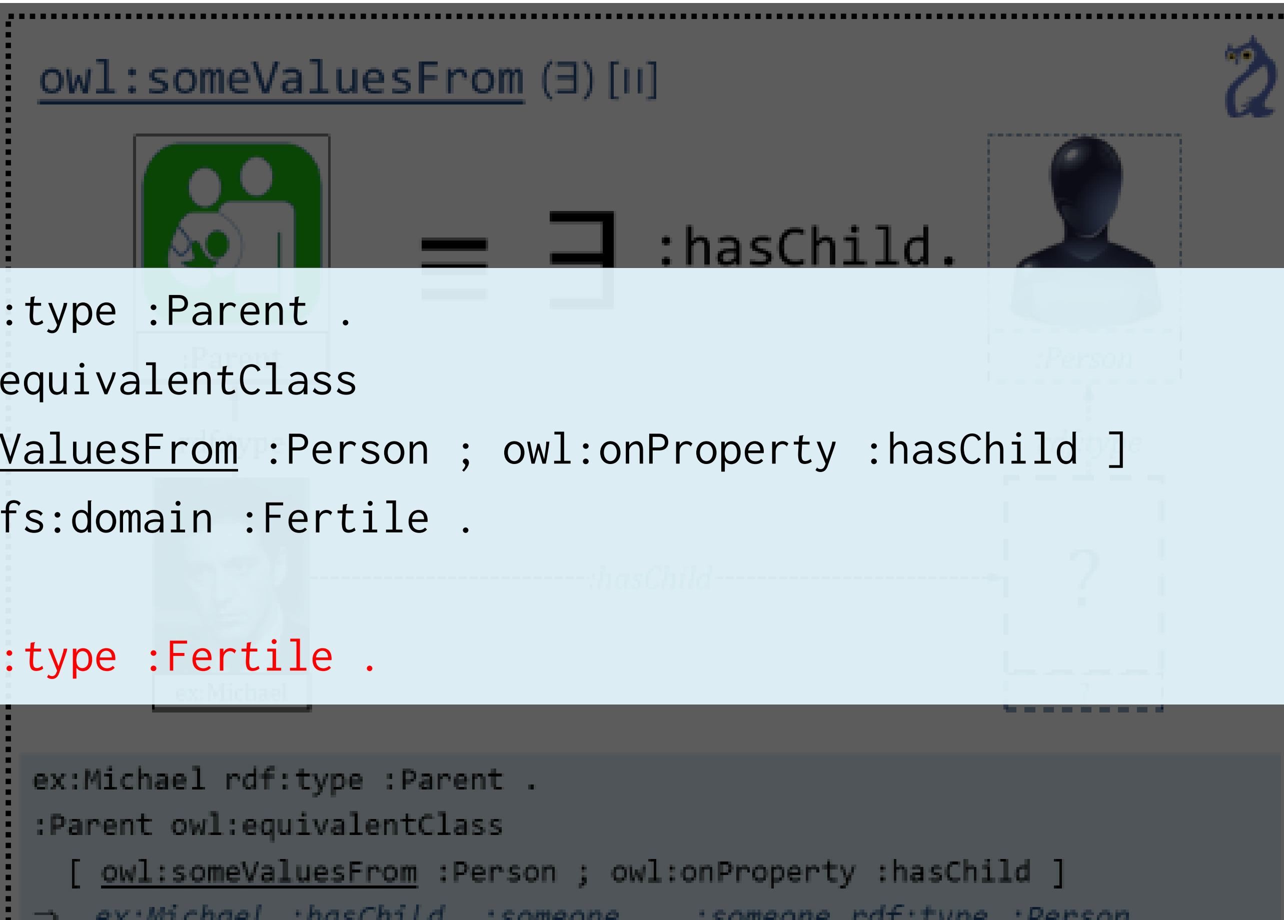
How is OWL2RL/RDF incomplete?

owl:someValuesFrom (∃) [11]




```
ex:Michael rdf:type :Parent .
:Parent owl:equivalentClass
  [ owl:someValuesFrom :Person ; owl:onProperty :hasChild ]
⇒  ex:Michael :hasChild _:someone . _:someone rdf:type :Person .
```

How is OWL2RL/RDF incomplete? EXISTENTIALS

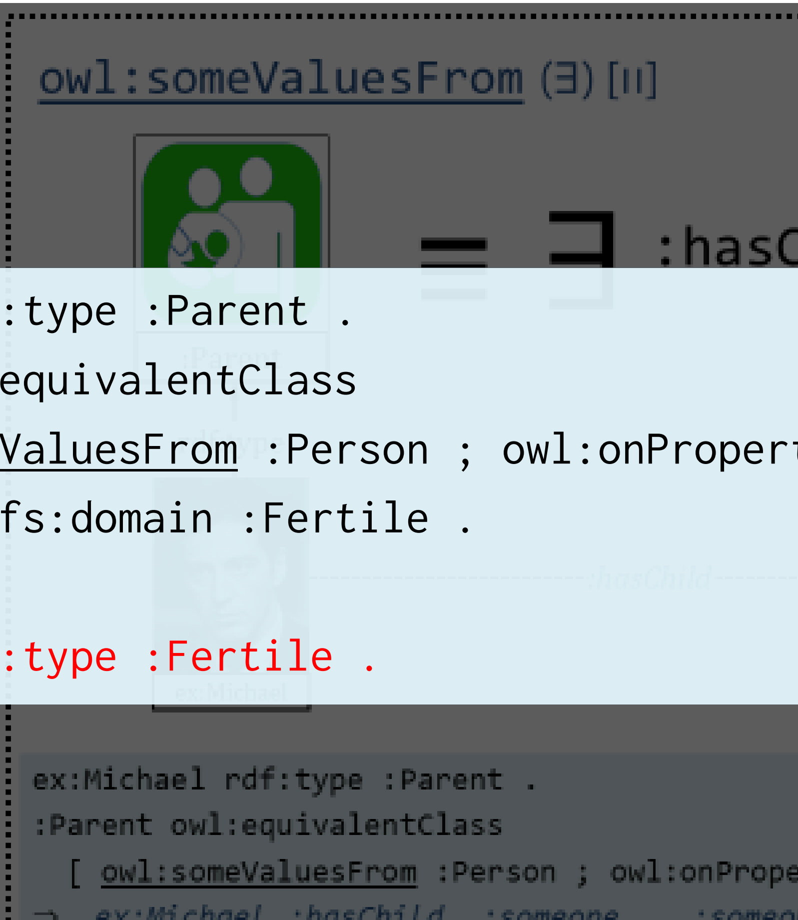



```
:Michael rdf:type :Parent .
:Parent owl:equivalentClass
  [ owl:someValuesFrom :Person ; owl:onProperty :hasChild ]
:hasChild rdfs:domain :Fertile .
⇒
:Michael rdf:type :Fertile .
```

OWL 2 RL/RDF rules will miss this valid inference ...

Misses the information that Michael has **SOME** child!

How is OWL2RL/RDF incomplete? EXISTENTIALS



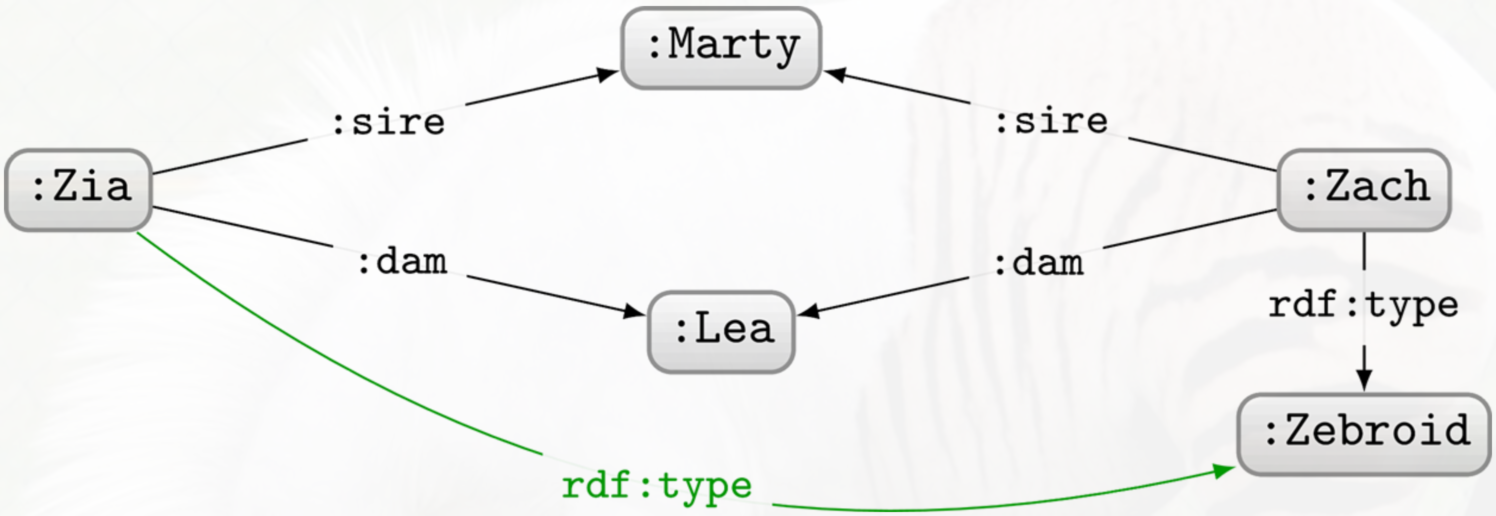

```
:Michael rdf:type :Parent .
:Parent owl:equivalentClass
  [ owl:someValuesFrom :Person ; owl:onProperty :hasChild ]
:hasChild rdfs:domain :Fertile .
⇒
:Michael rdf:type :Fertile .
```

OWL 2 RL/RDF rules will miss this vali

Misses the information that Michael has **SOME** child!

How is OWL2RL/RDF incomplete?

- Missing features
  - `owl:ReflexiveProperty`, `owl:hasSelf`, `owl:minCardinality` …
- Problems with disjunction (OR cases)
  - `owl:unionOf`, `owl:oneOf`, `owl:maxCardinality`, …
- Problems with existentials
  - `owl:someValuesFrom`, `owl:minCardinality`, …
- Problems with counting
  - `owl:minCardinality`, owl:cardinality, …
- Problems with negation
  - `owl:disjointWith`, `owl:propertyDisjointWith`, `owl:complementOf` …
- Incomplete "schema" inferences



What can we intuitively conclude about `Zia`?

`Zia` is also a `Zebroid`!

But not with OWL 2 RL/RDF ☹

Complete reasoners

that may not halt

Complete reasoners that may not halt

- Only line of work on this I know of:

Reasoning in the OWL 2 Full Ontology Language using First-Order Automated Theorem Proving

Michael Schneider[1*] and Geoff Sutcliffe[2]

[1] FZI Research Center for Information Technology, Germany
[2] University of Miami, USA

Abstract. OWL 2 has been standardized by the World Wide Web Consortium (W3C) as a family of ontology languages for the Semantic Web. The most expressive of these languages is OWL 2 Full, but to date no reasoner has been implemented for this language. Consistency and entailment checking are known to be undecidable for OWL 2 Full. We have translated a large fragment of the OWL 2 Full semantics into first-order logic, and used automated theorem proving systems to do reasoning based on this theory. The results are promising, and indicate that this approach can be applied in practice for effective OWL reasoning, beyond the capabilities of current Semantic Web reasoners.
This is an *extended version* of a paper with the same title that has been published at CADE 2011, LNAI 6803, pp. 446–460. The extended version provides appendices with additional resources that were used in the reported evaluation.

Key words: Semantic Web, OWL, First-order logic, ATP

1 Introduction

The Web Ontology Language OWL 2 [16] has been standardized by the World Wide Web Consortium (W3C) as a family of ontology languages for the Semantic Web. OWL 2 includes OWL 2 DL [10], the OWL 2 RL/RDF rules [9], as well as OWL 2 Full [12]. The focus of this work is on reasoning in OWL 2 Full, the most

Complete reasoners that may not halt

- Cons:
  - Erm … reasoner may never halt

What might the "pros" be in this case?

- Pros:
  - Avoid complicated decidability restrictions!

Imagine restricting C or Java to be decidable

1. Don't allow features like loops/recursion
   - But not all programs with loops/recursion fail to halt!
2. Restrict how features like loops/recursion can be used
   - More detailed restrictions allow more programmes but are more complicated to understand ☹

Restrict OWL to guarantee decidability

Restrict OWL to guarantee decidability:
How to guarantee decidability?

- We've seen how to prove that something is undecidable

How can we prove that something is decidable?

- Give an algorithm that halts …
- (Or something non-constructive)

Restrict OWL to guarantee decidability: Sublanguages of OWL 2

- Description Logic community
  - Predates OWL
  - Looks at decidable subsets of First Order Logic
  - Results can be applied to OWL!

- OWL 2 Full: The unrestricted, undecidable language
- OWL 2 DL: A restricted, decidable version

Restrict OWL to guarantee decidability: Sublanguages of OWL 2

Any ideas what we should restrict to make OWL decidable?

But What's the entailment question?

$$T \equiv D_1 \sqcup D_2 \sqcup \ldots \sqcup D_{k-1} \sqcup D_k$$
$$D_i \sqcap D_j \sqsubseteq \bot (\text{for } 1 \leq i < j \leq k)$$
$$T \sqsubseteq (\exists r.T) \sqcap (\exists a.T)$$
$$D_1 \sqsubseteq \forall r.(\bigsqcup_{D' \in R(D_1)} D') \sqcap \forall a.(\bigsqcup_{D' \in A(D_1)} D')$$
$$\ldots$$
$$D_k \sqsubseteq \forall r.(\bigsqcup_{D' \in R(D_k)} D') \sqcap \forall a.(\bigsqcup_{D' \in A(D_k)} D')$$
$$d \sqsubseteq a \circ r, \quad d \sqsubseteq r \circ a, \quad \mathsf{Func}(d)$$

← Must restrict something here (for example)

$$T \equiv \bot$$

Goal: Ontology *O* entails *O'* if and only if *D* has no infinite tiling

If *T* can have <u>any</u> member (a "tile"), it must have an infinite tiling!

If *T* can have <u>no</u> member, it must not have an infinite tiling.

Restrict OWL to guarantee decidability: Sublanguages of OWL 2

Any ideas what we should restrict to make OWL decidable?

But What's the entailment question?

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

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

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

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

$$\ldots$$

$$D_k \sqsubseteq \forall r.(\bigsqcup D') \sqcap \forall a.(\bigsqcup D')$$

← Must restrict something here (for example)

For example, OWL 2 DL restricts functional properties to only be used on "simple properties" (e.g., properties not used in chains)

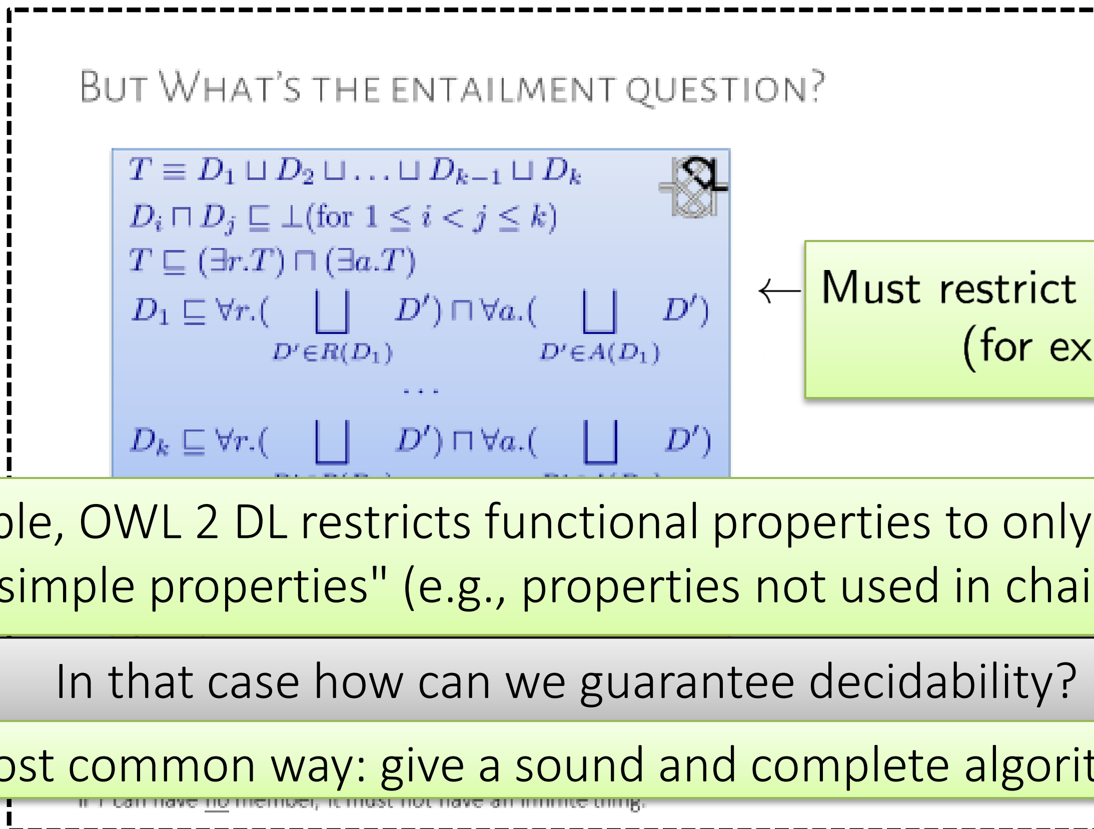
Is this enough to guarantee decidability?

We don't know. We just know this undecidability proof won't work.

(In fact, there are other proofs not needing functional property chains.)

Restrict OWL to guarantee decidability: Sublanguages of OWL 2

Any ideas what we should restrict to make OWL decidable?



← Must restrict something here (for example)

For example, OWL 2 DL restricts functional properties to only be used on "simple properties" (e.g., properties not used in chains)

In that case how can we guarantee decidability?

Most common way: give a sound and complete algorithm!

Restrict OWL to guarantee decidability: Sublanguages of OWL 2

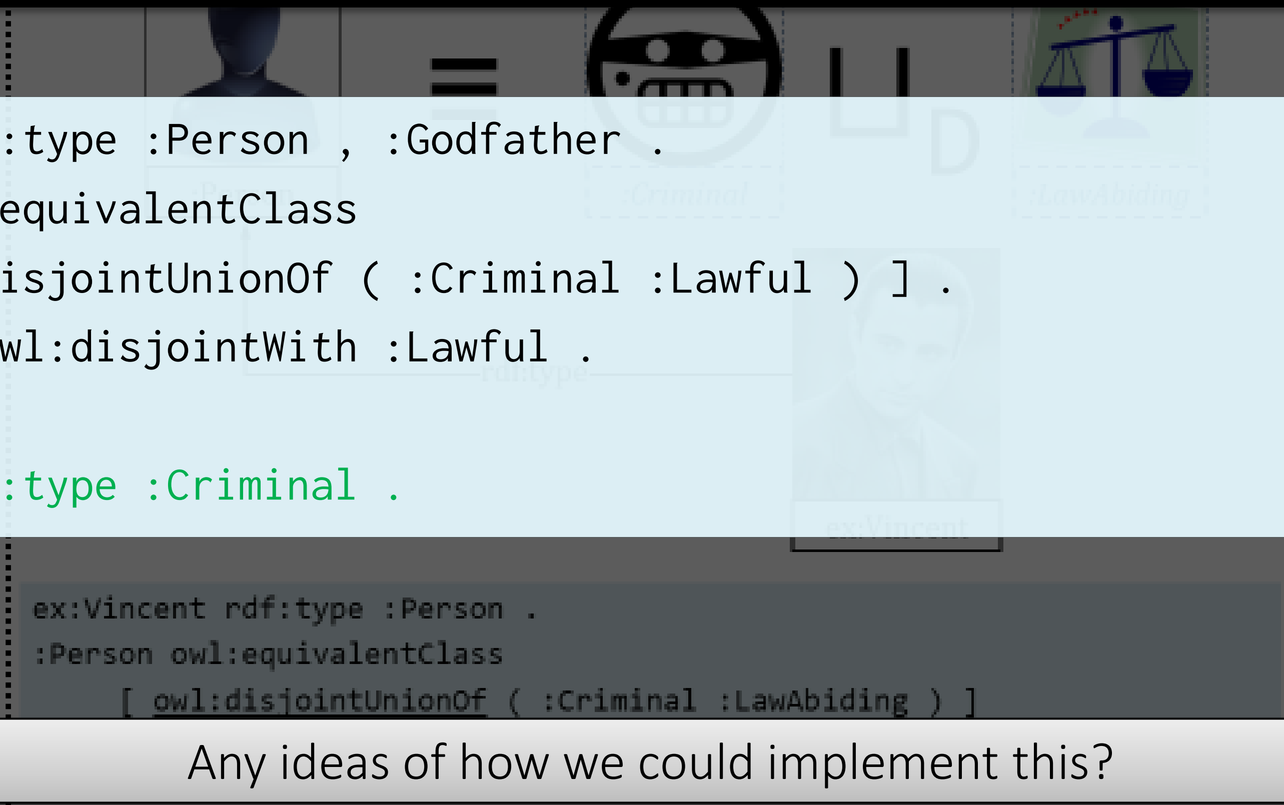
- OWL 2 DL restricts:
  - functional properties to be “simple” (no chains, no transitivity)
  - likewise properties used with has-self, cardinalities, inverse functionality, asymmetry and irreflexivity must be simple
  - need to follow specific RDF syntax and explicitly declare classes, object properties (with IRI values), datatype properties (with literal values)
  - … more (it’s really quite messy ☹)

But in OWL 2 DL ...

So long as O and O' follow the OWL 2 DL restrictions, you are guaranteed a correct answer to $O \models O'$!

But in OWL 2 DL, we can get this entailment ...

So long as O and O' follow the OWL 2 DL restrictions, you are guaranteed a correct answer to $O \models O'$!




```
:Vincent rdf:type :Person , :Godfather .
:Person owl:equivalentClass
     [ owl:disjointUnionOf ( :Criminal :Lawful ) ] .
:Godfather owl:disjointWith :Lawful .
⇒
:Vincent rdf:type :Criminal .
```

Any ideas of how we could implement this?

An algorithm for OWL 2 DL: Tableaux

So long as O and O' follow the OWL 2 DL restrictions, you are guaranteed a correct answer to $O \models O'$!

- Tableaux Algorithm (sketch):
  1. Add ¬O' to O
  2. Expand knowledge using rules
     - Infer low-level assertions
     - Branch on all possibilities created by disjunction
     - Postulate fresh individuals for existentials
     - [...]
  3. If (and only if) every branch is inconsistent: $O \models O'$

Disjunction: Expand all possibilities

Unsatisfiable?

```
:Vincent rdf:type :Person , :Godfather .
:Godfather owl:disjointWith :Lawful .
:Person owl:equivalentClass [ owl:disjointUnionOf ( :Lawful :Criminal ) ] .
¬ :Vincent rdf:type :Criminal .
```

Branch for OR

```
:Vincent rdf:type :Person , :Godfather .
:Godfather owl:disjointWith :Lawful .
:Vincent rdf:type :Lawful .
¬ :Vincent rdf:type :Criminal .
```

```
:Vincent rdf:type :Person , :Godfather .
:Godfather owl:disjointWith :Lawful .
:Vincent rdf:type :Criminal .
¬ :Vincent rdf:type :Criminal .
```

Unsatisfiable in all branches $\rightarrow O \cup \neg O'$ unsatisfiable $\rightarrow O \models O'$

An algorithm for OWL 2 DL: Tableaux

Disjunction: Expand all possibilities

```
:Vincent rdf:type :Person , :Godfather .
```

Unsatisfiable?

```
:Vincent rdf:type :Person , :Godfather .
:Person owl:equivalentClass
     [ owl:disjointUnionOf ( :Criminal :Lawful ) ] .
:Godfather owl:disjointWith :Lawful .
⇒
:Vincent rdf:type :Criminal .
```

```
:Vincent rdf:type :Lawful .
¬ :Vincent rdf:type :Criminal .
```

```
:Vincent rdf:type :Criminal .
¬ :Vincent rdf:type :Criminal .
```

Unsatisfiable in all branches $\rightarrow O \cup \neg O'$ unsatisfiable $\rightarrow O \models O'$

An algorithm for OWL 2 DL: Tableaux

Disjunction: Expand all possibilities

```
:Vincent rdf:type :Person , :Godfather .
```

Unsatisfiable?

```
:Vincent rdf:type :Person , :Godfather .
:Person owl:equivalentClass
     [ owl:disjointUnionOf ( :Criminal :Lawful ) ] .
:Godfather owl:disjointWith :Lawful .
⇒
:Vincent rdf:type :Lawful .  # is it entailed ???
```

```
:Vincent rdf:type :Lawful .
¬ :Vincent rdf:type :Criminal .
```

```
:Vincent rdf:type :Criminal .
¬ :Vincent rdf:type :Criminal .
```

Unsatisfiable in all branches $\rightarrow O \cup \neg O'$ unsatisfiable $\rightarrow O \models O'$

An algorithm for OWL 2 DL: Tableaux

Disjunction: Expand all possibilities

```
:Vincent rdf:type :Person , :Godfather .
:Godfather owl:disjointWith :Lawful .
:Person owl:equivalentClass [ owl:disjointUnionOf ( :Lawful :Criminal ) ] .
¬ :Vincent rdf:type :Lawful.
```

Unsatisfiable?

Branch for OR

```
:Vincent rdf:type :Person , :Godfather .
:Godfather owl:disjointWith :Lawful .
:Vincent rdf:type :Lawful .
¬ :Vincent rdf:type :Lawful .
```

```
:Vincent rdf:type :Person , :Godfather .
:Godfather owl:disjointWith :Lawful .
:Vincent rdf:type :Criminal .
¬ :Vincent rdf:type :Lawful .
```

OKAY

Satisfiable in a branch $\rightarrow O \cup \neg O'$ satisfiable $\rightarrow O \not\models O'$

Disjunction: Expand all possibilities

:Vincent rdf:type :Person , :Godfather .

Unsatisfiable?

```
:Vincent rdf:type :Person , :Godfather .
:Person owl:equivalentClass
     [ owl:disjointUnionOf ( :Criminal :Lawful ) ] .
:Godfather owl:disjointWith :Lawful .
```

⇒

~~:Vincent rdf:type :Lawful .~~ # it is <u>not</u> entailed

Branch for OR

:Vincent rdf:type :Lawful .

¬ :Vincent rdf:type :Lawful .

:Vincent rdf:type :Criminal .

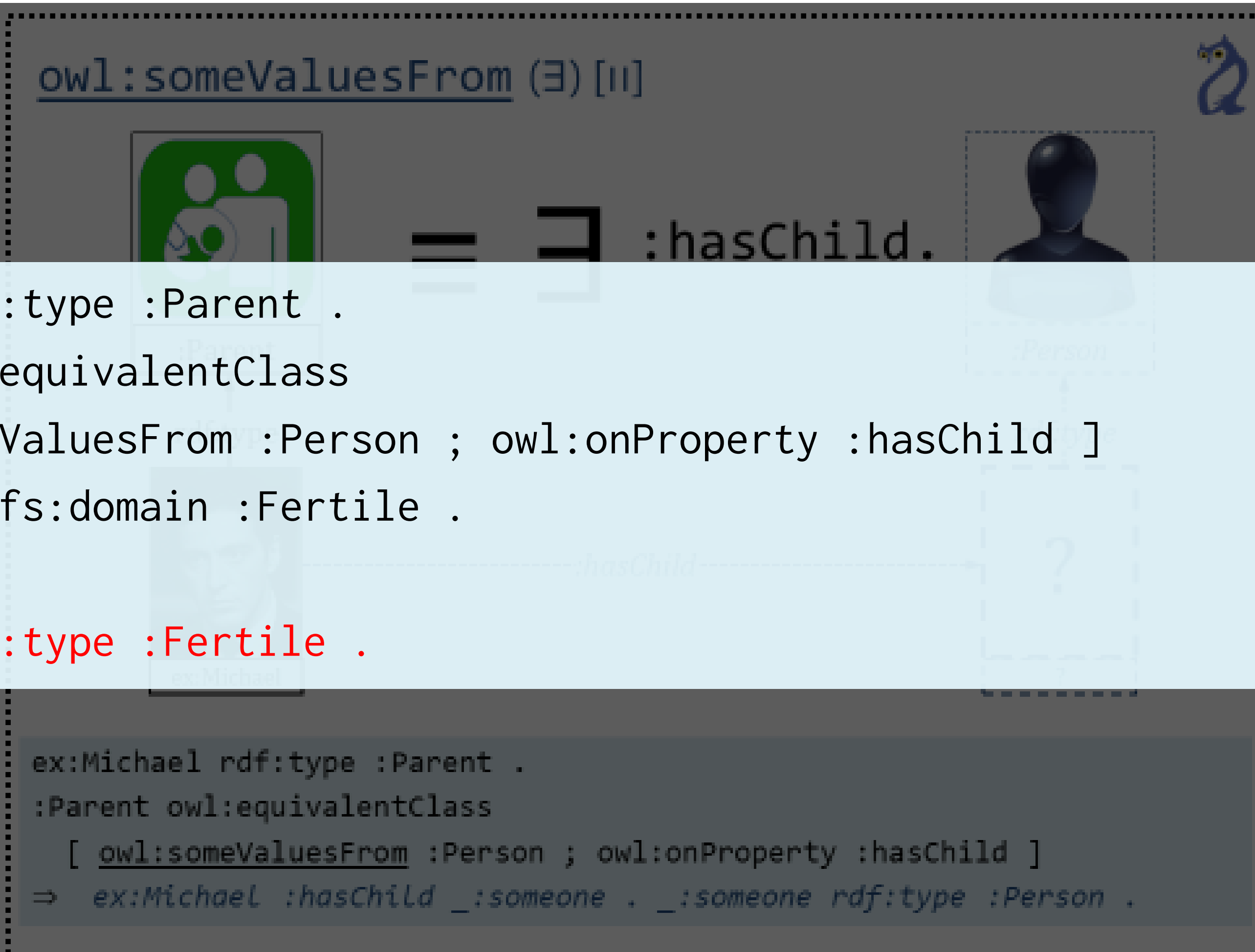
¬ :Vincent rdf:type :Lawful .

OKAY

Satisfiable in a branch $\rightarrow O \cup \neg O'$ satisfiable $\rightarrow O \not\models O'$

An algorithm for OWL 2 DL: Tableaux

```
:Michael rdf:type :Parent .
:Parent owl:equivalentClass
  [ owl:someValuesFrom :Person ; owl:onProperty :hasChild ]
:hasChild rdfs:domain :Fertile .
⇒
:Michael rdf:type :Fertile .
```



An algorithm for OWL 2 DL: Tableaux

Existentials: Try create fresh individuals

Unsatisfiable?

```
:Michael rdf:type :Parent .
:Parent owl:equivalentClass
  [ owl:someValuesFrom :Person ; owl:onProperty :hasChild ]
:hasChild rdfs:domain :Fertile .
¬ :Michael rdf:type :Fertile .
```

Propose a hypothetical child

```
:Michael rdf:type :Parent .
:Michael :hasChild :X .
:X rdf:type :Person .
:Michael rdf:type :Fertile .
¬ :Michael rdf:type :Fertile .
```

Unsatisfiable in all branches $\rightarrow O \cup \neg O'$ unsatisfiable $\rightarrow O \models O'$

An algorithm for OWL 2 DL: Tableaux

Existentials: Try create fresh individuals

```
:Michael rdf:type :Parent .
```

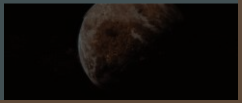
Unsatisfiable?

```
:Michael rdf:type :Parent .
:Parent owl:equivalentClass
  [ owl:someValuesFrom :Person ; owl:onProperty :hasChild ]
:hasChild rdfs:domain :Fertile .
```

$\Rightarrow$

```
:Michael rdf:type :Fertile .
```

```
:X rdf:type :Person .
:Michael rdf:type :Fertile .
¬ :Michael rdf:type :Fertile .
```



Unsatisfiable in all branches $\rightarrow O \cup \neg O'$ unsatisfiable $\rightarrow O \models O'$

An algorithm for OWL 2 DL: Tableaux

So long as O and O' follow the OWL 2 DL restrictions, you are guaranteed a correct answer to $O \models O'$!

- Tableaux Algorithm (sketch):
  1. Add ¬O' to O
  2. Expand knowledge using rules
     - Infer low-level assertions
     - Branch on all possibilities created by disjunction
     - Postulate fresh individuals for existentials
     - [...]
  3. If (and only if) every branch is inconsistent: $O \models O'$

Tableaux algorithm is just "brute force" checking models of the ontologies.
But optimisations and tricks possible for specific logics (like OWL).

An algorithm for OWL 2 DL: Tableaux

So long as O and O' follow the OWL 2 DL restrictions, you are guaranteed a correct answer to $O \models O'$!

- Tableaux Algorithm (sketch):
  1. Add ¬O' to O
  2. Expand knowledge using rules
     - Infer low-level assertions
     - Branch on all possibilities created by disjunction
     - Postulate fresh individuals for existentials
     - [...]
  3. If (and only if) every branch is inconsistent: $O \models O'$

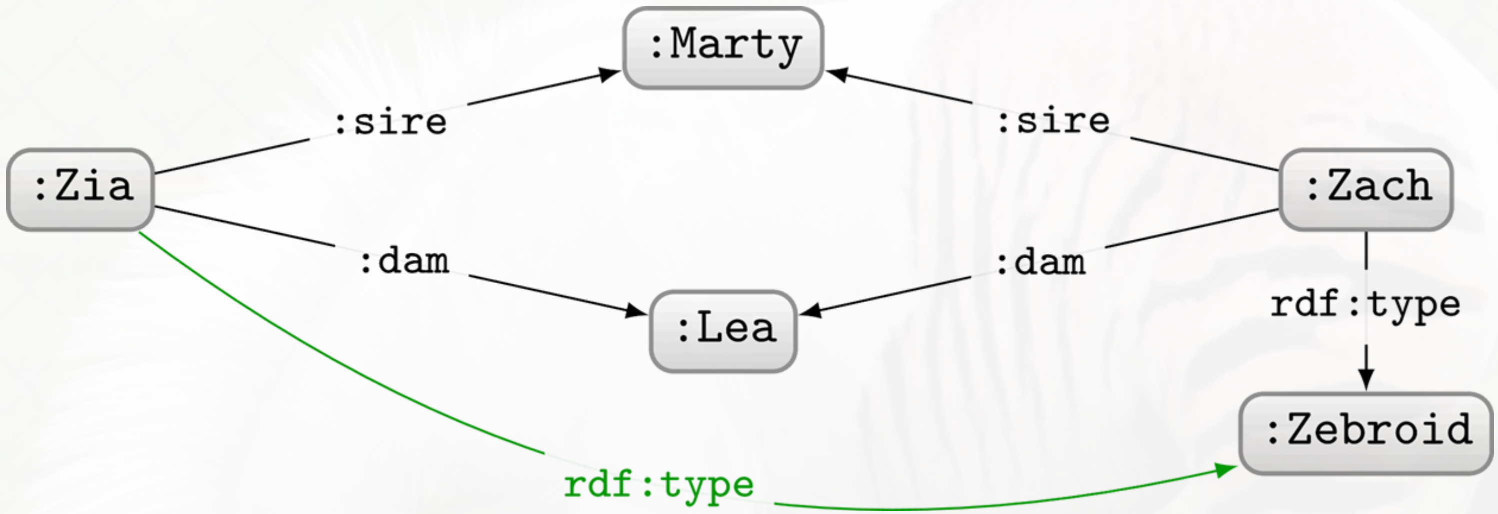
Why do we need to restrict OWL in that case?

To ensure that the tableaux algorithm (with additional tricks) terminates.

AN ALGORITHM FOR OWL 2 DL: TABLEAUX

So long as O and O' follow the OWL 2 DL restrictions, you are guaranteed a correct answer to $O \models O'$!

- Tableaux Algorithm
  - We have a complete entailment algorithm that supports a lot of OWL features and terminates



What can we intuitively conclude about `Zia`?

`Zia` is also a `Zebroid`!

And we can entail this OWL 2 DL! ☺

So, any problems here?

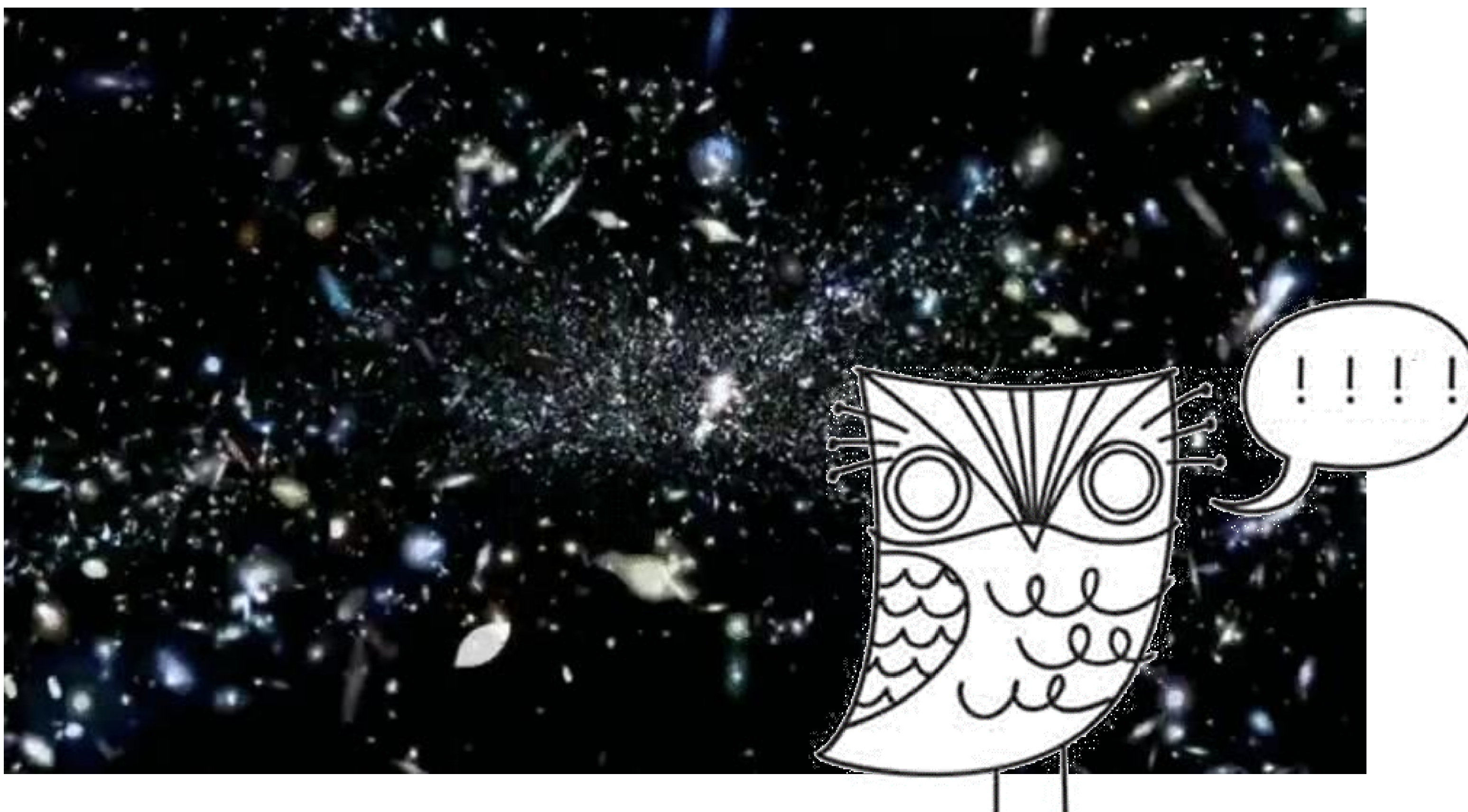
OWL 2 DL: Practical problems

- A few practical problems:
  - We have to give the entailments to check
    - Cannot just ask to compute the entailments
  - Restrictions are complicated
    - Very complicated
    - And often are broken by real-world ontologies
  - Tableaux entailment checks are really expensive
    - Branch for every disjunction suggests exponential
    - If fact, it's N2EXPTIME-complete (!!?!!!)
      - $O(2^{2^n})$ on a non-deterministic machine

N2EXPTIME-Complete (OWL 2 DL's small print) …

- Checking entailment is guaranteed to halt for OWL 2 DL restricted ontologies*

  * halt may not occur before heat death of the universe



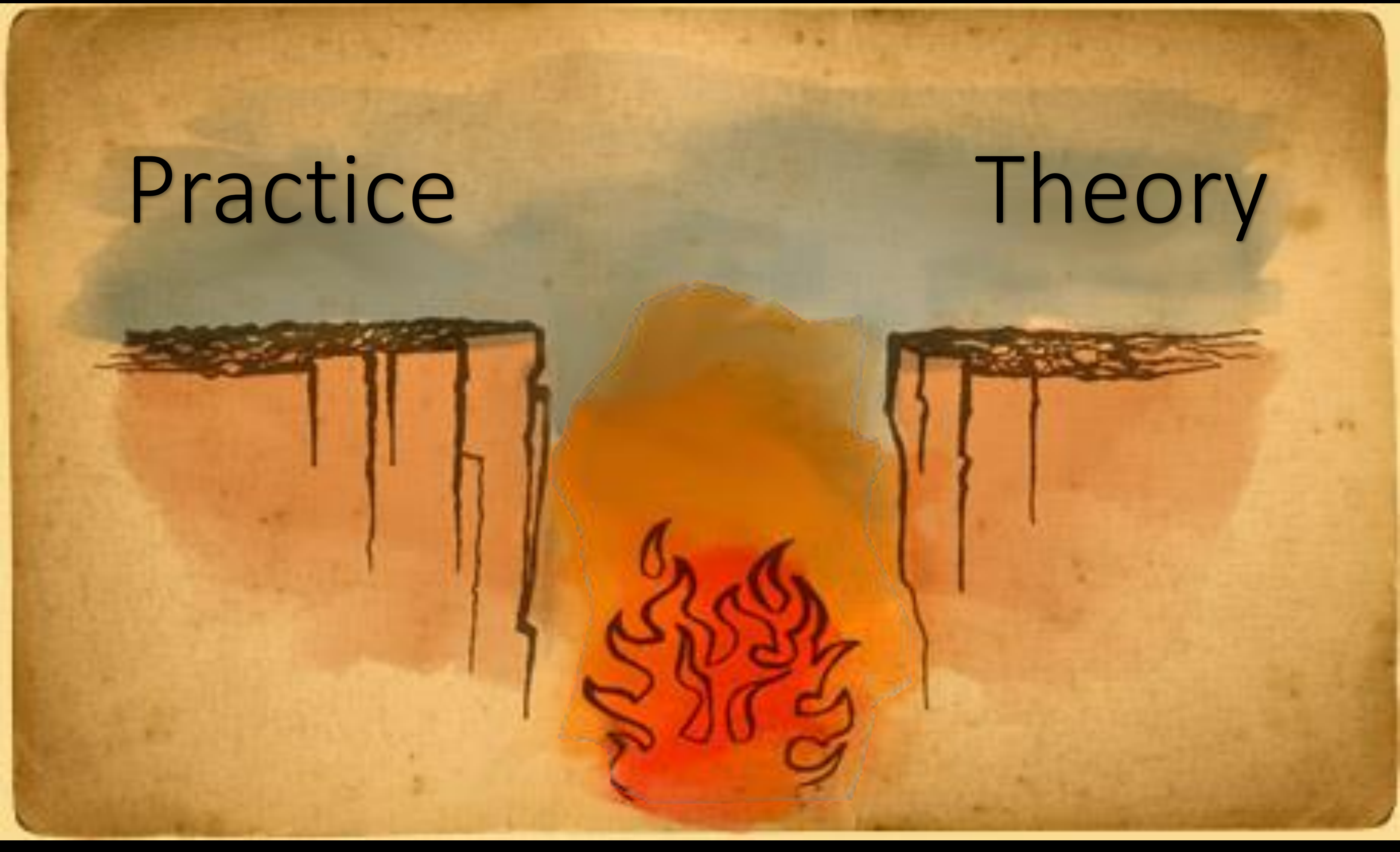
OWL 2 Profiles (briefly)

- More efficient sublanguages of OWL 2 DL
  - More restrictions to allow complete reasoning with more efficient algorithms

- OWL 2 RL: A restriction of OWL 2 DL such that OWL 2 RL/RDF rules provide complete reasoning
- OWL 2 EL: Tractable algorithm for classifying ontologies
- OWL 2 QL: Tractable algorithm rewriting SQL queries

Impressions …

Division between Theory and Practice



Knowledge Representation on the Web:
An open research problem

End of OWL classes (labs to come)

Moving on to SPARQL next

Questions?