CC7220-1
LA WEB DE DATOS
PRIMAVERA 2019

LECTURE 5: WEB ONTOLOGY LANGUAGE (OWL) [II]

Aidan Hogan
aidhog@gmail.com

LAST TIME ...

SEMANTIC WEB: LOGIC

DATA:

Ireland

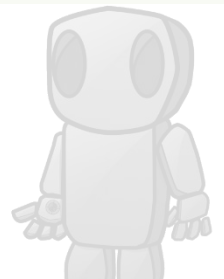(Ireland,partOf,Europe)
(Ireland,isA,Country)
(Ireland,capital,Dublin)

Dublin

(Ireland,capital,Dublin)
(Dublin,population,1000000)

LOGIC: "$(b,\text{capital},a) \rightarrow (a,\text{partOf},b)$"
"$(a,\text{partOf},b), (b,\text{partOf},c) \rightarrow (a,\text{partOf},c)$"

QUERY: "(x,partOf,y)?"

OUTPUT: $\{(x \mapsto \text{Ireland}, y \mapsto \text{Europe}),$
$(x \mapsto \text{Dublin}, y \mapsto \text{Ireland}),$
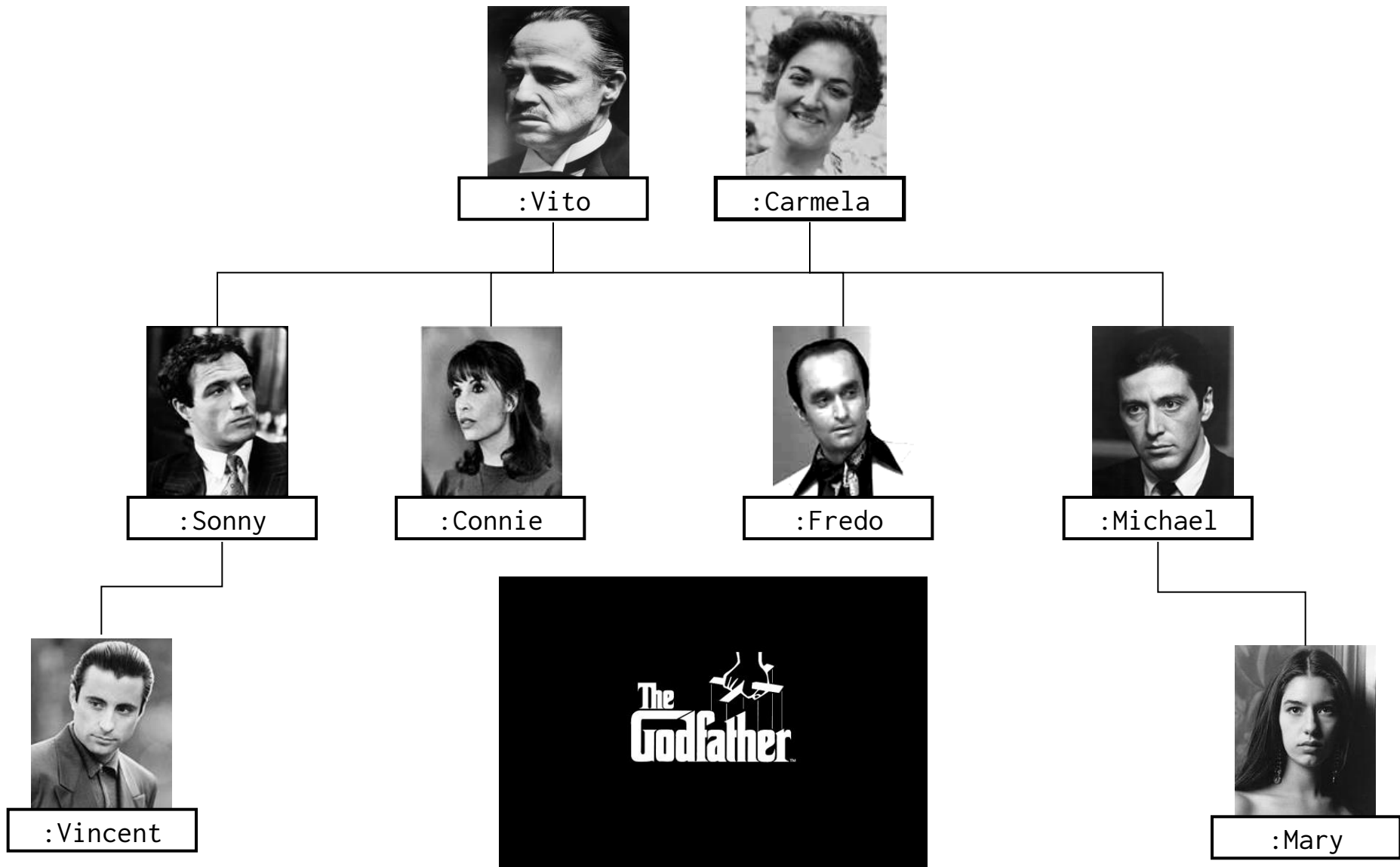$(x \mapsto \text{Dublin}, y \mapsto \text{Europe})\}$

← **RDFS**

← **OWL**

FAMILY RELATIONS IN 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 .
:Carmela :hasChild :Michael .
:Michael :hasDaughter :Mary .
: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?

Capture the main ideas of OWL

Fig 1. OWL according to the standard
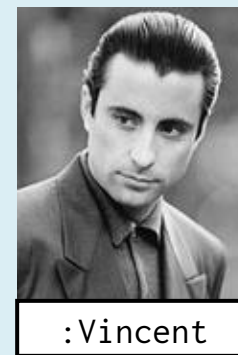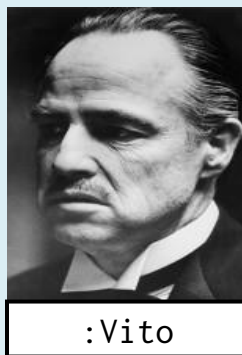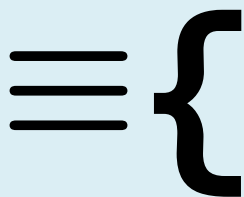

Fig 2. OWL according to this class

Defining OWL

owl:oneOf ({})

:Godfather owl:equivalentClass [owl:oneOf (:Vito :Michael :Vincent)]

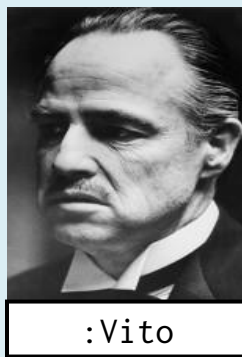:Godfather ≡ { :Vito , :Michael , :Vincent }

How can we formally define what this means?

owl:oneOf ({})

:Godfather owl:equivalentClass [owl:oneOf (:Vito :Michael :Vincent)]
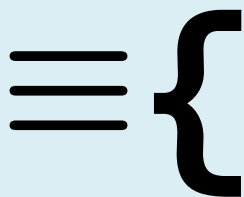
:Godfather

≡ { :Vito , :Michael , :Vincent }

rdf:type   rdf:type   rdf:type

We could try with rules like:

```
?c owl:equivalentClass [ owl:oneOf (?x1 <...> ?xn) ]
    → ?x1 a ?c . <...> ?xn a ?c .
```

owl:oneOf ({})

:Godfather owl:equivalentClass [owl:oneOf (:Vito :Michael :Vincent)]
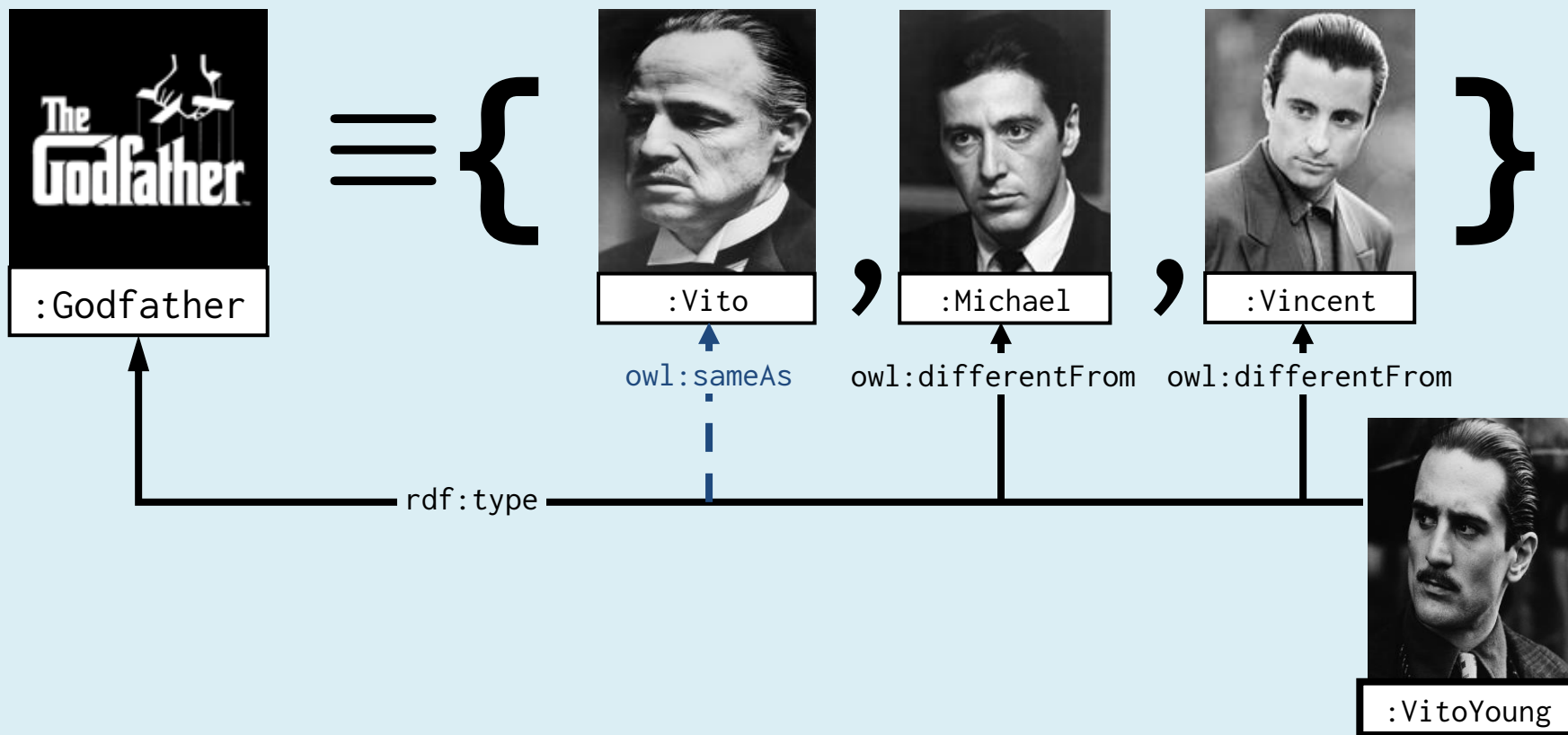
How do we define a rule for this case?

We could try a rule like:

?c owl:equivalentClass [owl:oneOf (?x1 <...> ?xn)] . ?x a ?c .
?x owl:differentFrom ?x2 , <...> , ?xn . → ?x owl:sameAs ?x1 .

owl:oneOf ({})

:Godfather owl:equivalentClass [owl:oneOf (:Vito :Michael :Vincent)]

How do we define a rule for this case?

What if it's not the first element?

Have we considered all possible cases like this?

owl:oneOf ({})

:Godfather owl:equivalentClass [owl:oneOf (:Vito :Michael :Vincent)]

How do we define a rule for this case?

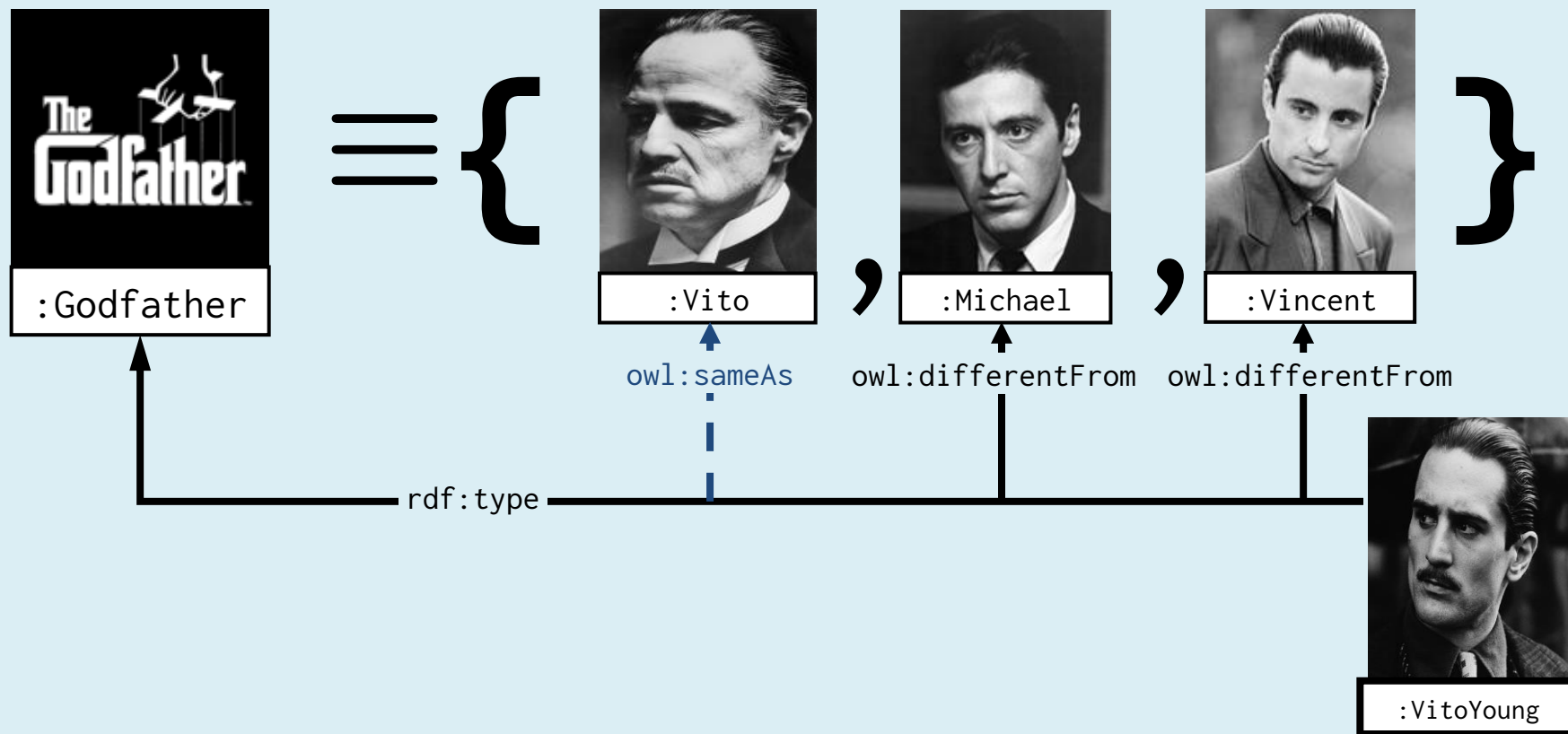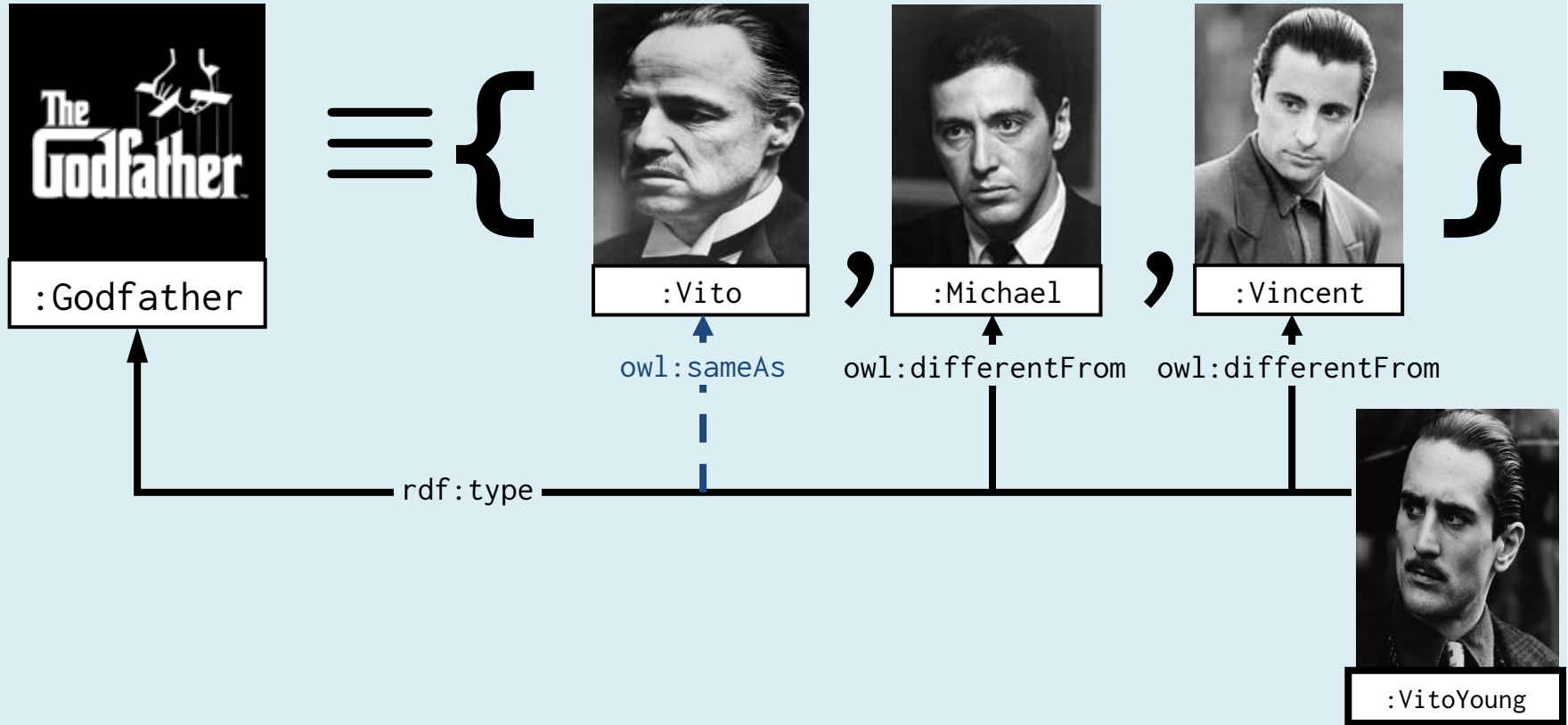A finite set of rules may not be able to define everything we need in OWL!

owl:oneOf ({})

`:Godfather owl:equivalentClass [ owl:oneOf (:Vito :Michael :Vincent) ]`

| :Godfather | ≡ { | :Vito | , | :Michael | , | :Vincent | } |
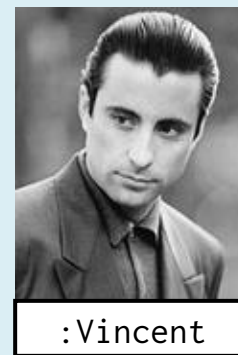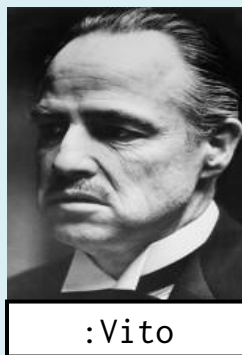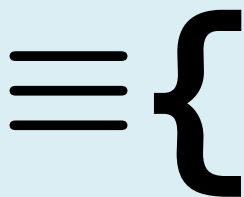
How can we begin to formally define what classes mean?

Use set theory:

There is a set of individuals (like , etc.).

There is a set of classes (like , etc.)

Classes map to sets of individuals (like ↦ { , , })

Names map to individuals (like `:Vito` ↦)

Other names map to classes (like `:Godfather` ↦)

owl:SymmetricProperty

:sibling rdf:type owl:SymmetricProperty .

:Connie :sibling→ :Fredo
:Connie ←:sibling-- :Fredo
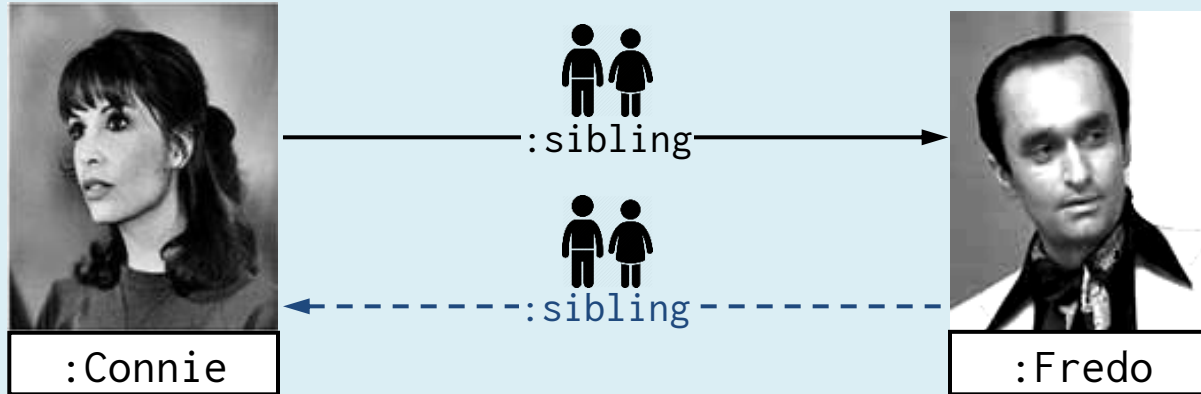
How can we begin to formally define what properties mean?

Extend the set theory:

There is a set of individuals. Classes map to sets of individuals.

There is a set of properties (like 👫, etc.).

Properties map to sets of pairs of individuals (like 👫 ↦ { (🖼 , 🖼) , … })

Names map to individuals, classes and or properties (like :sibling ↦ 👫)

The Universe

Define the universe as a set:

Set of everything: U

To Simplify matters

- Define the universe as a set:
  - Set of everything: U

What about classes and properties?

Classes are (sub)sets of U
Properties are sets of pairs from U

Vocabulary / Data

- Vocabulary (IRIs):   V

- Triples (RDF):   V × V × V
  - For brevity, we skip literals and blank nodes

- Graph (RDF): $2^{V \times V \times V}$

How do we relate the data to the universe?

INTERPRETATIONS

INTERPRETATION OF A VOCABULARY

- Interpretation $\mathcal{I}$ of vocabulary V: $\mathcal{I}$(V) = (U,I,C,P)
  - U: Universe / Set of everything (in the interpretation)
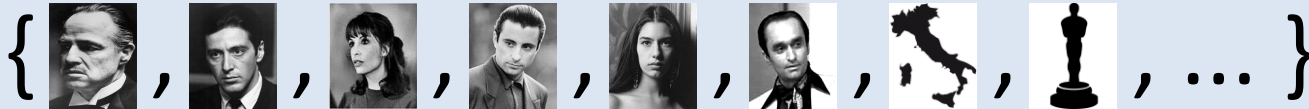
INTERPRETATION OF A VOCABULARY

- Interpretation $\mathcal{I}$ of vocabulary V: $\mathcal{I}$(V) = (U,I,C,P)

  U: Universe / Set of everything (in the interpretation)

  I: Maps a vocabulary term v ∈ V to an element of U

  (interpretation of Individuals)

I(:Mary) =   I(:Fredo) =   I(:Vito) =

I(:Italy) =   I(:Oscar) =   I(:Sonny) =

...

INTERPRETATION OF A VOCABULARY

- Interpretation $\mathcal{I}$ of vocabulary V: $\mathcal{I}$(V) = (U,I,C,P)

    U: Universe / Set of everything (in the interpretation)

    I: Maps a vocabulary term v ∈ V to an element of U

    (interpretation of Individuals)

    C: Maps a vocabulary term v ∈ V to a subset of U

    (interpretation of Classes)

C(:Person) =   { , , , , , , ... }

C(:Godfather) = { , , , ... }

...

INTERPRETATION OF A VOCABULARY

- Interpretation $\mathcal{I}$ of vocabulary V: $\mathcal{I}$(V) = (U,I,C,P)

  U: Universe / Set of everything (in the interpretation)

  I: Maps a vocabulary term v ∈ V to an element of U

  (interpretation of Individuals)

  C: Maps a vocabulary term v ∈ V to a subset of U

  (interpretation of Classes)

  P: Maps a vocabulary term v ∈ V to a set of pairs from U

  (interpretation of Properties)

`P(:Parent)=`   { (<image>, <image>), (<image>, <image>), ... }

`P(:Sibling)=`   { (<image>, <image>), (<image>, <image>), ... }

...

Interpretation of a vocabulary

- Interpretation $\mathcal{I}$ of vocabulary V: $\mathcal{I}$(V) = (U,I,C,P)

  U: Universe / Set of everything (in the interpretation)

  I: Maps a vocabulary term v ∈ V to an element of U

  (interpretation of Individuals)

  C: Maps a vocabulary term v ∈ V to a subset of U

  (interpretation of Classes)

  P: Maps a vocabulary term v ∈ V to a set of pairs from U

  (interpretation of Properties)

How do we interpret data/graphs and not just terms?

Models

MODELS

- Let G be an RDF graph using vocabulary V

- Let $\mathcal{I}$ be an interpretation of vocabulary V:
  - $\mathcal{I}(V) = (U,I,C,P)$

- $\mathcal{I}$ is a model of G if and only if:
  - If (s,p,o) ∈ G   then (I(s), I(o)) ∈ P(p)

How do we define the semantics of types?

MODELS (WITH CLASSES/TYPE/SUBCLASS)

- Let G be an RDF graph using vocabulary V

- Let $\mathcal{I}$ be an interpretation of vocabulary V:
  - $\mathcal{I}$(V) = (U,I,C,P)

- $\mathcal{I}$ is a model of G if and only if:
  - If (s,p,o) ∈ G   then (I(s), I(o)) ∈ P(p)
  - If (I(s), I(o)) ∈ P(`rdf:type`)   then I(s) ∈ C(o)

How do we define the semantics of sub-class?

MODELS (WITH RDFS SEMANTICS)

- Let G be an RDF graph using vocabulary V

- Let $\mathcal{I}$ be an interpretation of vocabulary V:
  - $\mathcal{I}(V) = (U,I,C,P)$

- $\mathcal{I}$ is a model of G if and only if:
  - If $(s,p,o) \in G$ then $(I(s), I(o)) \in P(p)$
  - If $(I(s), I(o)) \in P(\texttt{rdf:type})$ then $I(s) \in C(o)$
  - If $(I(s), I(o)) \in P(\texttt{rdfs:subClassOf})$ then $C(s) \subseteq C(o)$

How do we define the semantics of sub-property?

MODELS (WITH RDFS SEMANTICS)

- Let G be an RDF graph using vocabulary V

- Let $\mathcal{I}$ be an interpretation of vocabulary V:
  - $\mathcal{I}$(V) = (U,I,C,P)

- $\mathcal{I}$ is a model of G if and only if:
  - If (s,p,o) ∈ G   then (I(s), I(o)) ∈ P(p)
  - If (I(s), I(o)) ∈ P(`rdf:type`)   then I(s) ∈ C(o)
  - If (I(s), I(o)) ∈ P(`rdfs:subClassOf`)   then C(s) ⊆ C(o)
  - If (I(s), I(o)) ∈ P(`rdfs:subPropertyOf`)   then P(s) ⊆ P(o)

How do we define the semantics of domain/range?

MODELS (WITH RDFS SEMANTICS)

- Let G be an RDF graph using vocabulary V

- Let $\mathcal{I}$ be an interpretation of vocabulary V:
  - $\mathcal{I}(V) = (U,I,C,P)$

- $\mathcal{I}$ is a model of G if and only if:
  - If $(s,p,o) \in G$ then $(I(s), I(o)) \in P(p)$
  - If $(I(s), I(o)) \in P(\texttt{rdf:type})$ then $I(s) \in C(o)$
  - If $(I(s), I(o)) \in P(\texttt{rdfs:subClassOf})$ then $C(s) \subseteq C(o)$
  - If $(I(s), I(o)) \in P(\texttt{rdfs:subPropertyOf})$ then $P(s) \subseteq P(o)$
  - If $(I(s), I(o)) \in P(\texttt{rdfs:domain})$ then $\pi_1(P(s)) \subseteq C(o)$
  - If $(I(s), I(o)) \in P(\texttt{rdfs:range})$ then $\pi_2(P(s)) \subseteq C(o)$

$(\pi_n : \text{project } n^{\text{th}} \text{ element})$

MODELS (WITH OWL SEMANTICS)

- $\mathcal{I}$ is a model of G if and only if:
    - If $(s,p,o) \in G$ then $(I(s), I(o)) \in P(p)$
    - If $(I(s), I(o)) \in P(\texttt{rdf:type})$ then $I(s) \in C(o)$
    - If $(I(s), I(o)) \in P(\texttt{rdfs:subClassOf})$ then $C(s) \subseteq C(o)$
    - If $(I(s), I(o)) \in P(\texttt{rdfs:subPropertyOf})$ then $P(s) \subseteq P(o)$
    - If $(I(s), I(o)) \in P(\texttt{rdfs:domain})$ then $\pi_1(P(s)) \subseteq C(o)$
    - If $(I(s), I(o)) \in P(\texttt{rdfs:range})$ then $\pi_2(P(s)) \subseteq C(o)$

What about OWL semantics?

How do we define the semantics of same-as/different-from?

MODELS (WITH OWL SEMANTICS)

- $\mathcal{I}$ is a model of G if and only if:
  - If (s,p,o) ∈ G   then (I(s), I(o)) ∈ P(p)
  - If (I(s), I(o)) ∈ P(`rdf:type`)   then I(s) ∈ C(o)
  - If (I(s), I(o)) ∈ P(`rdfs:subClassOf`)   then C(s) ⊆ C(o)
  - If (I(s), I(o)) ∈ P(`rdfs:subPropertyOf`)   then P(s) ⊆ P(o)
  - If (I(s), I(o)) ∈ P(`rdfs:domain`)   then π_1(P(s)) ⊆ C(o)
  - If (I(s), I(o)) ∈ P(`rdfs:range`)   then π_2(P(s)) ⊆ C(o)
  - If (I(s), I(o)) ∈ P(`owl:sameAs`)   then I(s) = I(o)
  - If (I(s), I(o)) ∈ P(`owl:differentFrom`)   then I(s) ≠ I(o)

How do we define inverse properties?

MODELS (WITH OWL SEMANTICS)

- $\mathcal{I}$ is a model of G if and only if:
  - If (s,p,o) ∈ G   then (I(s), I(o)) ∈ P(p)
  - If (I(s), I(o)) ∈ P(`rdf:type`)   then I(s) ∈ C(o)
  - If (I(s), I(o)) ∈ P(`rdfs:subClassOf`)   then C(s) ⊆ C(o)
  - If (I(s), I(o)) ∈ P(`rdfs:subPropertyOf`) then P(s) ⊆ P(o)
  - If (I(s), I(o)) ∈ P(`rdfs:domain`)   then π_1(P(s)) ⊆ C(o)
  - If (I(s), I(o)) ∈ P(`rdfs:range`)   then π_2(P(s)) ⊆ C(o)
  - If (I(s), I(o)) ∈ P(`owl:sameAs`)   then I(s) = I(o)
  - If (I(s), I(o)) ∈ P(`owl:differentFrom`)   then I(s) ≠ I(o)
  - If (I(s), I(o)) ∈ P(`owl:inverseOf`)   then ∀x,y : (x,y) ∈ P(s)
                                               if and only if (y,x) ∈ P(o)

How do we define transitive properties?

MODELS (WITH OWL SEMANTICS)

- $\mathcal{I}$ is a model of G if and only if:
  - If (s,p,o) ∈ G   then (I(s), I(o)) ∈ P(p)
  - If (I(s), I(o)) ∈ P(`rdf:type`)   then I(s) ∈ C(o)
  - If (I(s), I(o)) ∈ P(`rdfs:subClassOf`)   then C(s) ⊆ C(o)
  - If (I(s), I(o)) ∈ P(`rdfs:subPropertyOf`) then P(s) ⊆ P(o)
  - If (I(s), I(o)) ∈ P(`rdfs:domain`)   then π_1(P(s)) ⊆ C(o)
  - If (I(s), I(o)) ∈ P(`rdfs:range`)   then π_2(P(s)) ⊆ C(o)
  - If (I(s), I(o)) ∈ P(`owl:sameAs`)   then I(s) = I(o)
  - If (I(s), I(o)) ∈ P(`owl:differentFrom`) then I(s) ≠ I(o)
  - If (I(s), I(o)) ∈ P(`owl:inverseOf`)   then ∀x,y : (x,y) ∈ P(s)
                                               if and only if (y,x) ∈ P(o)
  - If I(s) ∈ C(`owl:TransitiveProperty`)   then ∀x,y,z : (x,y) ∈ P(s) and
                                               (y,z) ∈ P(s) imply (x,z) ∈ P(s)

How do we define disjoint classes?

MODELS (WITH OWL SEMANTICS)

- $\mathcal{I}$ is a model of G if and only if:
  - If (s,p,o) ∈ G   then (I(s), I(o)) ∈ P(p)
  - If (I(s), I(o)) ∈ P(`rdf:type`)   then I(s) ∈ C(o)
  - If (I(s), I(o)) ∈ P(`rdfs:subClassOf`)   then C(s) ⊆ C(o)
  - If (I(s), I(o)) ∈ P(`rdfs:subPropertyOf`) then P(s) ⊆ P(o)
  - If (I(s), I(o)) ∈ P(`rdfs:domain`)   then π_1(P(s)) ⊆ C(o)
  - If (I(s), I(o)) ∈ P(`rdfs:range`)   then π_2(P(s)) ⊆ C(o)
  - If (I(s), I(o)) ∈ P(`owl:sameAs`)   then I(s) = I(o)
  - If (I(s), I(o)) ∈ P(`owl:differentFrom`) then I(s) ≠ I(o)
  - If (I(s), I(o)) ∈ P(`owl:inverseOf`)   then ∀x,y : (x,y) ∈ P(s)
                                        if and only if (y,x) ∈ P(o)
  - If I(s) ∈ C(`owl:TransitiveProperty`)   then ∀x,y,z : (x,y) ∈ P(s) and
                                        (y,z) ∈ P(s) imply (x,z) ∈ P(s)
  - If (I(s), I(o)) ∈ P(`owl:disjointWith`)   then C(s) ∩ C(o) = Ø

... and so on.

Aside:
If-then vs. If-and-only-if semantics

If-then semantics

- $\mathcal{I}$ is a model of G if and only if:
  - If (s,p,o) ∈ G   then (I(s), I(o)) ∈ P(p)
  - If (I(s), I(o)) ∈ P(`rdf:type`)   then I(s) ∈ C(o)
  - If (I(s), I(o)) ∈ P(`rdfs:subClassOf`)   then C(s) ⊆ C(o)
  - If (I(s), I(o)) ∈ P(`rdfs:subPropertyOf`)   then P(s) ⊆ P(o)
  - If (I(s), I(o)) ∈ P(`rdfs:domain`)   then π_1(P(s)) ⊆ C(o)
  - If (I(s), I(o)) ∈ P(`rdfs:range`)   then π_2(P(s)) ⊆ C(o)
  - If (I(s), I(o)) ∈ P(`owl:sameAs`)   then I(s) = I(o)
  - If (I(s), I(o)) ∈ P(`owl:differentFrom`)   then I(s) ≠ I(o)
  - If (I(s), I(o)) ∈ P(`owl:inverseOf`)   then ∀x,y : (x,y) ∈ P(s)
                                                     if and only if (y,x) ∈ P(o)
  - If I(s) ∈ C(`owl:TransitiveProperty`)   then ∀x,y,z : (x,y) ∈ P(s) and
                                                     (y,z) ∈ P(s) imply (x,z) ∈ P(s)
  - If (I(s), I(o)) ∈ P(`owl:disjointWith`)   then C(s) ∩ C(o) = ∅

Some data condition   ⇒   Some set condition

IF-AND-ONLY-IF SEMANTICS

- $\mathcal{I}$ is a model of G if and only if:
  - if (s,p,o) ∈ G   then (I(s), I(o)) ∈ P(p)
  - (I(s), I(o)) ∈ P(`rdf:type`)   if and only if I(s) ∈ C(o)
  - (I(s), I(o)) ∈ P(`rdfs:subClassOf`)   if and only if C(s) ⊆ C(o)
  - (I(s), I(o)) ∈ P(`rdfs:subPropertyOf`)   if and only if P(s) ⊆ P(o)
  - (I(s), I(o)) ∈ P(`rdfs:domain`)   if and only if π_1(P(s)) ⊆ C(o)
  - (I(s), I(o)) ∈ P(`rdfs:range`)   if and only if π_2(P(s)) ⊆ C(o)
  - (I(s), I(o)) ∈ P(`owl:sameAs`)   if and only if I(s) = I(o)
  - (I(s), I(o)) ∈ P(`owl:differentFrom`)   if and only if I(s) ≠ I(o)
  - (I(s), I(o)) ∈ P(`owl:inverseOf`)   if and only if ∀x,y : (x,y) ∈ P(s)
                                                    if and only if (y,x) ∈ P(o)
  - I(s) ∈ C(`owl:TransitiveProperty`)   if and only if ∀x,y,z : (x,y) ∈ P(s) and
                                                    (y,z) ∈ P(s) imply (x,z) ∈ P(s)
  - (I(s), I(o)) ∈ P(`owl:disjointWith`)   if and only if C(s) ∩ C(o) = ∅

Some data condition ⟺ **Some set condition**

WHAT'S THE DIFFERENCE?

G = {(x,`rdfs:subClassOf`,y),(y,`rdfs:subClassOf`,z)}

IF-THEN SEMANTICS

⇒ (I(x), I(y)) ∈ P(`rdfs:subClassOf`) ⇒ C(x) ⊆ C(y)

⇒ (I(y), I(z)) ∈ P(`rdfs:subClassOf`) ⇒ C(y) ⊆ C(z)

⇒ C(x) ⊆ C(z)

IF-AND-ONLY-IF SEMANTICS

⇒ (I(x), I(y)) ∈ P(`rdfs:subClassOf`) ⇒ C(x) ⊆ C(y)

⇒ (I(y), I(z)) ∈ P(`rdfs:subClassOf`) ⇒ C(y) ⊆ C(z)

⇒ C(x) ⊆ C(z)

⇒ (I(x), I(z)) ∈ P(`rdfs:subClassOf`)

(Back to) Models

A __model__ is any world that an ontology might describe

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


Is this a model of the ontology?

No, since (:Carmela) needs to be an (:ancestorOf) (:Mary)

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 .
:Carmela :hasChild :Michael .
:Michael :hasDaughter :Mary .
```


Now we have a model of the ontology (for the given semantics).

MODELS OF ONTOLOGIES

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 .

:Carmela :hasChild :Michael .

:Michael :hasDaughter :Mary .
```
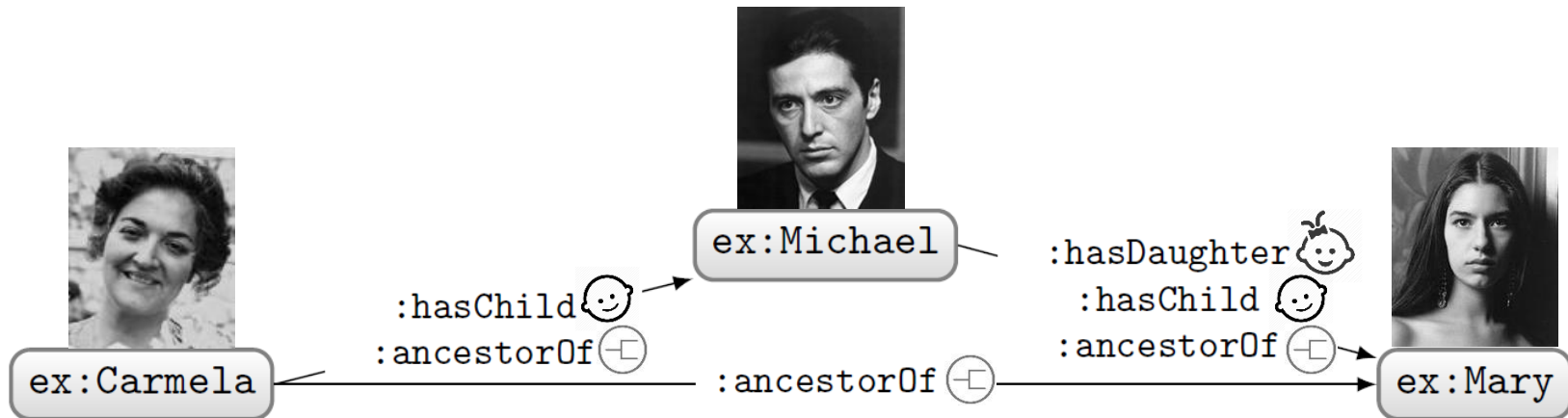

ex:Cake

:likes 👍

ex:Michael

:hasDaughter
:hasChild

:hasChild
:ancestorOf

:ancestorOf

ex:Carmela

:ancestorOf
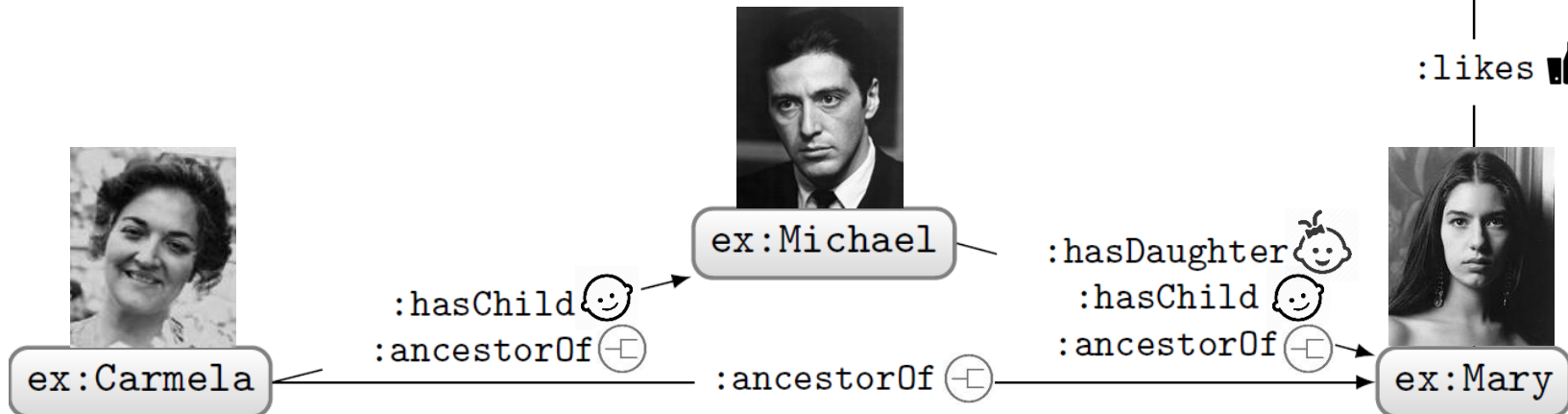
ex:Mary

Is this a model of the ontology?

This is also a model (given the Open World Assumption)!

MAPPING OF NAMES TO THINGS PART OF MODEL

ex:Cake

ex:Michael

ex:Carmela

ex:Mary

A different mapping would mean 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 .
:Carmela :hasChild :Michael .
:Michael :hasDaughter :Mary .
:Mary a :Person .
:Person owl:equivalentClass
  [ owl:qualifiedCardinality 2 ;
    owl:onProperty :hasParent ;
    owl:onClass :Person ] .
```

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

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

(*O'* forms part of the models of *O*)

(*O'* says nothing new over *O*)

$$O \models O'$$

ONTOLOGY ENTAILMENT

```
:Michael :hasParent :Carmela .

:Michael :hasChild :Mary .

:Carmela :ancestorOf :Mary .
```
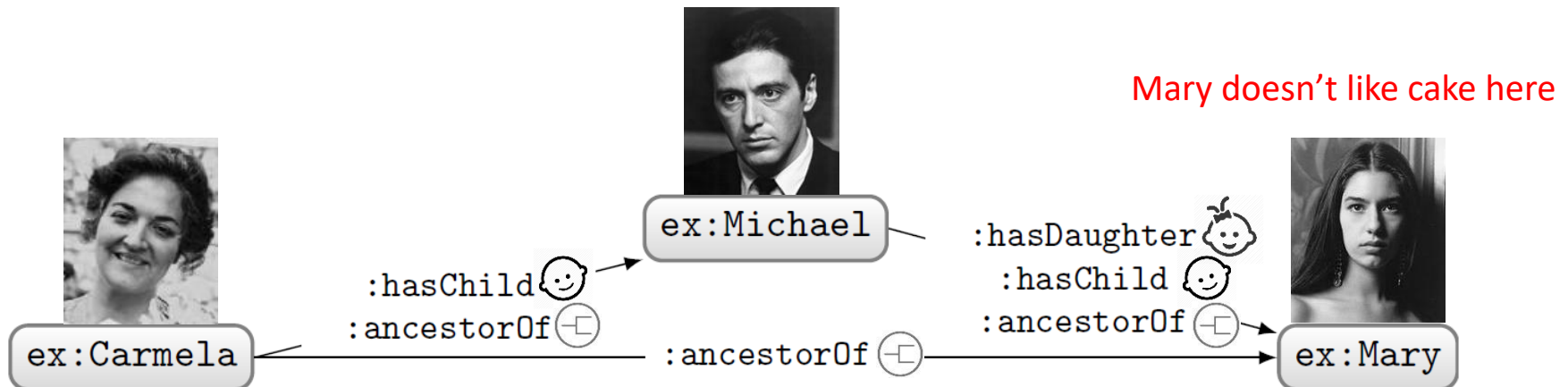
```
:Carmela :ancestorOf :Mary .

:Mary :likes :Cake .
```

Does *O* entail *O'* (*O* ⊨ *O'*)?

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

Mary doesn't like cake here

:hasChild

:ancestorOf

ex:Michael

:hasDaughter

:hasChild

:ancestorOf

ex:Carmela

:ancestorOf

ex:Mary

REASONING TASKS ...

Materialisation: Write down entailments

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

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

Any problems with this?

Materialisation: Write down entailments

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

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

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

Materialisation: Write down entailments

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

```
:Michael :hasParent :Carmela .
:Michael :hasChild :Mary .
:Carmela :ancestorOf :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 ] . ▬▬▬
```

MATERIALISATION: WRITE DOWN ENTAILMENTS

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


WELL, THAT ESCALATED QUICKLY

```
:Michael :hasParent :Carmela .
:Michael :hasChild :Mary .
:Carmela :ancestorOf :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 ] . ▪▪▪
```
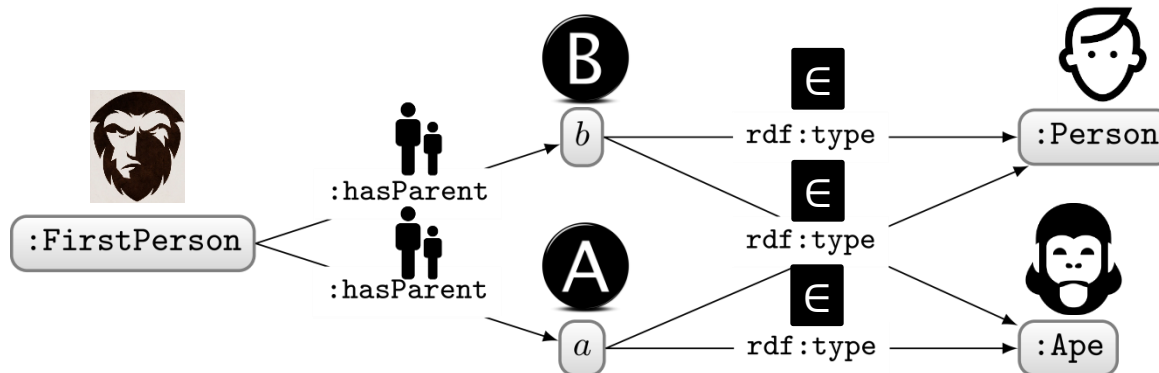
```
:Person owl:equivalentClass
  [ owl:qualifiedCardinality 2 ; owl:onProperty :hasParent ;
    owl:onClass :Person ] .
:FirstPerson a :Person ,
  [ owl:qualifiedCardinality 2 ; owl:onProperty :hasParent ;
    owl:onClass :Ape ] .
```

Does *O* have any model?

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

Ontology Satisfiability:
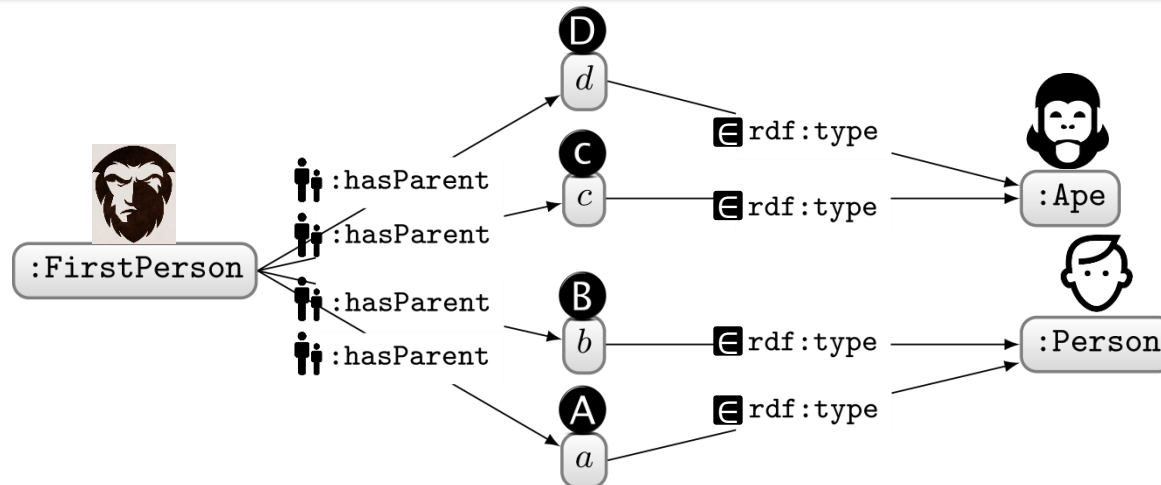
Does *O* have a "model"?

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

So does *O* have a model now?

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

```
: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**?

```
:Person rdfs:subClassOf
  [ owl:cardinality 2 ; owl:onProperty :hasParent ] .
```

OR

```
:Person owl:equivalentClass
  [ owl:allValuesFrom :Person ; owl:onProperty :hasParent ] .
```

OR

```
:FirstPerson a :Ape .
```

OR

```
:FirstPerson a owl:Nothing .
```

OR ...

```
:Person owl:equivalentClass
  [ owl:qualifiedCardinality 2 ; owl:onProperty :hasParent ;
```

An unsatisfiable ontology cannot model any world!

It is inconsistent!

OR

```
:FirstPerson a :Ape .
```

OR

```
:FirstPerson a owl:Nothing .
```

OR …

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

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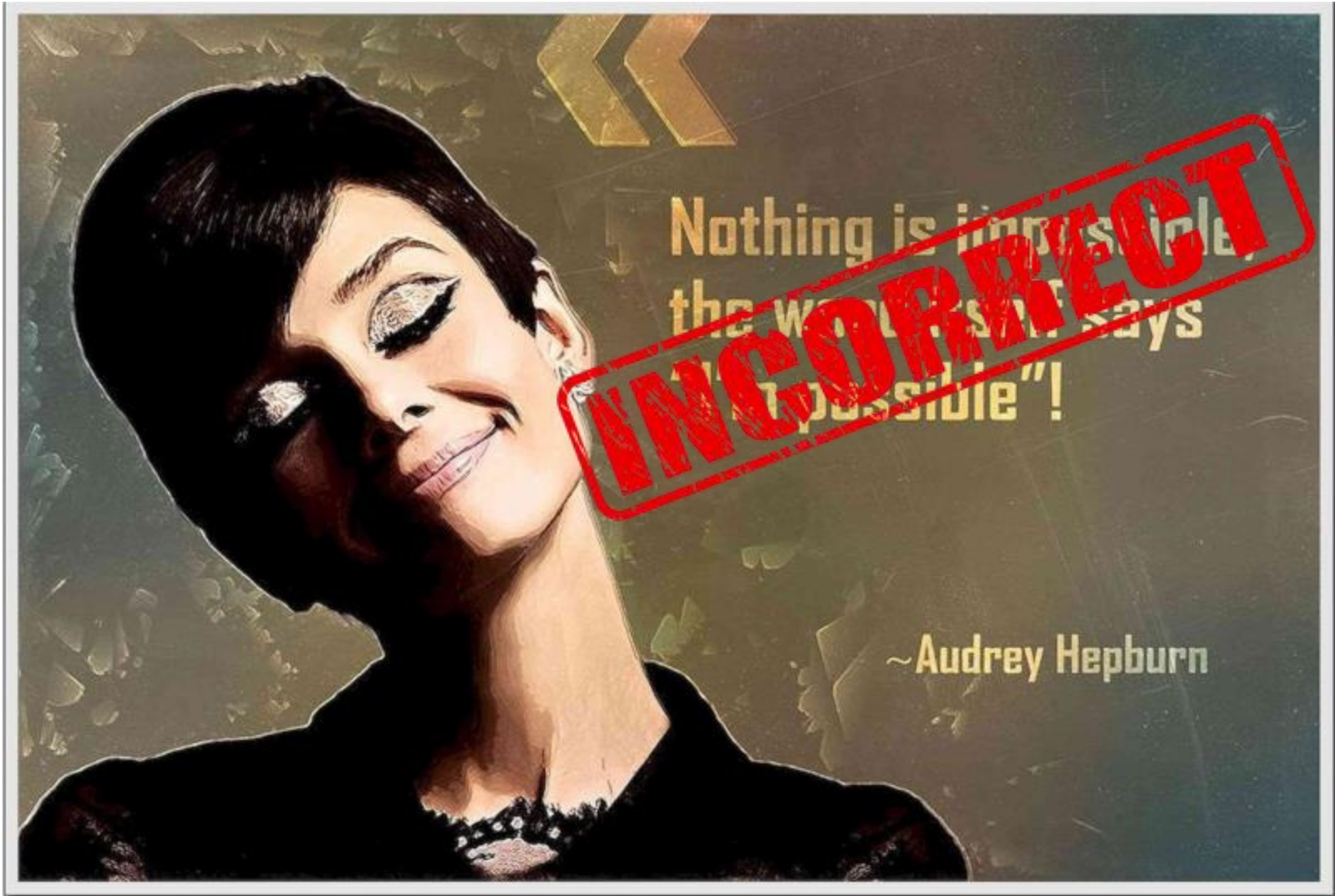
```
:Michael :hasChild :Mary .
```

- Could instead ask: is "*O* ∪ ¬*O'*" unsatisfiable?

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

Can reduce entailment to unsatisfiability!

So how do we test unsatisfiability then?

Undecidability …

A SIMPLE JAVA PROGRAM …

```java
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();
    }
}
```
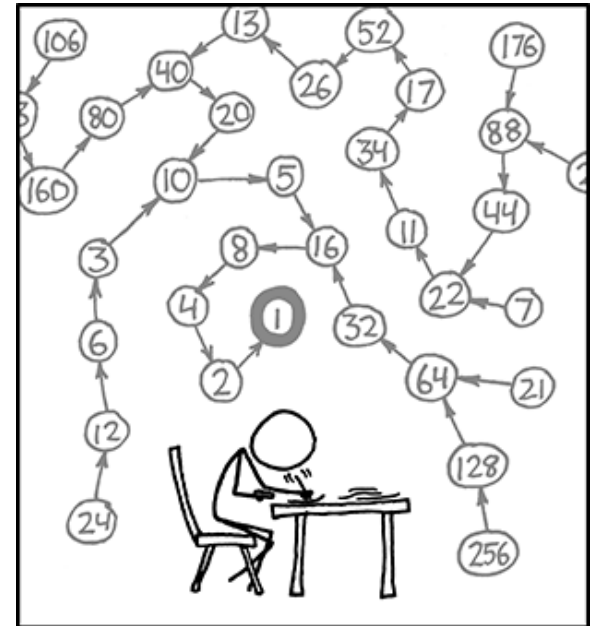
In: 6
3
10
5
16
8
4
2
1 **(end)**

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

A simple Java program …

```java
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();
    }
}
```


THE COLLATZ CONJECTURE STATES THAT IF YOU PICK A NUMBER, AND IF IT'S EVEN DIVIDE IT BY TWO AND IF IT'S ODD MULTIPLY IT BY THREE AND ADD ONE, AND YOU REPEAT THIS PROCEDURE LONG ENOUGH, EVENTUALLY YOUR FRIENDS WILL STOP CALLING TO SEE IF YOU WANT TO HANG OUT.

- 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
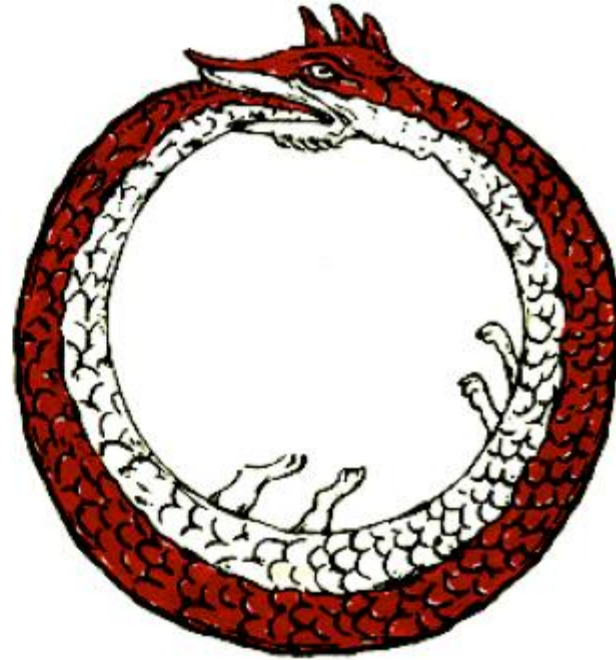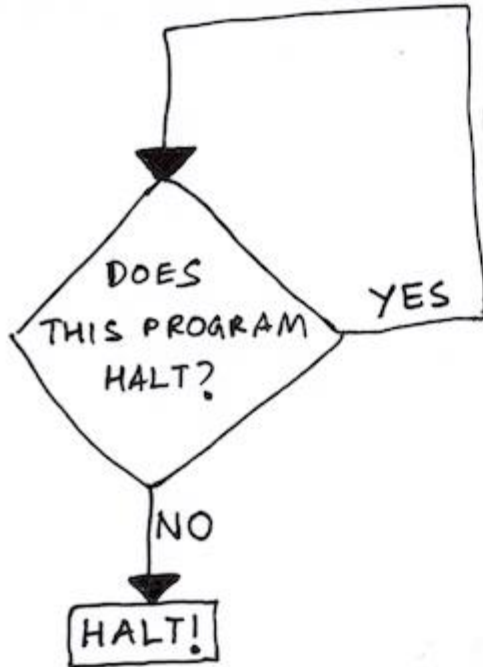
PROBLEM: CONSECUTIVE '1'S IN π

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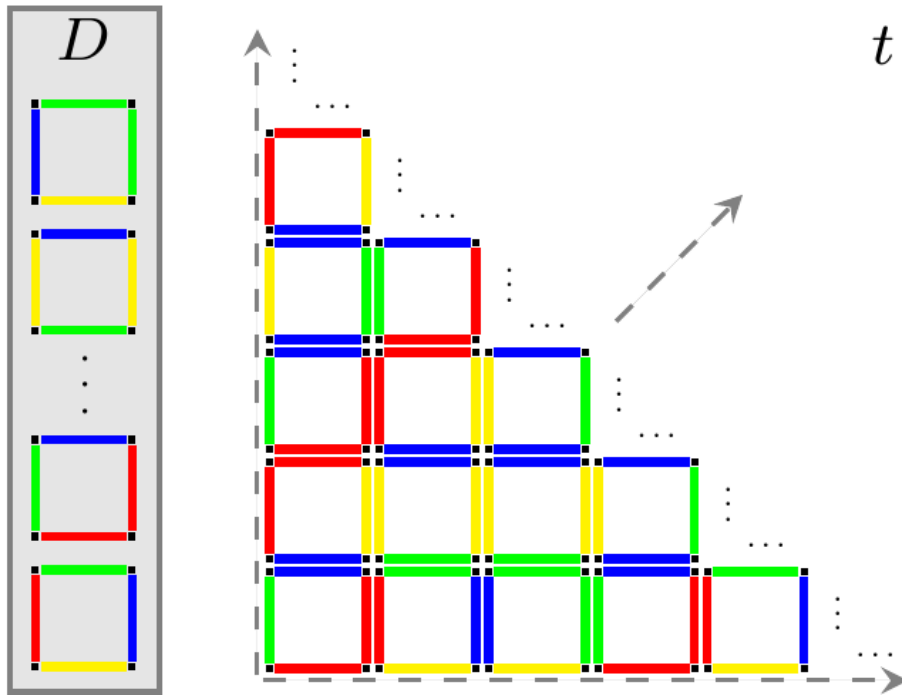
Is this problem **DECIDABLE** or **UNDECIDABLE**?

(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 <u>general</u> case
(for <u>all</u> programs and inputs)
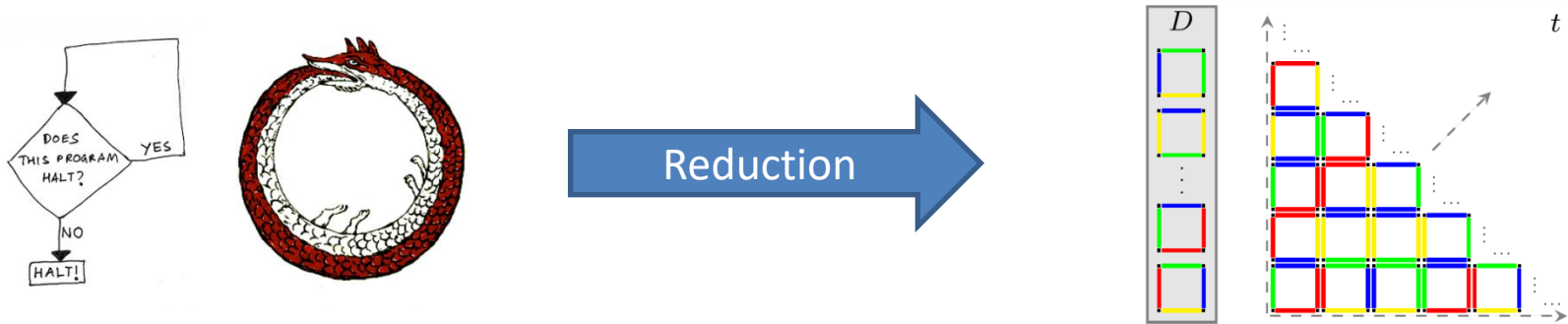
DOMINO TILING PROBLEM

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

Reduction

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?

Reduction
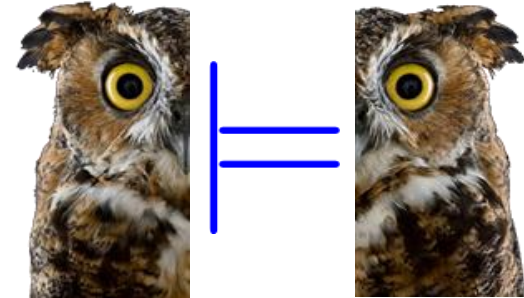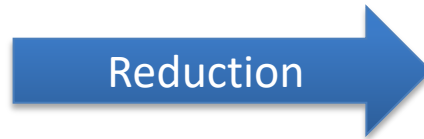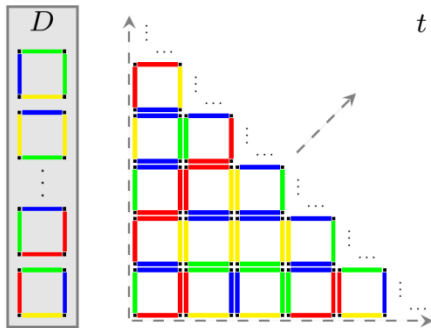
Does D have an infinite tiling?

Does OWL ontology *O* entail *O'*?

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

TO BE CONTINUED....➔

Questions?