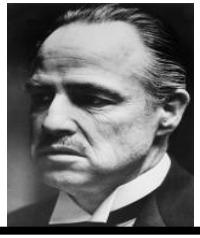
CC6202-1
La Web de Datos
Primavera 2016

Lecture 6: Web Ontology Language (III)

Aidan Hogan
aidhog@gmail.com

PREVIOUSLY ON
“LA WEB DE DATOS”

Modelling family relations with OWL



ex:Vito

ex:Carmela

ex:Sonny

ex:Connie

ex:Fredo

ex:Michael

ex:Vincent

ex:Mary

Materialisation:

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 .
```

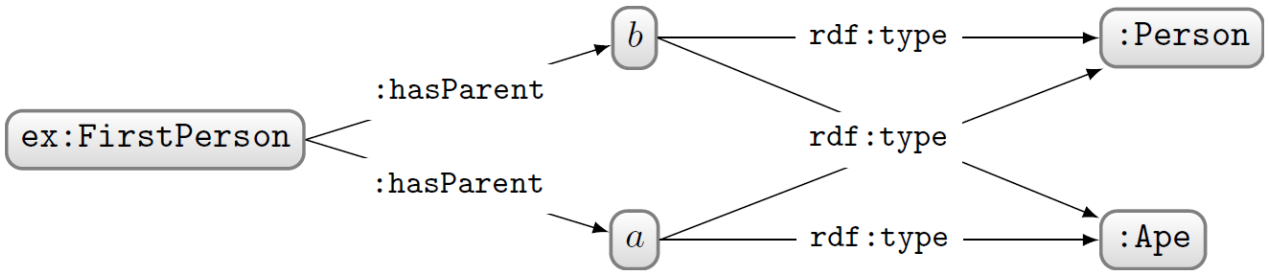
```
ex:Mary :hasParent _:parent1 . _:parent1 a :Person .
ex:Mary :hasParent _:parent2 . _:parent2 a :Person .
_:parent1 :hasParent _:parent11 . _:parent11 a :Person .
_:parent2 :hasParent _:parent12 . _:parent12 a :Person . …
```

Ontology Satisfiability:

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 ] .
```

So does *O* have a model?



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

Entailment checking:

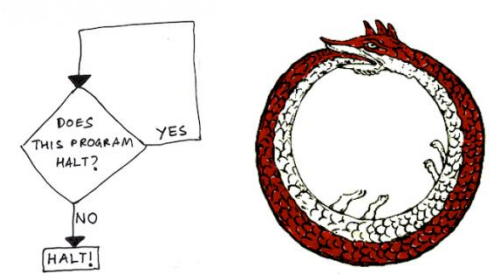
Does *O* entail *O′*?

```
: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 .
```

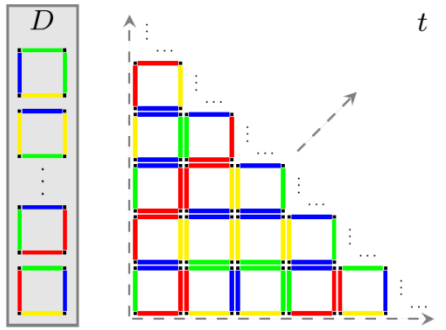
```
ex:Michael :hasParent ex:Carmela .
ex:Michael :hasChild ex:Mary .
ex:Carmela :ancestorOf ex:Mary .
```

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

OWL satisfiability/entailment is powerful



Reduction



OWL satisfiability/entailment checking also **undecidable**!

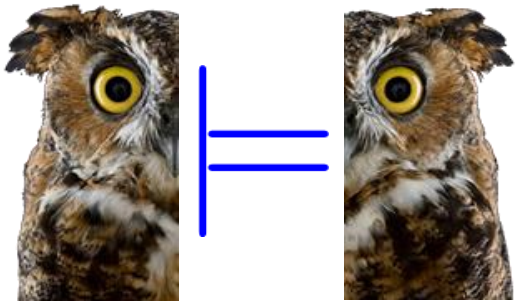
Otherwise could be used to solve Domino Tiling problem …

… and the Halting problem …

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

… and a bunch of other stuff

Reduction



TODAY’S TOPIC …

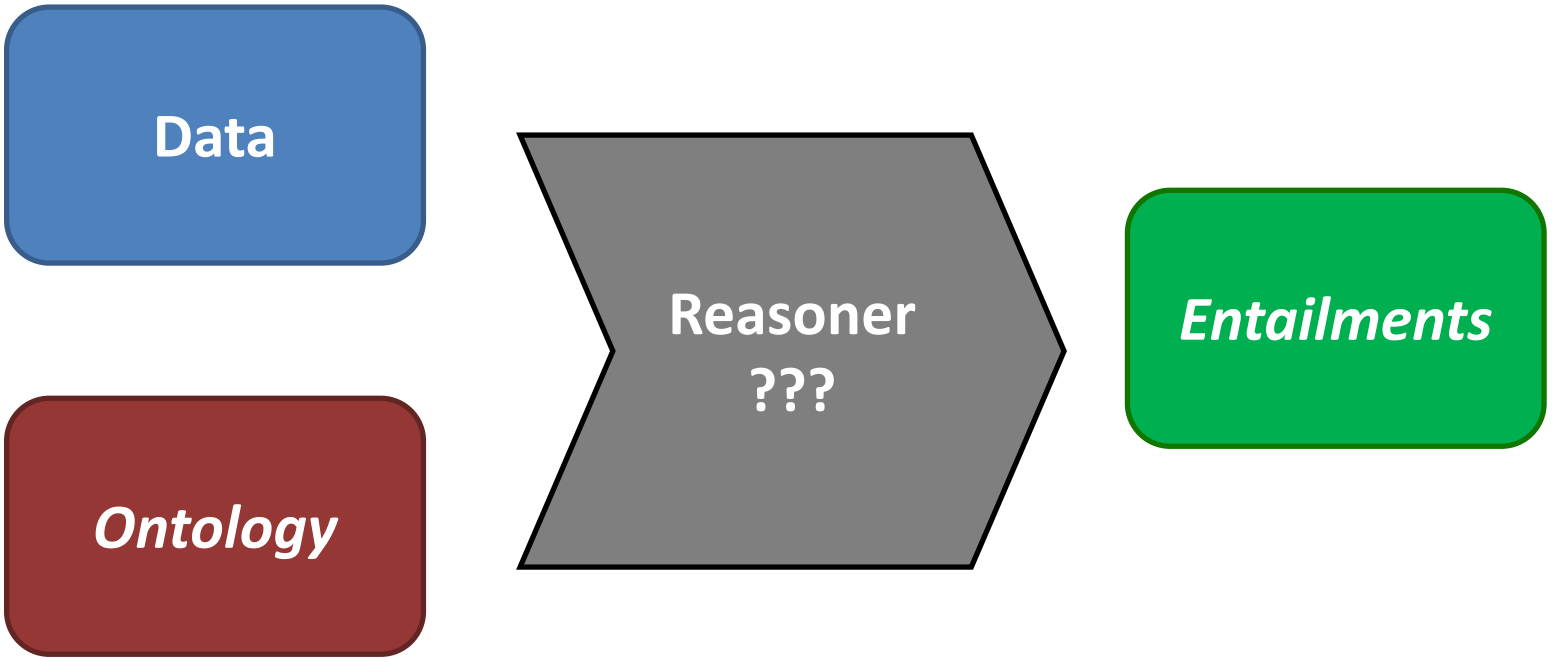
Options …

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 reasoning tasks become decidable
  - 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 …

In the labs …



- But what is the reasoner *actually* doing? …

INCOMPLETE REASONERS THAT HALT

Incomplete reasoners that halt:

Works for materialisation

```
: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 . …
```

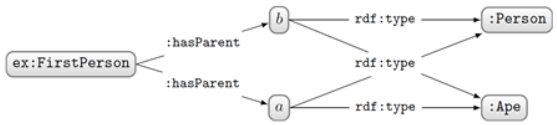
Incomplete reasoners that halt:
for Entailment/Satisfiability Checking?

Ontology Satisfiability:

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 ] .
```

So does O have a model?



YES! Ontology O is Satisfiable!

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 .
```

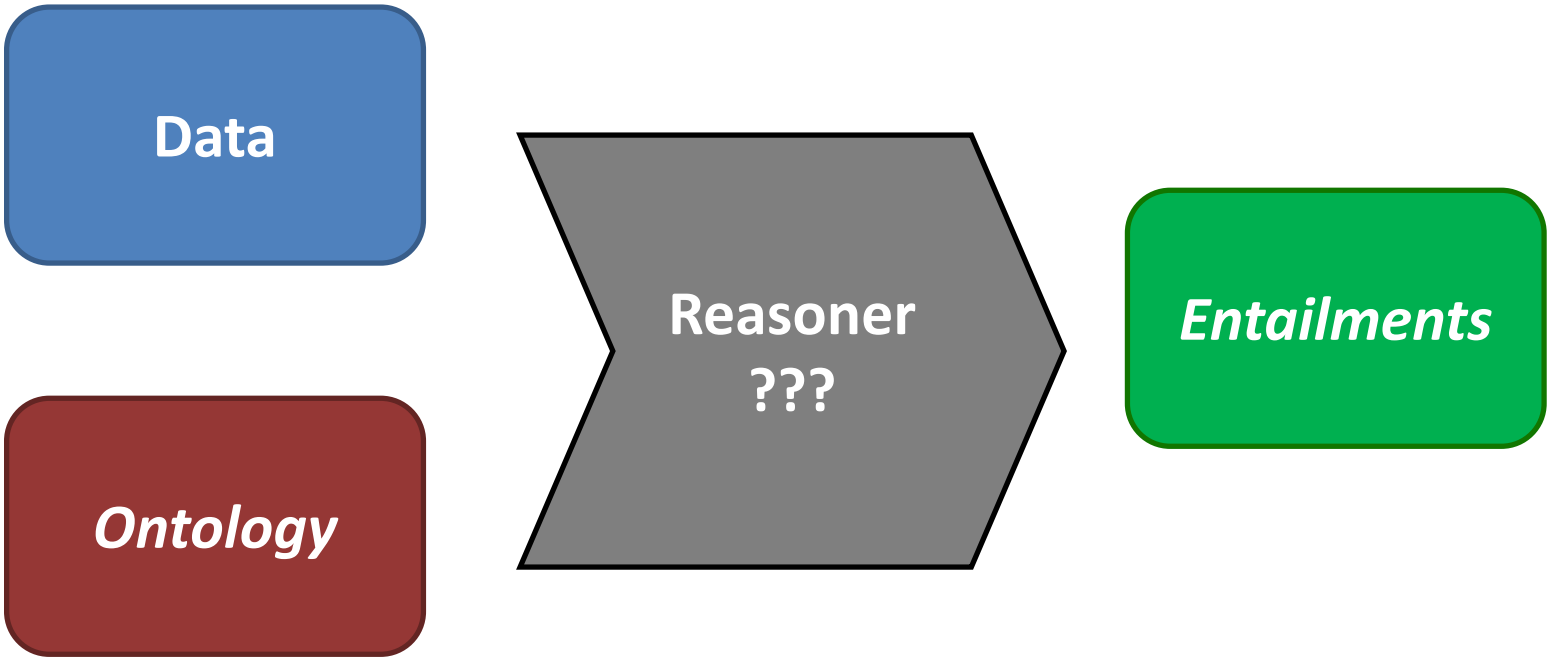
```
ex:Michael :hasParent ex:Carmela .
ex:Michael :hasChild ex:Mary .
ex:Carmela :ancestorOf ex:Mary .
```

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

Why can't we have incomplete satisfiability/entailment checkers?

- Both are decision problems (yes/no)
- What would an incomplete answer be? (ye/n)

In the labs …



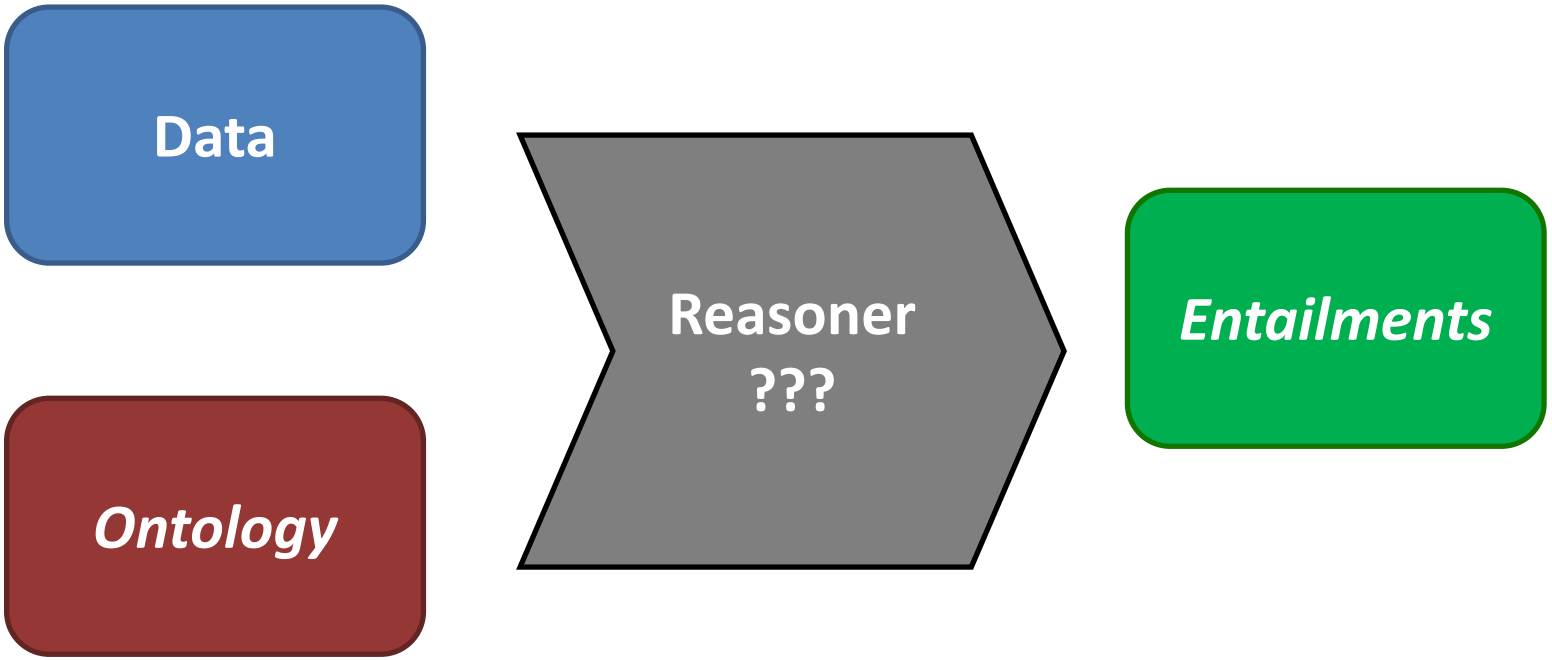
- The reasoner is doing (incomplete) materialisation!

Recall: RDFS reasoning using "rules"

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 .

(Don't worry about rdfD1, rdfs1, rdfs12, rdfs13)

In the labs …



- The reasoner is doing (incomplete) materialisation!
  - Using OWL 2 RL/RDF rules that support RDFS *and* OWL (2)

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

Lots of rules …

- **Goal**: be familiar with idea, not every rule
- Useful for reference
- **Homework**: read over them quickly
  - Let them wash over you ☺

OWL 2 RL/RDF (rules for OWL)

Equality

Table 4. The Semantics of Equality

	If	then	
eq-ref	T(?s, ?p, ?o)	T(?s, owl:sameAs, ?s) T(?p, owl:sameAs, ?p) T(?o, owl:sameAs, ?o)	
eq-sym	T(?x, owl:sameAs, ?y)	T(?y, owl:sameAs, ?x)	
eq-trans	T(?x, owl:sameAs, ?y) T(?y, owl:sameAs, ?z)	T(?x, owl:sameAs, ?z)	
eq-rep-s	T(?s, owl:sameAs, ?s') T(?s, ?p, ?o)	T(?s', ?p, ?o)	
eq-rep-p	T(?p, owl:sameAs, ?p') T(?s, ?p, ?o)	T(?s, ?p', ?o)	
eq-rep-o	T(?o, owl:sameAs, ?o') T(?s, ?p, ?o)	T(?s, ?p, ?o')	
eq-diff1	T(?x, owl:sameAs, ?y) T(?x, owl:differentFrom, ?y)	false	
eq-diff2	T(?x, rdf:type, owl:AllDifferent) T(?x, owl:members, ?y) LIST[?y, $?z_1$, ..., $?z_n$] T($?z_i$, owl:sameAs, $?z_j$)	false	for each $1 \leq i < j \leq n$
eq-diff3	T(?x, rdf:type, owl:AllDifferent) T(?x, owl:distinctMembers, ?y) LIST[?y, $?z_1$, ..., $?z_n$] T($?z_i$, owl:sameAs, $?z_j$)	false	for each $1 \leq i < j \leq n$

OWL 2 RL/RDF (rules for OWL) [Example]

Property Axioms

What rule(s) could we use for `owl:inverseOf`?

prp-inv1	T(?p_1, owl:inverseOf, ?p_2) T(?x, ?p_1, ?y)	T(?y, ?p_2, ?x)
prp-inv2	T(?p_1, owl:inverseOf, ?p_2) T(?x, ?p_2, ?y)	T(?y, ?p_1, ?x)

OWL 2 RL/RDF (rules for OWL)

Property Axioms

Table 5. The Semantics of Axioms about Properties

	If	then
prp-ap		T(ap, rdf:type, owl:AnnotationProperty)
prp-dom	T(?p, rdfs:domain, ?c) T(?x, ?p, ?y)	T(?x, rdf:type, ?c)
prp-rng	T(?p, rdfs:range, ?c) T(?x, ?p, ?y)	T(?y, rdf:type, ?c)
prp-fp	T(?p, rdf:type, owl:FunctionalProperty) T(?x, ?p, $?y_1$) T(?x, ?p, $?y_2$)	T($?y_1$, owl:sameAs, $?y_2$)
prp-ifp	T(?p, rdf:type, owl:InverseFunctionalProperty) T($?x_1$, ?p, ?y) T($?x_2$, ?p, ?y)	T($?x_1$, owl:sameAs, $?x_2$)
prp-irp	T(?p, rdf:type, owl:IrreflexiveProperty) T(?x, ?p, ?x)	false
prp-symp	T(?p, rdf:type, owl:SymmetricProperty) T(?x, ?p, ?y)	T(?y, ?p, ?x)
prp-asyp	T(?p, rdf:type, owl:AsymmetricProperty) T(?x, ?p, ?y) T(?y, ?p, ?x)	false
prp-trp	T(?p, rdf:type, owl:TransitiveProperty) T(?x, ?p, ?y) T(?y, ?p, ?z)	T(?x, ?p, ?z)

...

OWL 2 RL/RDF (rules for OWL)

Property Axioms

prp-spo1	T(?p_1, rdfs:subPropertyOf, ?p_2) T(?x, ?p_1, ?y)	T(?x, ?p_2, ?y)	
prp-spo2	T(?p, owl:propertyChainAxiom, ?x) LIST[?x, ?p_1, ..., ?p_n] T(?u_1, ?p_1, ?u_2) T(?u_2, ?p_2, ?u_3) ... T(?u_n, ?p_n, ?u_{n+1})	T(?u_1, ?p, ?u_{n+1})	
prp-eqp1	T(?p_1, owl:equivalentProperty, ?p_2) T(?x, ?p_1, ?y)	T(?x, ?p_2, ?y)	
prp-eqp2	T(?p_1, owl:equivalentProperty, ?p_2) T(?x, ?p_2, ?y)	T(?x, ?p_1, ?y)	
prp-pdw	T(?p_1, owl:propertyDisjointWith, ?p_2) T(?x, ?p_1, ?y) T(?x, ?p_2, ?y)	false	
prp-adp	T(?x, rdf:type, owl:AllDisjointProperties) T(?x, owl:members, ?y) LIST[?y, ?p_1, ..., ?p_n] T(?u, ?p_i, ?v) T(?u, ?p_j, ?v)	false	for each $1 \le i < j \le n$
prp-inv1	T(?p_1, owl:inverseOf, ?p_2) T(?x, ?p_1, ?y)	T(?y, ?p_2, ?x)	
prp-inv2	T(?p_1, owl:inverseOf, ?p_2) T(?x, ?p_2, ?y)	T(?y, ?p_1, ?x)	

...

OWL 2 RL/RDF (rules for OWL)

Property Axioms

prp-key	T(?c, owl:hasKey, ?u) LIST[?u, ?p_1, ..., ?p_n] T(?x, rdf:type, ?c) T(?x, ?p_1, ?z_1) ... T(?x, ?p_n, ?z_n) T(?y, rdf:type, ?c) T(?y, ?p_1, ?z_1) ... T(?y, ?p_n, ?z_n)	T(?x, owl:sameAs, ?y)
prp-npa1	T(?x, owl:sourceIndividual, ?i_1) T(?x, owl:assertionProperty, ?p) T(?x, owl:targetIndividual, ?i_2) T(?i_1, ?p, ?i_2)	false
prp-npa2	T(?x, owl:sourceIndividual, ?i) T(?x, owl:assertionProperty, ?p) T(?x, owl:targetValue, ?lt) T(?i, ?p, ?lt)	false

OWL 2 RL/RDF (rules for OWL) [Example]

Class Axioms

What rule(s) could we use for `owl:disjointWith`?

cax-dw	T(?c_1, owl:disjointWith, ?c_2) T(?x, rdf:type, ?c_1) T(?x, rdf:type, ?c_2)	false

...

OWL 2 RL/RDF (rules for OWL)

Class Axioms

Table 7. The Semantics of Class Axioms

	If	then	
cax-sco	T(?c_1, rdfs:subClassOf, ?c_2) T(?x, rdf:type, ?c_1)	T(?x, rdf:type, ?c_2)	
cax-eqc1	T(?c_1, owl:equivalentClass, ?c_2) T(?x, rdf:type, ?c_1)	T(?x, rdf:type, ?c_2)	
cax-eqc2	T(?c_1, owl:equivalentClass, ?c_2) T(?x, rdf:type, ?c_2)	T(?x, rdf:type, ?c_1)	
cax-dw	T(?c_1, owl:disjointWith, ?c_2) T(?x, rdf:type, ?c_1) T(?x, rdf:type, ?c_2)	false	
cax-adc	T(?x, rdf:type, owl:AllDisjointClasses) T(?x, owl:members, ?y) LIST[?y, ?c_1, ..., ?c_n] T(?z, rdf:type, ?c_i) T(?z, rdf:type, ?c_j)	false	for each $1 \le i < j \le n$

...

OWL 2 RL/RDF (rules for OWL) [Example]

Class definitions

What rule(s) could we use for `owl:intersectionOf`?

cls-int1	T(?c, owl:intersectionOf, ?x) LIST[?x, ?c_1, ..., ?c_n] T(?y, rdf:type, ?c_1) T(?y, rdf:type, ?c_2) ... T(?y, rdf:type, ?c_n)	T(?y, rdf:type, ?c)
cls-int2	T(?c, owl:intersectionOf, ?x) LIST[?x, ?c_1, ..., ?c_n] T(?y, rdf:type, ?c)	T(?y, rdf:type, ?c_1) T(?y, rdf:type, ?c_2) ... T(?y, rdf:type, ?c_n)

What rule(s) could we use for `owl:allValuesFrom`?

cls-avf	T(?x, owl:allValuesFrom, ?y) T(?x, owl:onProperty, ?p) T(?u, rdf:type, ?x) T(?u, ?p, ?v)	T(?v, rdf:type, ?y)

No way to write rule for "opposite" direction:
if ... T(?x, owl:allValuesFrom, ?y) and T(?x, owl:onProperty, ?p)
and where T(?u,?p,?v) implies ?v of rdf:type ?y then T(?u, rdf:type, ?x)

OWL 2 RL/RDF (rules for OWL)

Class definitions

Table 6. The Semantics of Classes

	If	then	
cls-thing		T(owl:Thing, rdf:type, owl:Class)	
cls-nothing1		T(owl:Nothing, rdf:type, owl:Class)	
cls-nothing2	T(?x, rdf:type, owl:Nothing)	false	
cls-int1	T(?c, owl:intersectionOf, ?x) LIST[?x, ?c_1, ..., ?c_n] T(?y, rdf:type, ?c_1) T(?y, rdf:type, ?c_2) ... T(?y, rdf:type, ?c_n)	T(?y, rdf:type, ?c)	
cls-int2	T(?c, owl:intersectionOf, ?x) LIST[?x, ?c_1, ..., ?c_n] T(?y, rdf:type, ?c)	T(?y, rdf:type, ?c_1) T(?y, rdf:type, ?c_2) ... T(?y, rdf:type, ?c_n)	
cls-uni	T(?c, owl:unionOf, ?x) LIST[?x, ?c_1, ..., ?c_n] T(?y, rdf:type, ?c_i)	T(?y, rdf:type, ?c)	for each $1 \leq i \leq n$
cls-com	T(?c_1, owl:complementOf, ?c_2) T(?x, rdf:type, ?c_1) T(?x, rdf:type, ?c_2)	false	

...

OWL 2 RL/RDF (rules for OWL)

Class definitions

cls-svf1	T(?x, owl:someValuesFrom, ?y) T(?x, owl:onProperty, ?p) T(?u, ?p, ?v) T(?v, rdf:type, ?y)	T(?u, rdf:type, ?x)
cls-svf2	T(?x, owl:someValuesFrom, owl:Thing) T(?x, owl:onProperty, ?p) T(?u, ?p, ?v)	T(?u, rdf:type, ?x)
cls-avf	T(?x, owl:allValuesFrom, ?y) T(?x, owl:onProperty, ?p) T(?u, rdf:type, ?x) T(?u, ?p, ?v)	T(?v, rdf:type, ?y)
cls-hv1	T(?x, owl:hasValue, ?y) T(?x, owl:onProperty, ?p) T(?u, rdf:type, ?x)	T(?u, ?p, ?y)
cls-hv2	T(?x, owl:hasValue, ?y) T(?x, owl:onProperty, ?p) T(?u, ?p, ?y)	T(?u, rdf:type, ?x)
cls-maxc1	T(?x, owl:maxCardinality, "0"^^xsd:nonNegativeInteger) T(?x, owl:onProperty, ?p) T(?u, rdf:type, ?x) T(?u, ?p, ?y)	false
cls-maxc2	T(?x, owl:maxCardinality, "1"^^xsd:nonNegativeInteger) T(?x, owl:onProperty, ?p) T(?u, rdf:type, ?x) T(?u, ?p, $?y_1$) T(?u, ?p, $?y_2$)	T($?y_1$, owl:sameAs, $?y_2$)

...

OWL 2 RL/RDF (rules for OWL)

Class definitions

cls-maxqc1	T(?x, owl:maxQualifiedCardinality, "0"^^xsd:nonNegativeInteger) T(?x, owl:onProperty, ?p) T(?x, owl:onClass, ?c) T(?u, rdf:type, ?x) T(?u, ?p, ?y) T(?y, rdf:type, ?c)	false
cls-maxqc2	T(?x, owl:maxQualifiedCardinality, "0"^^xsd:nonNegativeInteger) T(?x, owl:onProperty, ?p) T(?x, owl:onClass, owl:Thing) T(?u, rdf:type, ?x) T(?u, ?p, ?y)	false
cls-maxqc3	T(?x, owl:maxQualifiedCardinality, "1"^^xsd:nonNegativeInteger) T(?x, owl:onProperty, ?p) T(?x, owl:onClass, ?c) T(?u, rdf:type, ?x) T(?u, ?p, $?y_1$) T($?y_1$, rdf:type, ?c) T(?u, ?p, $?y_2$) T($?y_2$, rdf:type, ?c)	T($?y_1$, owl:sameAs, $?y_2$)
cls-maxqc4	T(?x, owl:maxQualifiedCardinality, "1"^^xsd:nonNegativeInteger) T(?x, owl:onProperty, ?p) T(?x, owl:onClass, owl:Thing) T(?u, rdf:type, ?x) T(?u, ?p, $?y_1$) T(?u, ?p, $?y_2$)	T($?y_1$, owl:sameAs, $?y_2$)
cls-oo	T(?c, owl:oneOf, ?x) LIST[?x, $?y_1$, ..., $?y_n$]	T($?y_1$, rdf:type, ?c) ... T($?y_n$, rdf:type, ?c)

OWL 2 RL/RDF (rules for OWL) [Example]

Schema

cax-sco	T(?c_1, rdfs:subClassOf, ?c_2) T(?x, rdf:type, ?c_1)	T(?x, rdf:type, ?c_2)

… but what other rule(s) are we missing for `rdfs:subClassOf`?

scm-sco	T(?c_1, rdfs:subClassOf, ?c_2) T(?c_2, rdfs:subClassOf, ?c_3)	T(?c_1, rdfs:subClassOf, ?c_3)
scm-eqc2	T(?c_1, rdfs:subClassOf, ?c_2) T(?c_2, rdfs:subClassOf, ?c_1)	T(?c_1, owl:equivalentClass, ?c_2)

…

OWL 2 RL/RDF (rules for OWL)

Schema

Table 9. The Semantics of Schema Vocabulary

	If	then
scm-cls	T(?c, rdf:type, owl:Class)	T(?c, rdfs:subClassOf, ?c) T(?c, owl:equivalentClass, ?c) T(?c, rdfs:subClassOf, owl:Thing) T(owl:Nothing, rdfs:subClassOf, ?c)
scm-sco	T(?c_1, rdfs:subClassOf, ?c_2) T(?c_2, rdfs:subClassOf, ?c_3)	T(?c_1, rdfs:subClassOf, ?c_3)
scm-eqc1	T(?c_1, owl:equivalentClass, ?c_2)	T(?c_1, rdfs:subClassOf, ?c_2) T(?c_2, rdfs:subClassOf, ?c_1)
scm-eqc2	T(?c_1, rdfs:subClassOf, ?c_2) T(?c_2, rdfs:subClassOf, ?c_1)	T(?c_1, owl:equivalentClass, ?c_2)
scm-op	T(?p, rdf:type, owl:ObjectProperty)	T(?p, rdfs:subPropertyOf, ?p) T(?p, owl:equivalentProperty, ?p)
scm-dp	T(?p, rdf:type, owl:DatatypeProperty)	T(?p, rdfs:subPropertyOf, ?p) T(?p, owl:equivalentProperty, ?p)
scm-spo	T(?p_1, rdfs:subPropertyOf, ?p_2) T(?p_2, rdfs:subPropertyOf, ?p_3)	T(?p_1, rdfs:subPropertyOf, ?p_3)
scm-eqp1	T(?p_1, owl:equivalentProperty, ?p_2)	T(?p_1, rdfs:subPropertyOf, ?p_2) T(?p_2, rdfs:subPropertyOf, ?p_1)
scm-eqp2	T(?p_1, rdfs:subPropertyOf, ?p_2) T(?p_2, rdfs:subPropertyOf, ?p_1)	T(?p_1, owl:equivalentProperty, ?p_2)

...

OWL 2 RL/RDF (rules for OWL)

Schema

scm-dom2	T(?p_2, rdfs:domain, ?c) T(?p_1, rdfs:subPropertyOf, ?p_2)	T(?p_1, rdfs:domain, ?c)
scm-rng1	T(?p, rdfs:range, ?c_1) T(?c_1, rdfs:subClassOf, ?c_2)	T(?p, rdfs:range, ?c_2)
scm-rng2	T(?p_2, rdfs:range, ?c) T(?p_1, rdfs:subPropertyOf, ?p_2)	T(?p_1, rdfs:range, ?c)
scm-hv	T(?c_1, owl:hasValue, ?i) T(?c_1, owl:onProperty, ?p_1) T(?c_2, owl:hasValue, ?i) T(?c_2, owl:onProperty, ?p_2) T(?p_1, rdfs:subPropertyOf, ?p_2)	T(?c_1, rdfs:subClassOf, ?c_2)
scm-svf1	T(?c_1, owl:someValuesFrom, ?y_1) T(?c_1, owl:onProperty, ?p) T(?c_2, owl:someValuesFrom, ?y_2) T(?c_2, owl:onProperty, ?p) T(?y_1, rdfs:subClassOf, ?y_2)	T(?c_1, rdfs:subClassOf, ?c_2)
scm-svf2	T(?c_1, owl:someValuesFrom, ?y) T(?c_1, owl:onProperty, ?p_1) T(?c_2, owl:someValuesFrom, ?y) T(?c_2, owl:onProperty, ?p_2) T(?p_1, rdfs:subPropertyOf, ?p_2)	T(?c_1, rdfs:subClassOf, ?c_2)

...

OWL 2 RL/RDF (rules for OWL)

Schema

scm-avf1	T(?c_1, owl:allValuesFrom, ?y_1) T(?c_1, owl:onProperty, ?p) T(?c_2, owl:allValuesFrom, ?y_2) T(?c_2, owl:onProperty, ?p) T(?y_1, rdfs:subClassOf, ?y_2)	T(?c_1, rdfs:subClassOf, ?c_2)
scm-avf2	T(?c_1, owl:allValuesFrom, ?y) T(?c_1, owl:onProperty, ?p_1) T(?c_2, owl:allValuesFrom, ?y) T(?c_2, owl:onProperty, ?p_2) T(?p_1, rdfs:subPropertyOf, ?p_2)	T(?c_2, rdfs:subClassOf, ?c_1)
scm-int	T(?c, owl:intersectionOf, ?x) LIST[?x, ?c_1, ..., ?c_n]	T(?c, rdfs:subClassOf, ?c_1) T(?c, rdfs:subClassOf, ?c_2) ... T(?c, rdfs:subClassOf, ?c_n)
scm-uni	T(?c, owl:unionOf, ?x) LIST[?x, ?c_1, ..., ?c_n]	T(?c_1, rdfs:subClassOf, ?c) T(?c_2, rdfs:subClassOf, ?c) ... T(?c_n, rdfs:subClassOf, ?c)

OWL 2 RL/RDF (rules for OWL)

Datatypes

Want to capture:

```
"2"^^xsd:integer owl:sameAs "2.0"^^xsd:decimal .

"2"^^xsd:integer owl:differentFrom "3.0"^^xsd:decimal .

"2"^^xsd:integer owl:differentFrom "2.0"^^xsd:string .
```

(Literals allowed in subject positions while reasoning, removed afterwards)

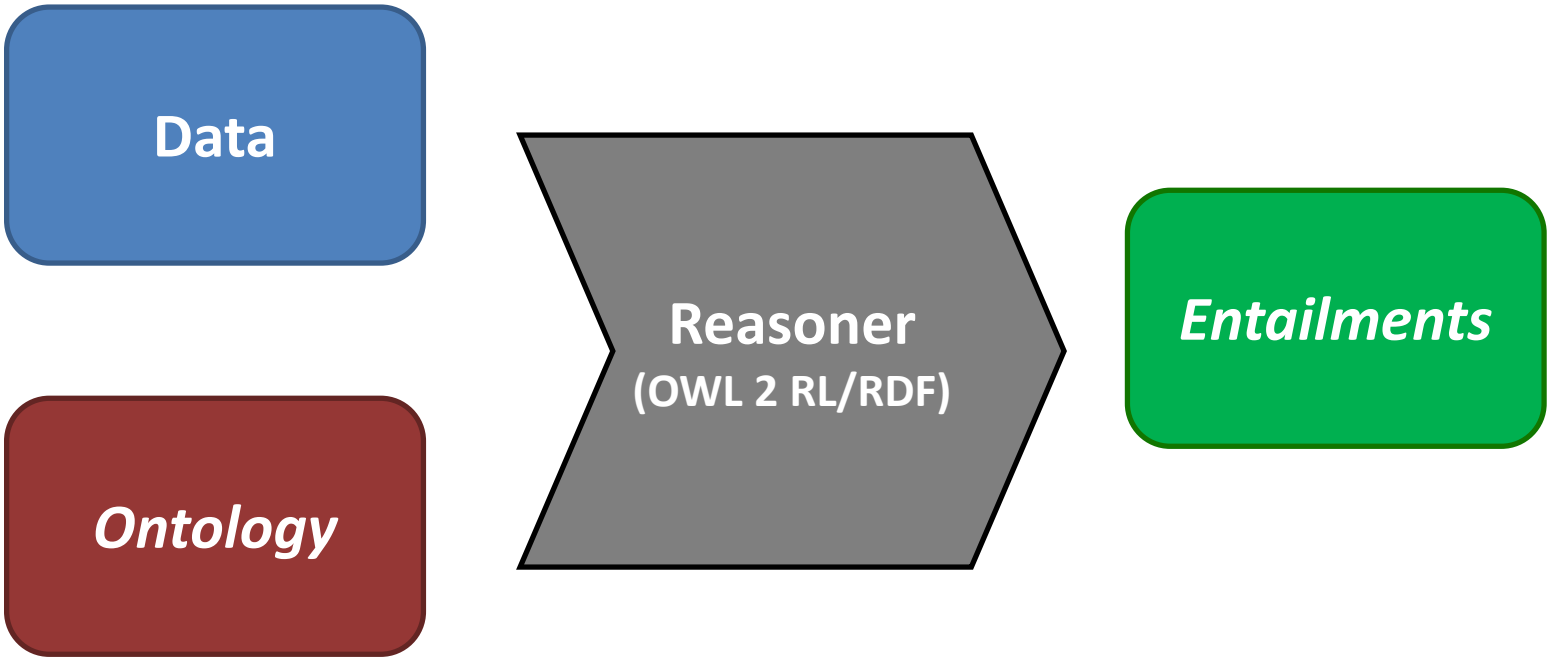
OWL 2 RL/RDF (rules for OWL)

Datatypes

Table 8. The Semantics of Datatypes

	If	then	
dt-type1		T(dt, rdf:type, rdfs:Datatype)	for each datatype dt supported in OWL 2 RL
dt-type2		T(lt, rdf:type, dt)	for each literal lt and each datatype dt supported in OWL 2 RL such that the data value of lt is contained in the value space of dt
dt-eq		T(lt_1, owl:sameAs, lt_2)	for all literals lt_1 and lt_2 with the same data value
dt-diff		T(lt_1, owl:differentFrom, lt_2)	for all literals lt_1 and lt_2 with different data values
dt-not-type	T(lt, rdf:type, dt)	false	for each literal lt and each datatype dt supported in OWL 2 RL such that the data value of lt is not contained in the value space of dt

In the labs …



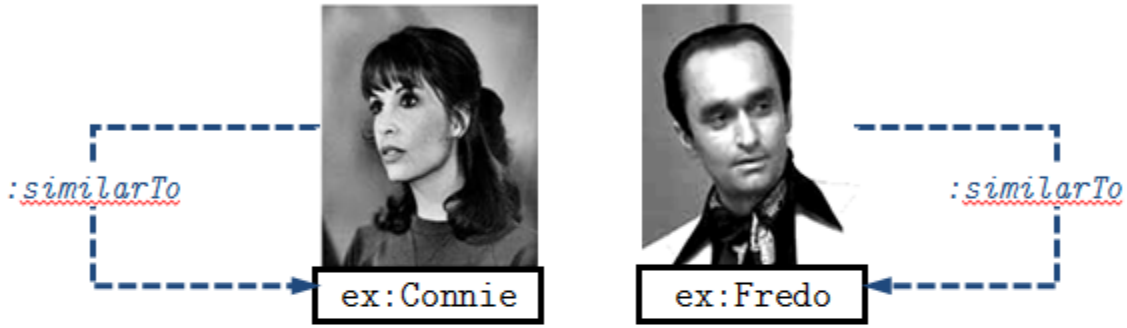
- Applies these OWL 2 RL/RDF rules recursively until nothing new is found

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

Why Incomplete?

Missing Features

owl:ReflexiveProperty



:similarTo rdf:type owl:ReflexiveProperty .

⇒ *ex:Connie :similarTo ex:Connie .*

ex:Freddie :similarTo ex:Freddie .

(everything :similarTo itself)

Why Incomplete?

Missing Features

owl:hasSelf (Self) [i]

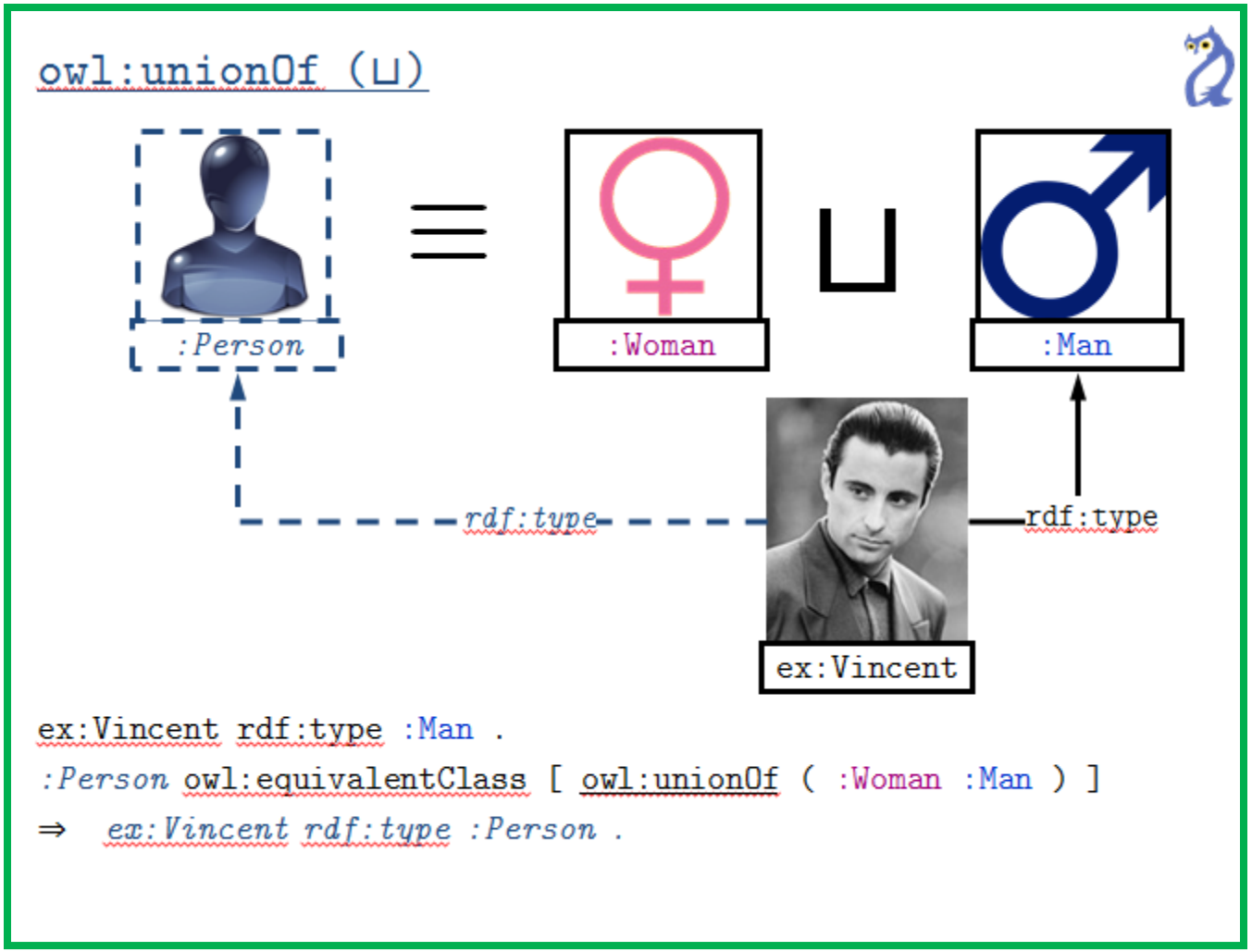
$\sqsubseteq$ Self(*:loves*)

```
ex:Vincent rdf:type :Narcissist .
:Narcissist rdfs:subClassOf
  [ owl:hasSelf true ; owl:onProperty :loves ]
⇒  ex:Vincent :loves ex:Vincent .
```

Why Incomplete?

Incomplete for some Features

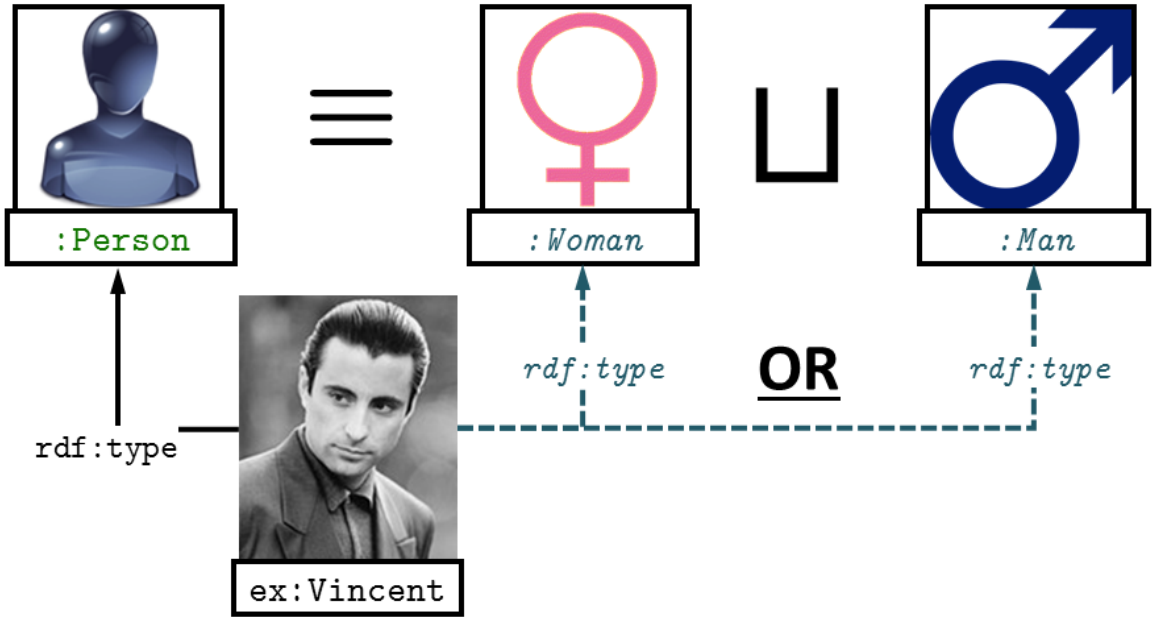
cls-uni	T(?c, owl:unionOf, ?x) LIST[?x, ?c_1, ..., ?c_n] T(?y, rdf:type, ?c_i)	T(?y, rdf:type, ?c)	for each $1 \le i \le n$



Why Incomplete?

Incomplete for some Features

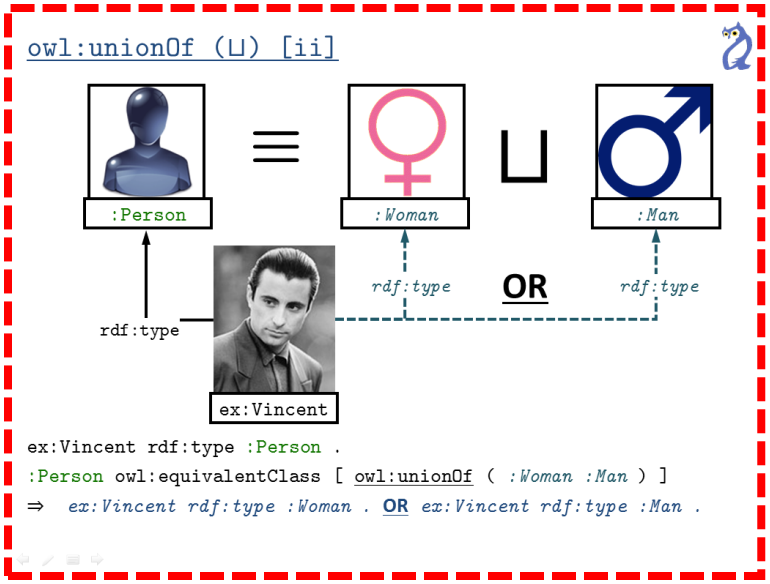
owl:unionOf (⊔) [ii]




```
ex:Vincent rdf:type :Person .
:Person owl:equivalentClass [ owl:unionOf ( :Woman :Man ) ]
⇒  ex:Vincent rdf:type :Woman . OR ex:Vincent rdf:type :Man .
```

Why Incomplete?

Incomplete for some Features



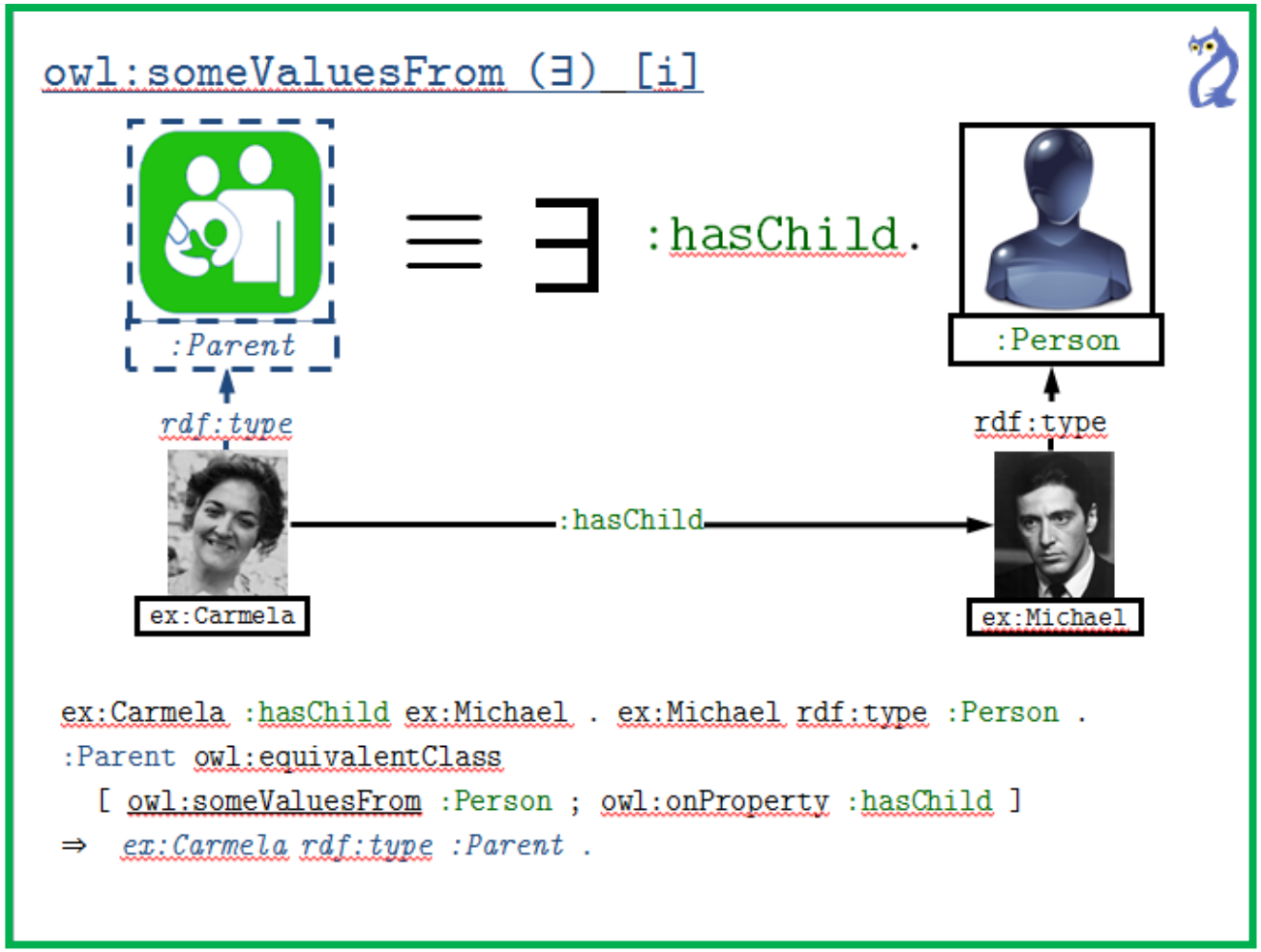

```
ex:Vincent rdf:type :Person , :Godfather .
:Godfather owl:disjointWith :Woman .
:Person owl:equivalentClass [ owl:unionOf ( :Woman :Man ) ]
⇒  ex:Vincent rdf:type :Man .
```

Will not get this (valid) entailment with OWL 2 RL/RDF

Why Incomplete?

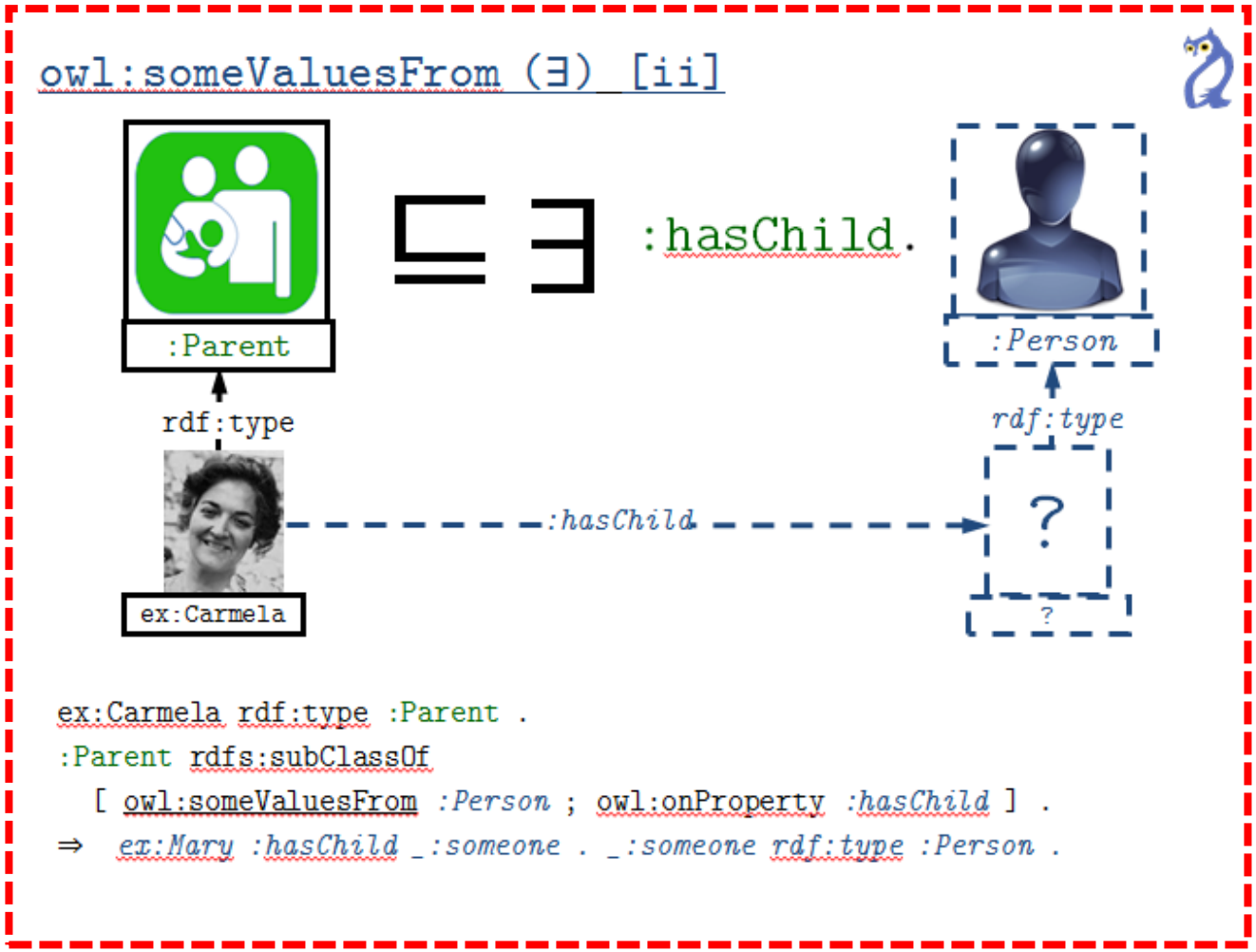
Incomplete for some Features

cls-svf1	T(?x, owl:someValuesFrom, ?y) T(?x, owl:onProperty, ?p) T(?u, ?p, ?v) T(?v, rdf:type, ?y)	T(?u, rdf:type, ?x)



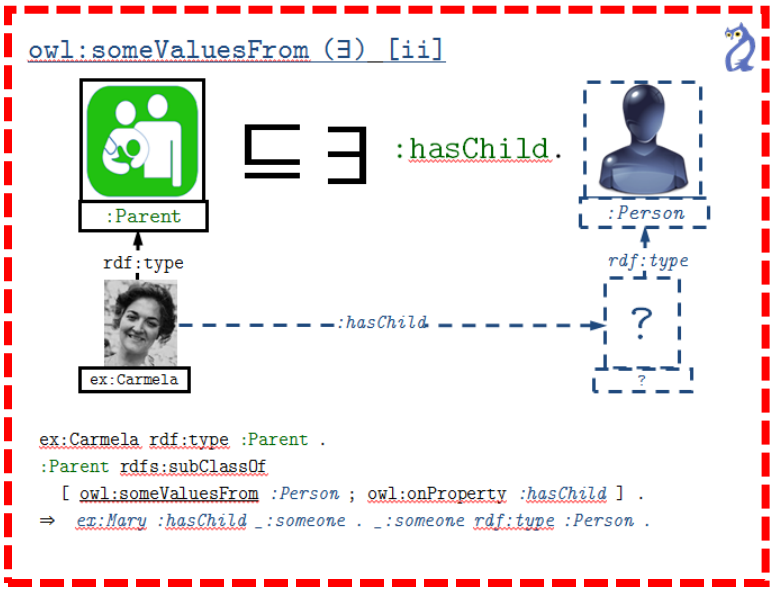
Why Incomplete?

Incomplete for some Features



Why Incomplete?

Incomplete for some Features




```
ex:Carmela rdf:type :Parent .
:Parent rdfs:subClassOf
  [ owl:someValuesFrom :Person ; owl:onProperty :hasChild ] .
:hasChild rdfs:domain :PostPuberty .
⇒ ex:Carmela rdf:type :PostPuberty .
```

Will not get this (valid) entailment with OWL 2 RL/RDF

Why Incomplete?

Incomplete for some Features

- No support for min-cardinality

```
ex:Carmela :hasChild ex:Sonny , ex:Connie , ex:Fredo , ex:Michael .
ex:Sonny :dateOfBirth “1916-07-23”^^xsd:date .
ex:Connie :dateOfBirth “1922-04-18”^^xsd:date .
ex:Fredo :dateOfBirth “1919-01-08”^^xsd:date .
ex:Michael :dateOfBirth “1920-11-15”^^xsd:date .
:dateOfBirth rdf:type owl:FunctionalProperty .
[ owl:minCardinality 3 ; owl:onProperty :hasChild ] rdfs:subClassOf
   :StressedParent .
⇒ ex:Carmela rdf:type :StressedParent .
```

Will not get this (valid) entailment with OWL 2 RL/RDF

Why Incomplete?

Incomplete for some Features

- Limited support for max-cardinality

cls-maxc1	T(?x, owl:maxCardinality, "0"^^xsd:nonNegativeInteger) T(?x, owl:onProperty, ?p) T(?u, rdf:type, ?x) T(?u, ?p, ?y)	false
cls-maxc2	T(?x, owl:maxCardinality, "1"^^xsd:nonNegativeInteger) T(?x, owl:onProperty, ?p) T(?u, rdf:type, ?x) T(?u, ?p, $?y_1$) T(?u, ?p, $?y_2$)	T($?y_1$, owl:sameAs, $?y_2$)

```
ex:Vincent rdf:type :Person ; :hasParent ex:Lucy, ex:Sonny, ex:Santino
   .
:Person rdfs:subClassOf [ owl:maxCardinality 2 ; owl:onProperty
   :hasParent ] .
ex:Lucy a :Woman . ex:Sonny a :Man . ex:Santino a :Man .
:Man owl:disjointWith :Woman .
⇒ ex:Sonny owl:sameAs ex:Santino .
```

Will not get this (valid) entailment with OWL 2 RL/RDF

Why Incomplete?

Incomplete for some Features

- No support for exact cardinality
- Support also limited for qualified cardinalities

Why Incomplete?

Missing Schema Inferences

- Just *some* missing examples for inverse-of:
  - `?a owl:inverseOf ?b . ⇒ ?b owl:inverseOf ?a .`
  - `?a owl:inverseOf ?a . ⇒ ?a rdf:type owl:SymmetricProperty .`
  - `?a rdf:type owl:SymmetricProperty . ⇒ ?a owl:inverseOf ?a .`
  - `?a owl:inverseOf ?b . ?b owl:inverseOf ?a . ⇒ ?a owl:equivalentProperty ?b .`
  - `?a owl:inverseOf ?b . ?b rdf:type owl:TransitiveProperty . ⇒ ?a rdf:type owl:TransitiveProperty .`
  - `?a owl:inverseOf ?b . ?b rdf:type owl:FunctionalProperty . ⇒ ?b rdf:type owl:InverseFunctionalProperty .`
  - `?a owl:inverseOf ?b . ?b rdfs:domain ?c . ⇒ ?a rdfs:range ?c .`
  - …

Why is OWL 2 RL/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`, …
- Problems with negation
  - `owl:disjointWith`, `owl:propertyDisjointWith`, `owl:complementOf` …
- Incomplete “schema” inferences

Finite rules not enough

- Could write a rule for any non-existential case

```
ex:Vincent rdf:type :Person ; :hasParent ex:Lucy, ex:Sonny, ex:Santino .
:Person rdfs:subClassOf [ owl:maxCardinality 2 ; owl:onProperty :hasParent
   ] .
ex:Lucy a :Woman . ex:Sonny a :Man . ex:Santino a :Man .
:Man owl:disjointWith :Woman .
⇒ ex:Sonny owl:sameAs ex:Santino .
```

```
?w rdf:type ?c ; ?p ?x , ?y , ?z .
?c owl:maxCardinality 2 ; owl:onProperty ?p .
?x owl:differentFrom ?y , ?z .
⇒ ?y owl:sameAs ?z .
```

- Infinite such rules (have to stop somewhere)

Existential rules are dangerous

- Could write rules for existential cases too

```
ex:Mary rdf:type :Person .
:Person rdfs:subClassOf [ owl:someValuesFrom :Person ; owl:onProperty
   :hasParent ] .
⇒ ex:Mary :hasParent _:x1 . _:x1 rdf:type :Person .
_:x1 :hasParent _:x2 . _:x2 rdf:type :Person .
… ∞
```

```
?x rdf:type ?c .
?c owl:someValuesFrom ?d ; owl:onProperty ?p .
⇒ ?x ?p _:b . _:b rdf:type ?d .
```

- Might lead to materialising ∞ entailments
  - (In this case if `?x rdf:type ?d . ⇒ ?x rdf:type ?c .`)

COMPLETE REASONERS THAT MAY NOT HALT

Complete reasoners that may not halt:

Quite Practical!

- 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 ☹

Complete reasoners that may not halt:

Rare in practice

- 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

not going to talk about this but good to know about! ☺

RESTRICT OWL TO GUARANTEE DECIDABILITY

Recap …

- 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

Core idea:
Restrict OWL so that complete reasoning is decidable over any ontology written within those restrictions

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?

- Most commonly: give an algorithm that halts …

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!

Restrict OWL to guarantee decidability:

How to guarantee decidability?

- Focus on satisfiability/entailment checking
  - Recall: Can (usually) reduce entailment to satisfiability
    - (So long as we can do negation in the language)

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 "$O \cup \neg O'$" 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!

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

- What is restricted?

What's the entailment question?

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

← Must restrict something here (for example)

$$\ldots$$
$$D_k \sqsubseteq \forall r.\Big(\bigsqcup_{D' \in R(D_k)} D'\Big) \sqcap \forall a.\Big(\bigsqcup_{D' \in A(D_k)} D'\Big)$$
$$d \sqsubseteq a \circ r, \quad d \sqsubseteq r \circ a, \quad \mathsf{Func}(d)$$

$$D \equiv \bot$$

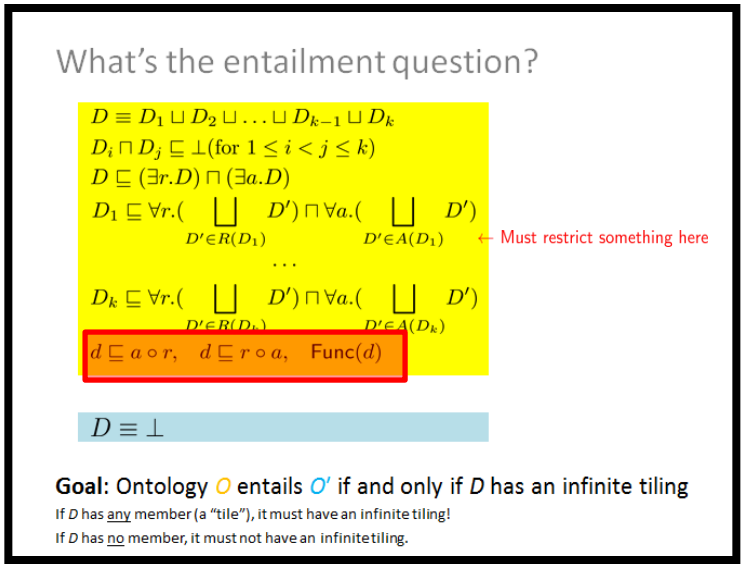
Goal: Ontology $\mathcal{O}$ entails $\mathcal{O}'$ if and only if D has an infinite tiling

If D has any member (a "tile"), it must have an infinite tiling!

If D has no member, it must not have an infinite tiling.

Restrict OWL to guarantee decidability:
Sublanguages of OWL 2

- What is restricted?



- For example, OWL 2 DL restricts:
  - functional properties to be "simple" (no chains, no transitivity)

Restrict OWL to guarantee decidability:
Sublanguages of OWL 2

- What is restricted?
- For example, OWL 2 DL restricts:
  - functional properties to be “simple” (no chains, no transitivity)
  - likewise properties used with hasSelf, cardinalities, inverse functionality, asymmetry and irreflexivity must be simple
  - inverse functional properties must be object properties
  - 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 ☹)

Restrict OWL to guarantee decidability:

On the plus side …

- OWL 2 DL still supports **disjunction**, existentials, counting, **negation**!

$$O \models O' \ ?$$

```
ex:Vincent rdf:type :Person , :Godfather .
:Godfather owl:disjointWith :Woman .
:Person owl:equivalentClass [ owl:unionOf ( :Woman :Man ) ] .
```

⊨ *ex:Vincent rdf:type :Man .*

Restrict OWL to guarantee decidability:

On the plus side …

- OWL 2 DL still supports disjunction, **existentials**, counting, negation!

$$\mathcal{O} \models \mathcal{O}' \ ?$$

```
ex:Carmela rdf:type :Parent .
:Parent rdfs:subClassOf
  [ owl:someValuesFrom :Person ; owl:onProperty :hasChild ] .
:hasChild rdfs:domain :PostPuberty .
```

⊨ *ex:Carmela rdf:type :PostPuberty .*

Restrict OWL to guarantee decidability:

On the plus side …

- OWL 2 DL still supports disjunction, existentials, **counting**, negation!

$O \models O'$?

```
ex:Carmela :hasChild ex:Sonny , ex:Connie , ex:Fredo , ex:Michael .
ex:Sonny :dateOfBirth “1916-07-23”^^xsd:date .
ex:Connie :dateOfBirth “1922-04-18”^^xsd:date .
ex:Fredo :dateOfBirth “1919-01-08”^^xsd:date .
ex:Michael :dateOfBirth “1920-11-15”^^xsd:date .
:dateOfBirth rdf:type owl:FunctionalProperty .
[ owl:minCardinality 3 ; owl:onProperty :hasChild ] rdfs:subClassOf
   :StressedParent .
```

⊨ *ex:Carmela rdf:type :StressedParent .*

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

Restrict OWL to guarantee decidability:

An algorithm for OWL 2 DL

- What sort of algorithm can we use?
- One answer: Tableau (positive *sketch* below)

```
ex:Vincent rdf:type :Person , :Godfather .
:Godfather owl:disjointWith :Woman .
:Person owl:equivalentClass [ owl:unionOf ( :Woman :Man )
```

⊨ *ex:Vincent rdf:type :Man .*

$O \models O'$?

Restrict OWL to guarantee decidability:

An algorithm for OWL 2 DL

- What sort of algorithm can we use?
- One answer: Tableau (positive *sketch* below)

```
ex:Vincent rdf:type :Person , :Godfather .
:Godfather owl:disjointWith :Woman .
:Person owl:equivalentClass [ owl:unionOf ( :Woman :Man )
    ] .
¬ ex:Vincent rdf:type :Man .
```

Unsatisfiable?

♂ Branch for OR ♀

```
ex:Vincent rdf:type :Person , :Godfather
    .
:Godfather owl:disjointWith :Woman .
ex:Vincent rdf:type Man .
¬ ex:Vincent rdf:type :Man .
```

```
ex:Vincent rdf:type :Person , :Godfather
    .
:Godfather owl:disjointWith :Woman .
ex:Vincent rdf:type Woman .
¬ ex:Vincent rdf:type :Man .
```

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

Restrict OWL to guarantee decidability:

An algorithm for OWL 2 DL

- What sort of algorithm can we use?
- One answer: Tableau (negative *sketch* below)

```
ex:Vincent rdf:type :Person , :Godfather .
:Godfather owl:disjointWith :Woman .
:Person owl:equivalentClass [ owl:unionOf ( :Woman :Man )
```

⊨ *ex:Vincent rdf:type :Woman .*

$O \models O'$?

Restrict OWL to guarantee decidability:

An algorithm for OWL 2 DL

- What sort of algorithm can we use?
- One answer: Tableau (negative *sketch* below)

```
ex:Vincent rdf:type :Person , :Godfather .
:Godfather owl:disjointWith :Woman .
:Person owl:equivalentClass [ owl:unionOf ( :Woman :Man )
    ] .
¬ ex:Vincent rdf:type :Woman .
```

Unsatisfiable?

♂ Branch for OR ♀

```
ex:Vincent rdf:type :Person , :Godfather
    .
:Godfather owl:disjointWith :Woman .
ex:Vincent rdf:type Man .
¬ ex:Vincent rdf:type :Woman .
```

OKAY

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

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

Restrict OWL to guarantee decidability:

An algorithm for OWL 2 DL!

- We have a complete algorithm that halts and that supports a lot of the OWL features!

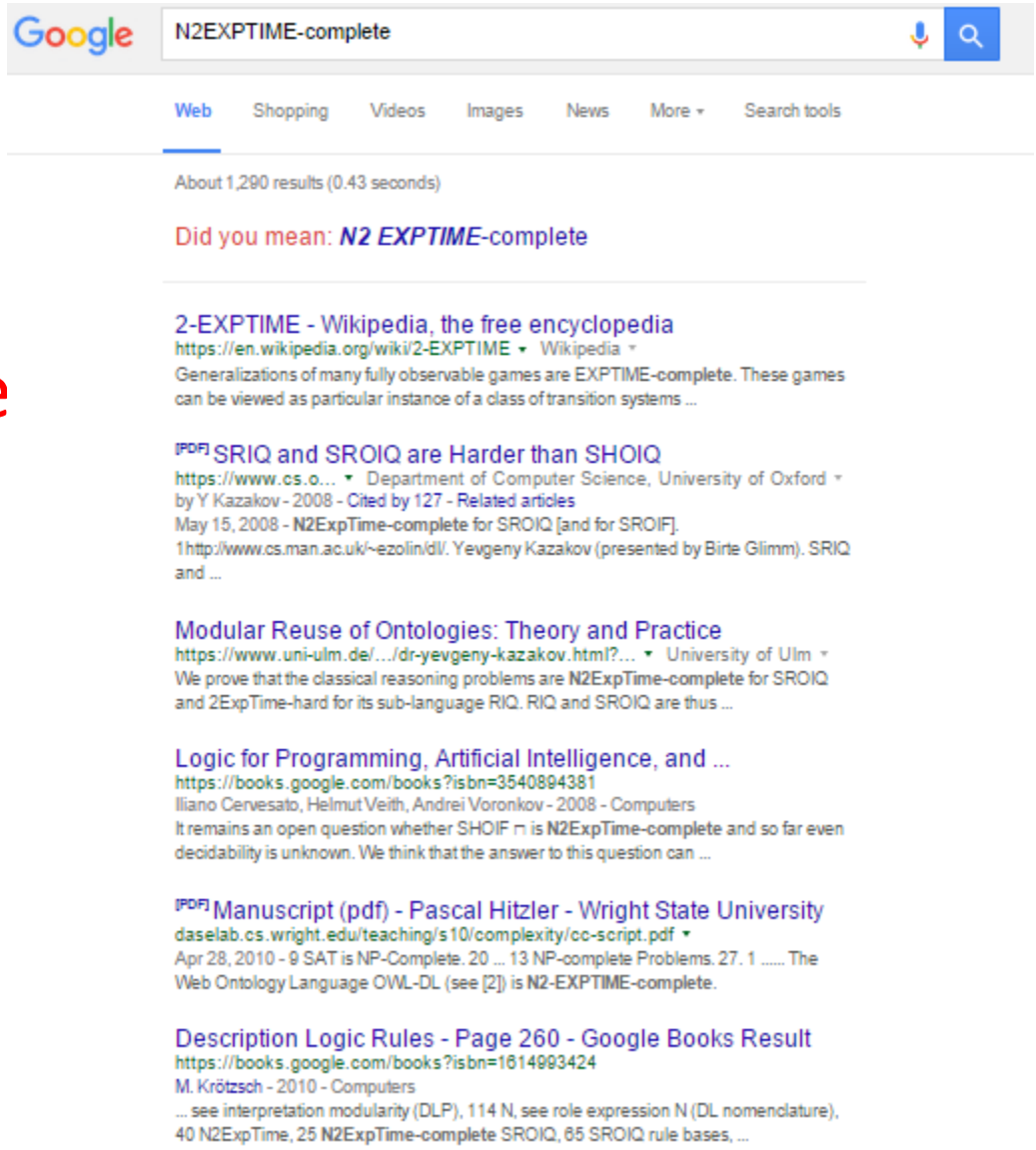
Restrict OWL to guarantee decidability:

An algorithm for OWL 2 DL!

- A few 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
  - Tableau reasoning is really expensive
    - Branch for every disjunction suggests exponential
    - N2EXPTIME-complete (!!?!!!)
      - 2^{2^n} on a non-deterministic machine
      - …

N2EXPTIME-Complete so nasty …

The only results returned by Google relate to OWL



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 DL performance considerations

- Not all OWL 2 DL ontologies will run into worst-cases
- Entailments will work fine for most small ontologies

- Scalability still a real issue in practice

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 (in some sense we won't get into)
- OWL 2 EL: Tractable algorithm for classifying ontologies
- OWL 2 QL: Tractable algorithm based on rewriting SQL queries

IMPRESSIONS …

Opinion of lots of people in the Semantic Web with respect to OWL …

Also perhaps part of the reason why you see things like …

Is OWL good for the Semantic Web?

- It provides formal foundations for semantics
- Indicates what's possible, what's not with respect to machine-readable semantics
  - What's efficient, what's not
- Offers options: OWL 2 RL/EL/QL/DL/Full

- Drives many applied/practical people crazy
- Some theoretical folks also consider it to have poor aesthetic
- Makes lots of bad assumptions for the Web
  - Not scalable
  - Strict in what it accepts
  - Blindly accepting

What do you think?

If we have time …

Let’s model a domain …

RECAP …

Coping with undecidability (reasoning) …

- Accept incomplete reasoners that halt
  - Complete language, incomplete reasoning, halts
  - e.g., OWL 2 RL/RDF rules can be applied on any RDF data using any OWL features in any way, but may not get all inferences
- Accept complete reasoners that may not halt
  - Complete language, complete reasoning, may not halt
  - e.g., can use a first-order-theorem prover, but it may run forever on some input ontologies
- Restrict OWL so reasoning becomes decidable
  - Restricted language, complete reasoning, halts
  - e.g., can restrict the OWL 2 Full language to sublanguages that have decidable/tractable reasoning algorithms

OWL 2 RL/RDF rules

- What we've been using in the labs
- Rules supporting a lot of OWL
  - but incomplete
- Can be run over any RDF/OWL data
  - no restrictions needed!
- Can materialise entailments
- *Relatively* efficient in practice
- Easy to implement, not so hard to understand

OWL 2 Full / Complete reasoning

- Not a lot of work
- One proposal using a First-Order-Logic theorem prover

OWL 2 DL

- Restrict OWL 2 Full to make entailment/satisfiability checking decidable
- Complete reasoning with respect to ontologies following restrictions
  - Supports some pretty complex entailments
  - Will always halt with a correct answer eventually
- Very bad worst-case: 2NEXPTIME-Complete
  - May not halt before end of universe
  - Worst-cases might be rare, but scalability and compute times still often encountered in practice
- Need to ask if something specific is entailed
  - Cannot materialise “all” entailments
- Restrictions make the whole thing nasty to understand

OWL 2 Profiles

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

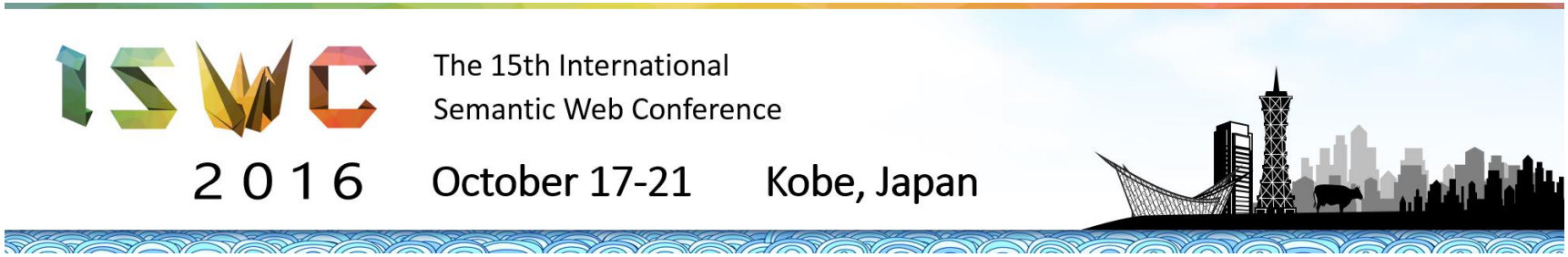
- OWL 2 RL: A restriction of OWL 2 DL such that OWL 2 RL/RDF rules provide complete reasoning (in some sense)
- OWL 2 EL: Tractable algorithm for classifying ontologies
- OWL 2 QL: Tractable algorithm based on rewriting SQL queries

End of main OWL part (after next lab)

… rest of material should be easier / more applied

(but I hope you learned something about why telling machines stuff about the world is hard)

No lecture/lab *next* week (Oct. 17/19)



Will post an assignment in the forum. ☺

Questions?