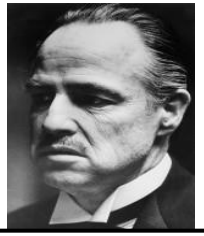
CC7220-1
La Web de Datos
Primavera 2022

Lecture 5: Web Ontology Language (OWL) [II]

Aidan Hogan
aidhog@gmail.com

Last time …

Family Relations in OWL



:Vito

:Carmela

:Sonny

:Connie

:Fredo

:Michael

:Vincent

:Mary

Today's Topic

An ontology is just some definitions …

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

… but what do they mean?

… and what can we do with ontologies?

Capture the main ideas of OWL

Fig 1. OWL according to the standard

Fig 2. OWL according to this class

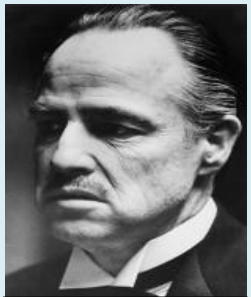




Defining OWL

owl:oneOf ({})

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

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

How can we formally define what this means?

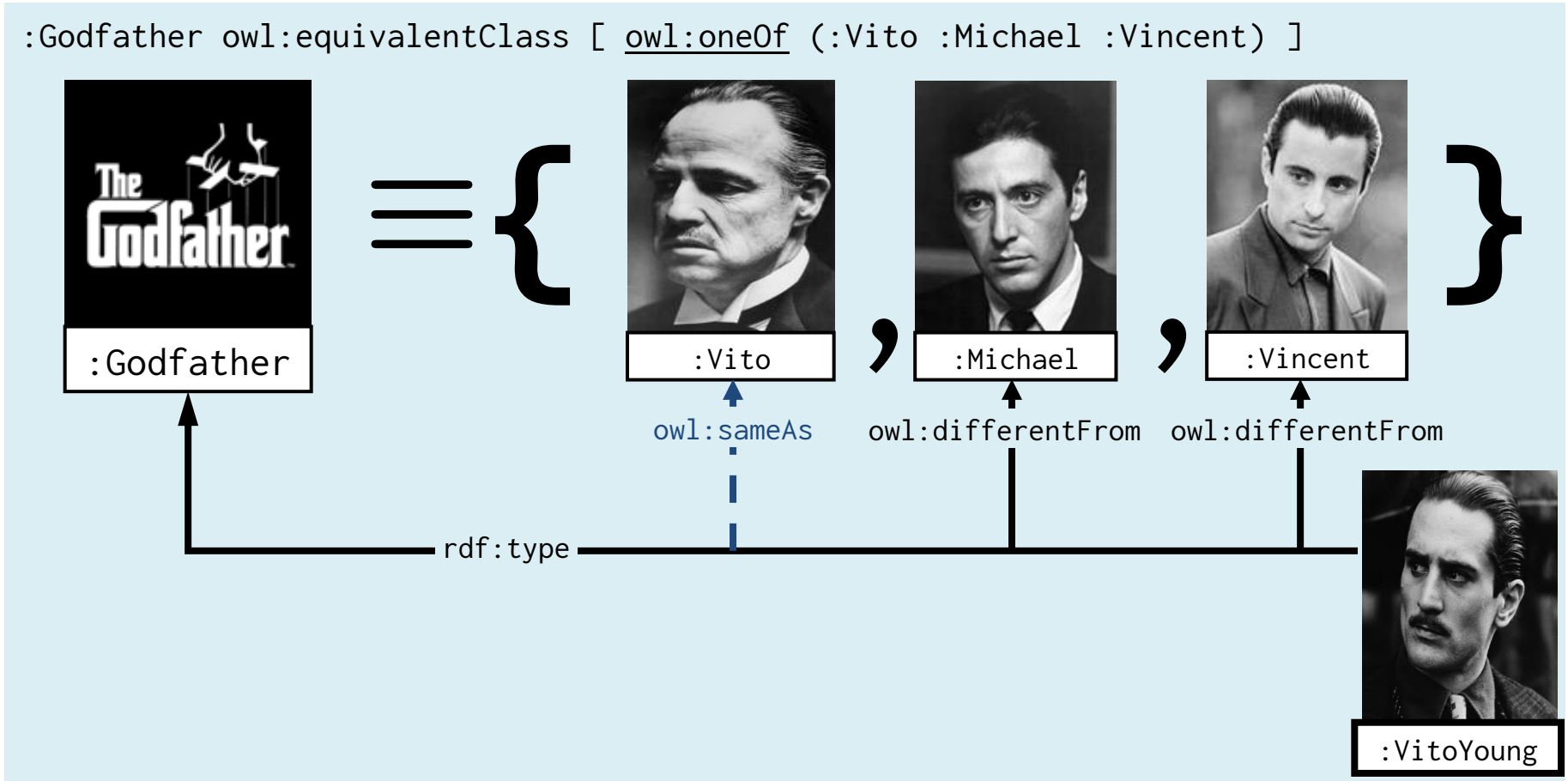
owl:oneOf ({})

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

We could try with rules like:

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

owl:oneOf ({})

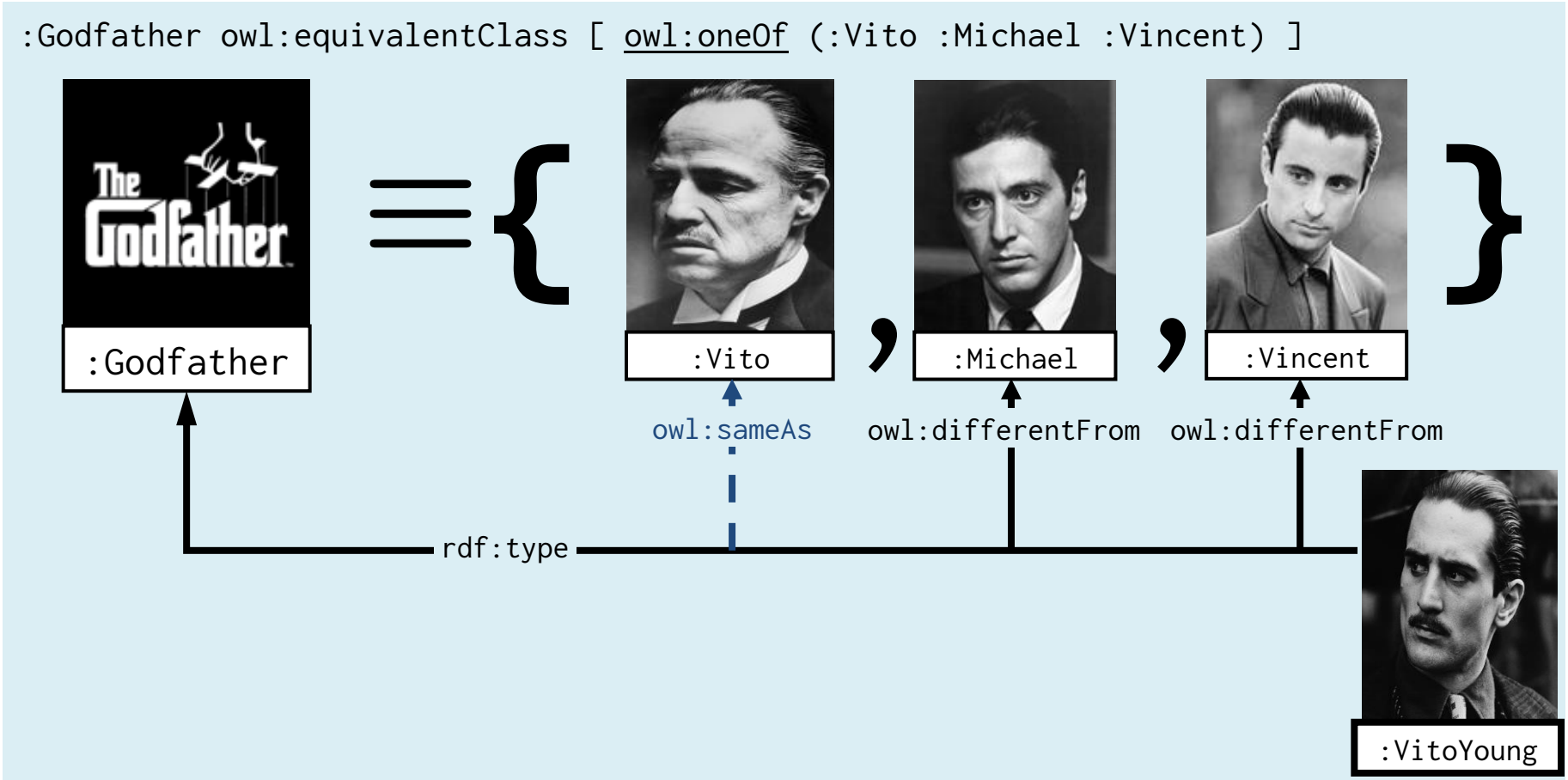


How do we define a rule for this case?

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 ({})

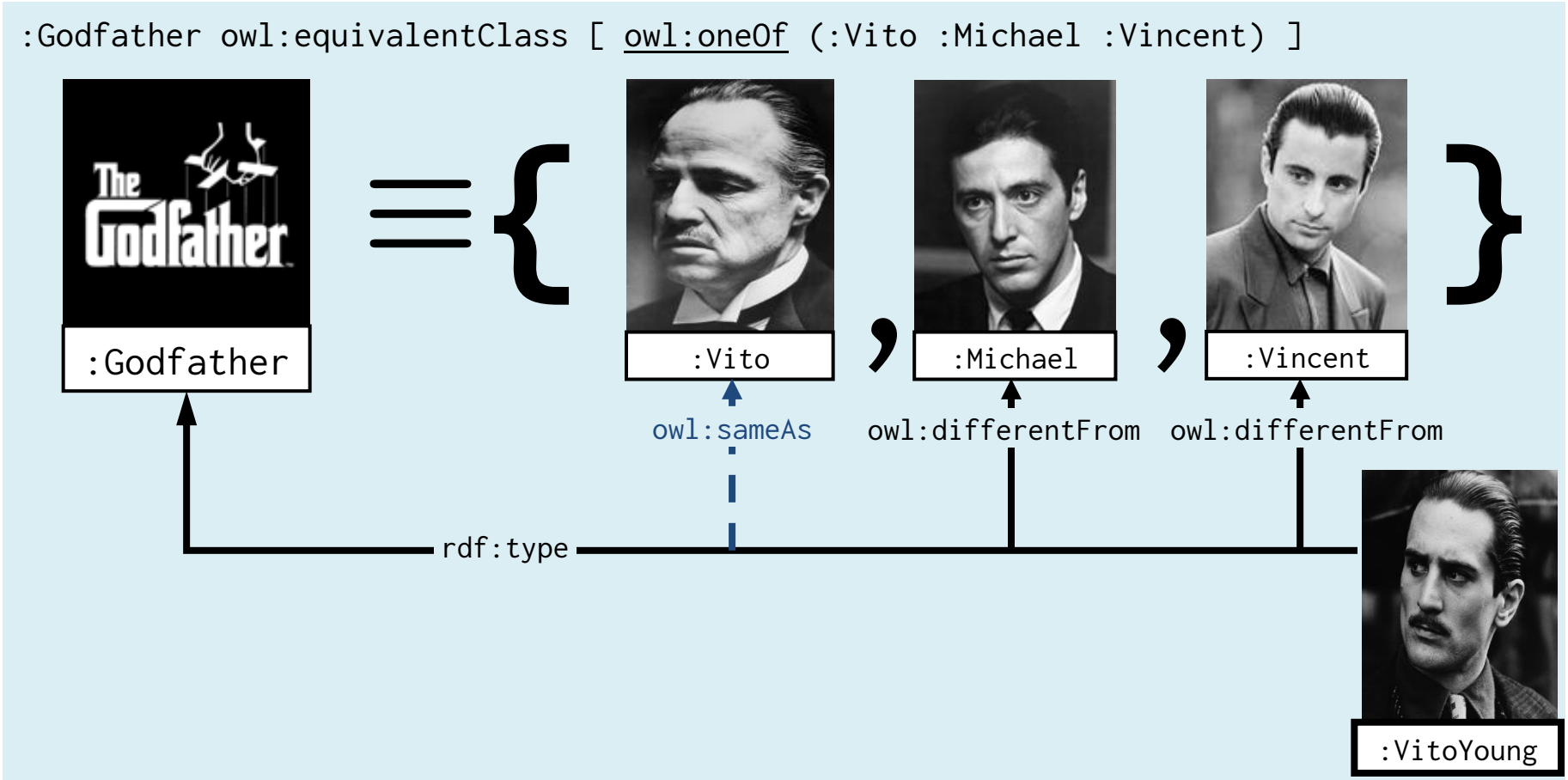


How do we define a rule for this case?

What if it's not the first element?

Have we considered all possible cases like this?

owl:oneOf ({})



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) ]
```

How can we begin to formally define what classes mean?

Use set theory:

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

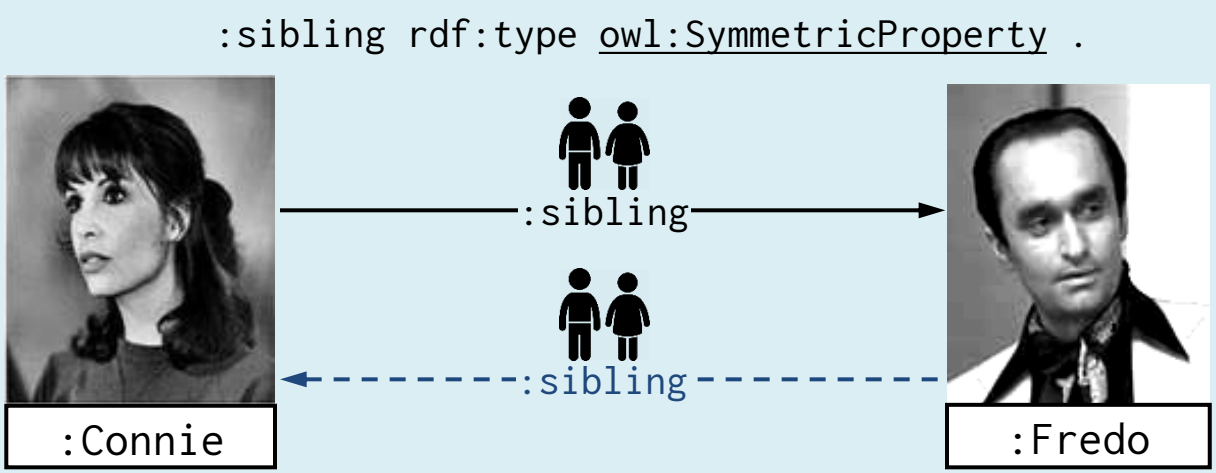
There is a set of classes (like  , etc.)

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

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

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

owl:SymmetricProperty



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

To Simplify matters

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/refer to (sub)sets of U
Properties are/refer to sets of pairs from U

Vocabulary / Data

Need to name/describe things in the universe

- Vocabulary (IRIs): V
- Triples (RDF): $V \times V \times V$
  - 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 $v \in V$ to an element of U

    (interpretation of Individuals)

I(:Mary) = 

I(:Fredo) = 

I(:Vito) = 

I(:Italy) = 

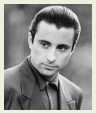
I(:Oscar) = 

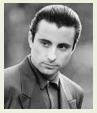
I(:Sonny) = 

...

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 $v \in V$ to an element of U
    (interpretation of **I**ndividuals)
  - C: Maps a vocabulary term $v \in V$ to a subset of U
    (interpretation of **C**lasses)

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 $v \in V$ to an element of U
    (interpretation of **I**ndividuals)
  - C: Maps a vocabulary term $v \in V$ to a subset of U
    (interpretation of **C**lasses)
  - P: Maps a vocabulary term $v \in V$ to a set of pairs from U
    (interpretation of **P**roperties)

P(:parent) = { ( , ), ( , ), … }

P(:sibling) = { ( , ), ( , ), … }

…

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 $v \in V$ to an element of U
    (interpretation of **I**ndividuals)
  - C: Maps a vocabulary term $v \in V$ to a subset of U
    (interpretation of **C**lasses)
  - P: Maps a vocabulary term $v \in V$ to a set of pairs from U
    (interpretation of **P**roperties)

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 $(s,p,o) \in G$ then $(I(s), I(o)) \in P(p)$

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) ∈ 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)

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 $(s,p,o) \in G$ 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 $P(s) \subseteq P(o)$

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 $\pi_2(P(s)) \subseteq C(o)$

(π_n : project n^{th} element)

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)

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) = ∅

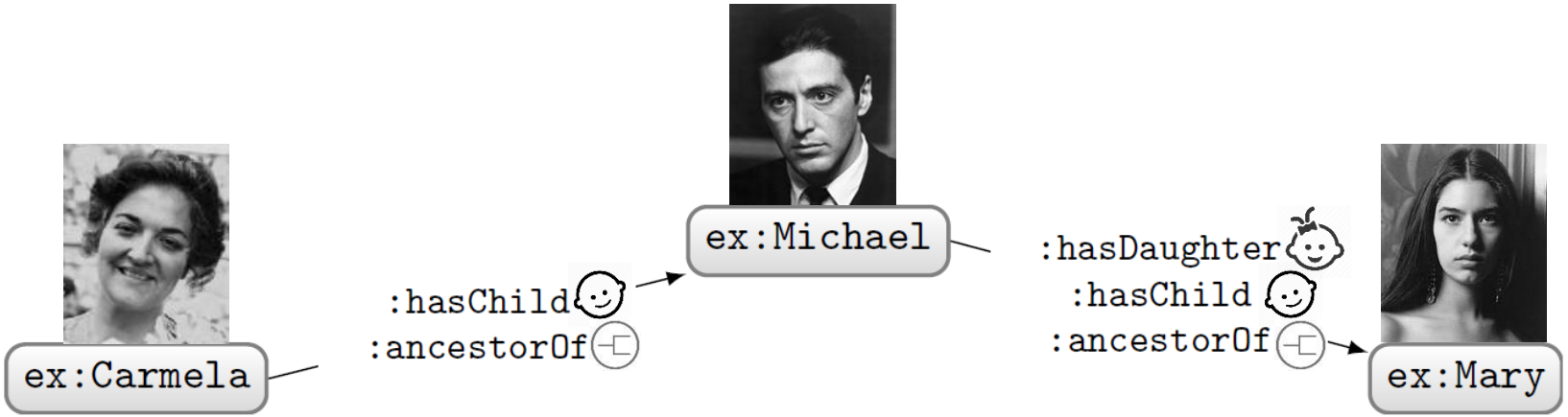
… and so on.

(Back to) Models

Models of graphs/ontologies

A model is any world that an ontology might describe

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



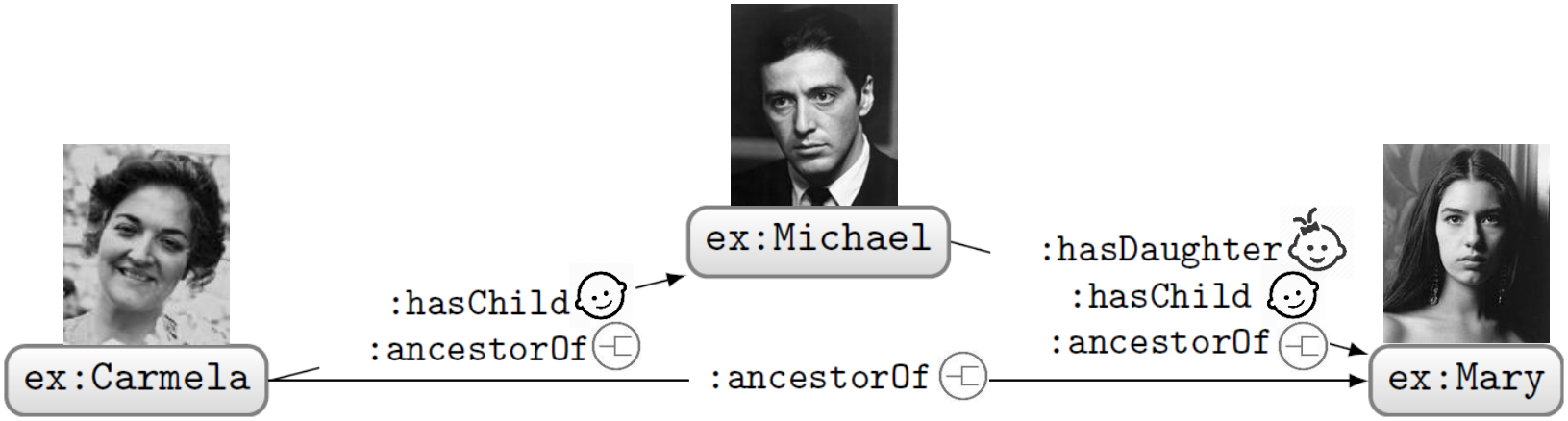
Is this a model of the ontology?

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

Models of ontologies

A model is any world that an ontology might describe

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

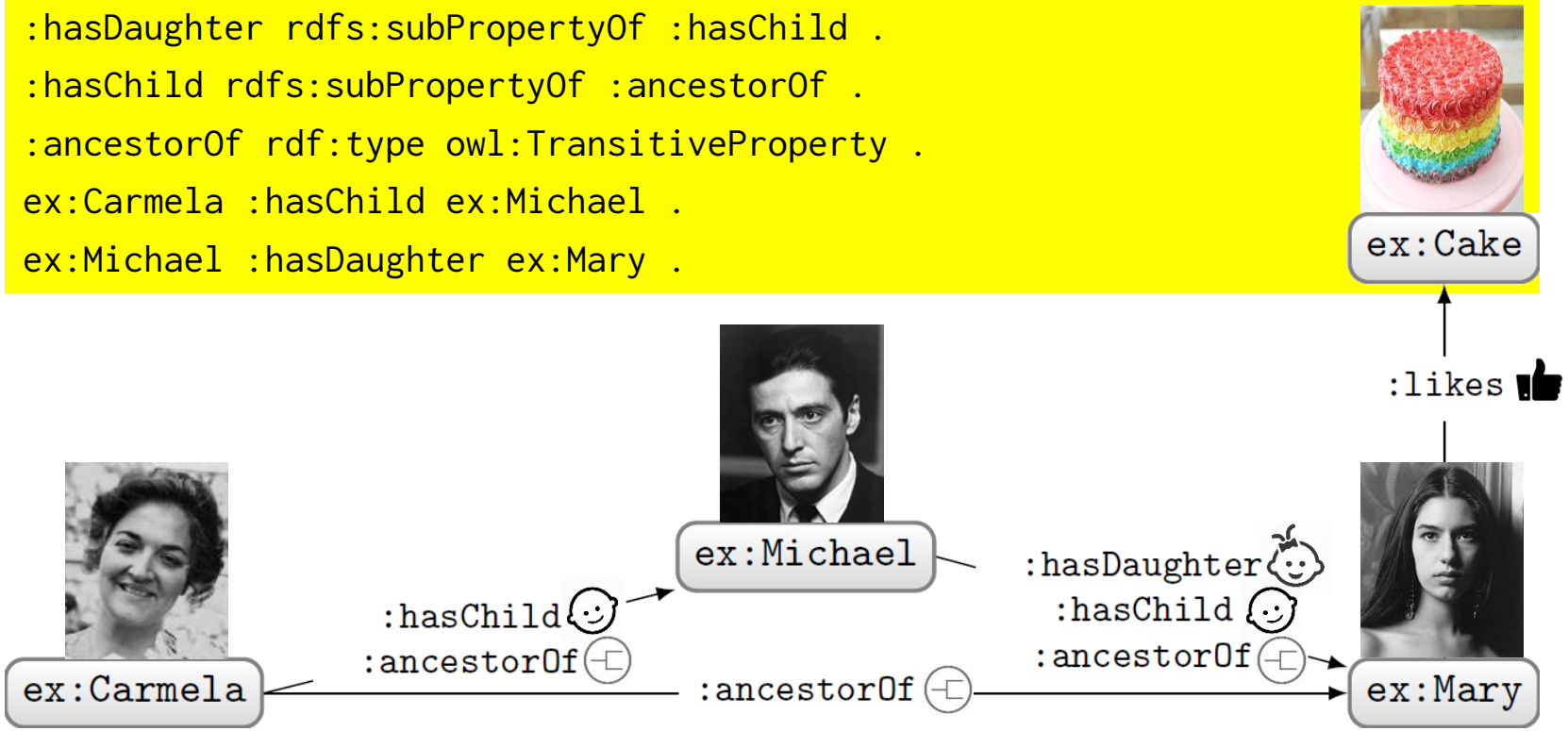


Now we have a model of the ontology (for the given semantics).

Models of ontologies

A model is any world that an ontology might describe

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



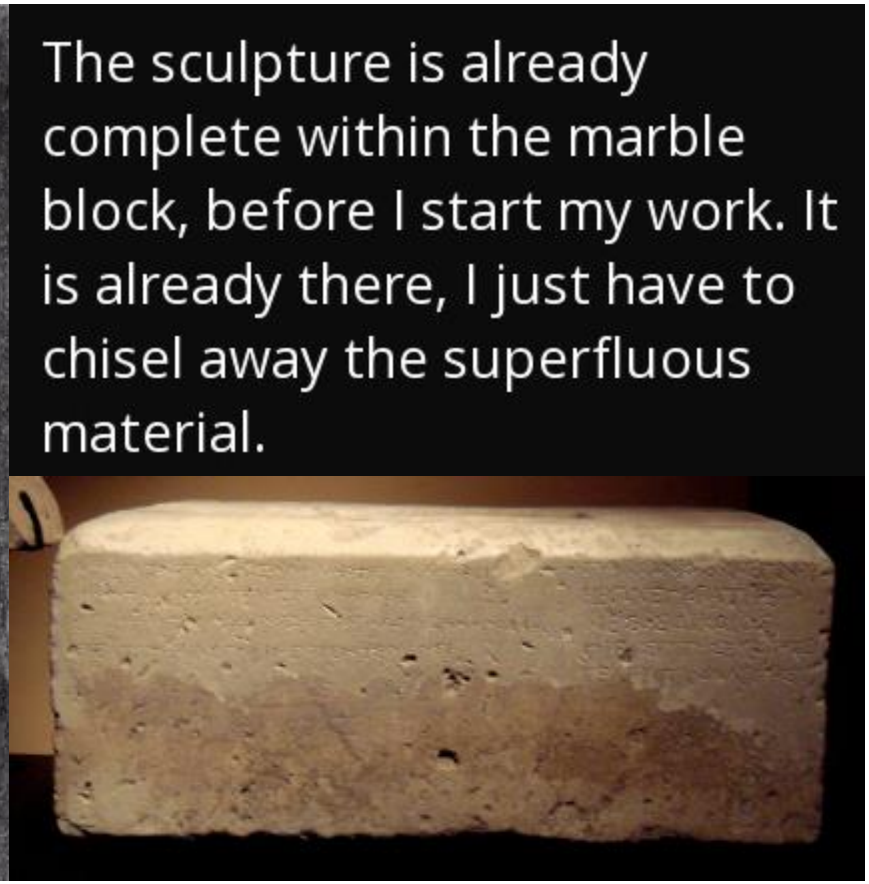
Is this a model of the ontology?

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

Entailment …

Axioms remove models

- At the start (almost) any world is possible!
- As we add axioms, fewer worlds are possible



Ontology O entails O' ($O \models O'$)

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

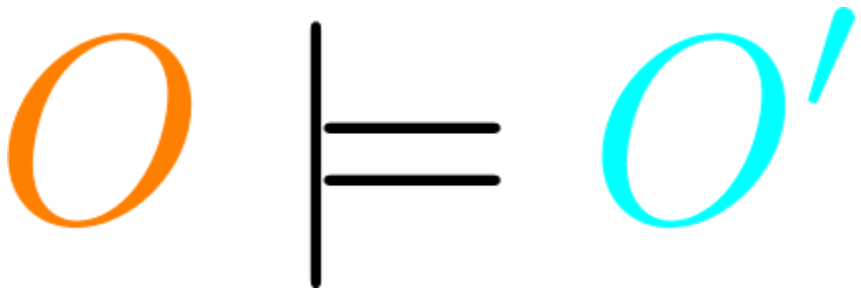
```
ex:Michael :hasParent ex:Carmela .
ex:Michael :hasChild ex:Mary .
ex:Carmela :ancestorOf ex:Mary .
```

Any model of O is a model of O'

(O' has a superset of the models of O)

(O' says nothing new over O)

Entailment symbol: ⊨



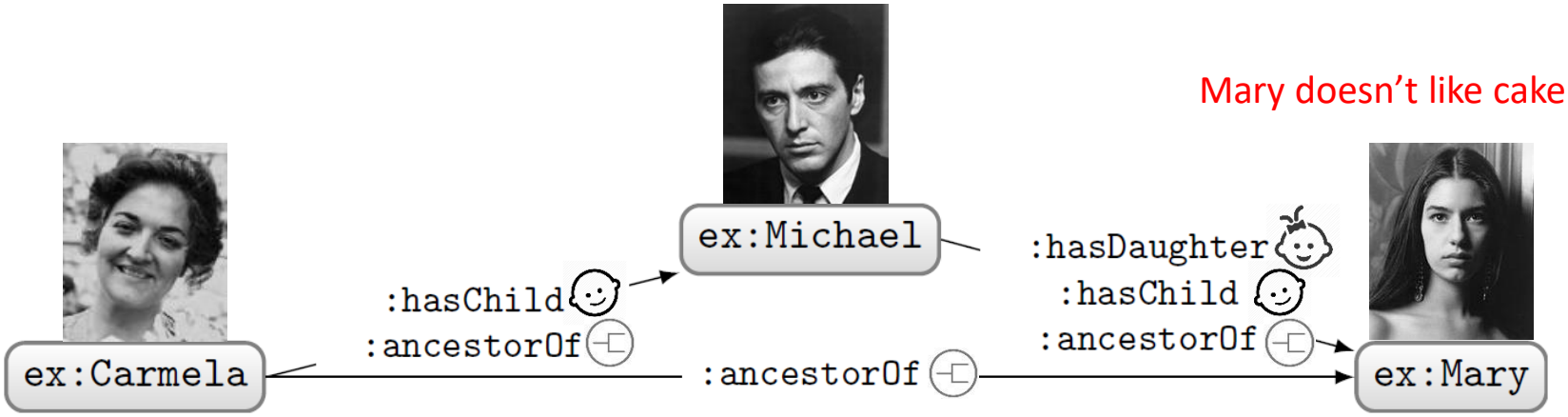
Ontology Entailment

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

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

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

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



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 .
ex:Carmela :hasChild ex:Michael .
ex:Michael :hasDaughter ex:Mary .
ex:Mary a :Person .
:Person owl:equivalentClass
  [ owl:qualifiedCardinality 2 ;
    owl:onProperty :hasParent ;
    owl:onClass :Person ] .
```

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

Any problems with this?

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 .
_: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 .
ex:Carmela :hasChild ex:Michael .
ex:Michael :hasDaughter ex:Mary .
ex:Mary a :Person .
:Person owl:equivalentClass
  [ owl:qualifiedCardinality 2 ;
    owl:onProperty :hasParent ;
    owl:onClass :Person ] .
```

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

```
:Person rdfs:subClassOf [ owl:maxQualifiedCardinality 2 ; owl:onProperty
   :hasParent ; owl:onClass :Person ] .
:Person rdfs:subClassOf [ owl:maxQualifiedCardinality 3 ; owl:onProperty
   :hasParent ; owl:onClass :Person ] . …
```

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 ] .
```

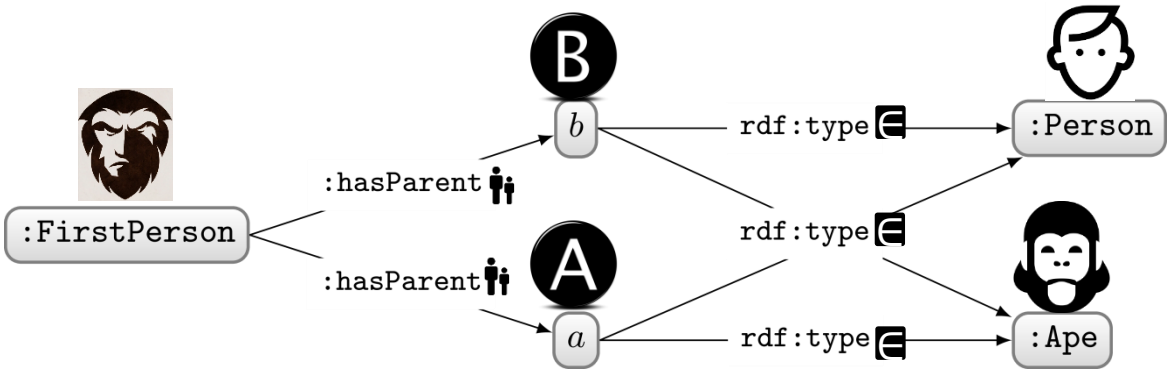
```
ex:Michael :hasParent ex:Carmela .
ex:Michael :hasChild ex:Mary .
ex:Carmela :ancestorOf ex:Mary .
```

```
:Person rdfs:subClassOf [ owl:maxQualifiedCardinality 2 ; owl:onProperty
   :hasParent ; owl:onClass :Person ] .
:Person rdfs:subClassOf [ owl:maxQualifiedCardinality 3 ; owl:onProperty
   :hasParent ; owl:onClass :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 ] .
```

Does *O* have any model?



Not shown: parents of *a* and *b*, and their parents, recursively

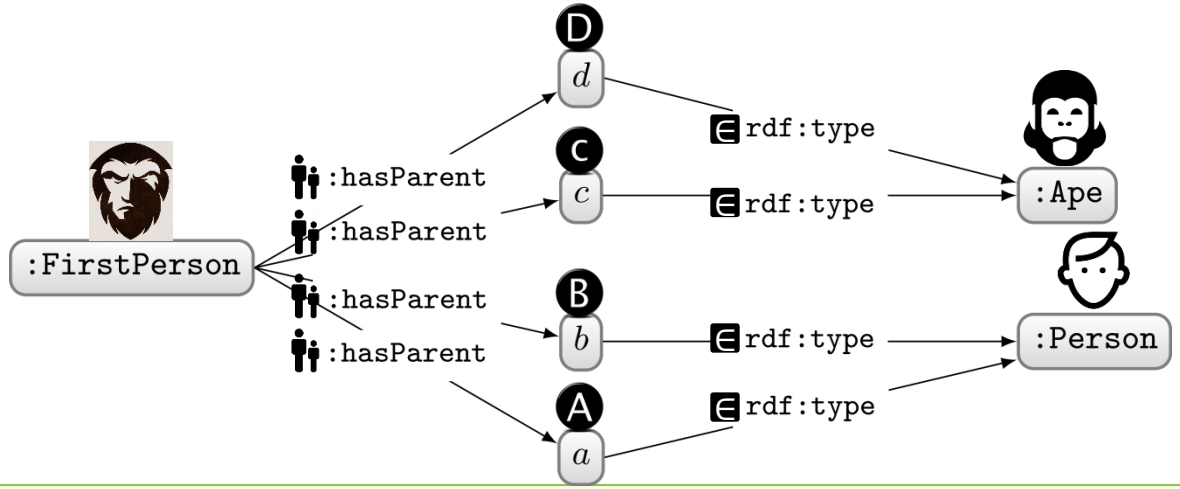
YES! Ontology *O* is **Satisfiable**!

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?



Not shown: parents of *a* and *b*, and their parents, recursively

YES! Ontology *O* is still **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 .
```

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 …

Ontology Satisfiability:

Does O have a "model"?

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

An unsatisfiable ontology cannot model any world!

It is inconsistent!

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

OR

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

OR …

Entailment checking: Does *O* entail *O'*?

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

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

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

Reasoning …

How can we perform reasoning?

* In case that O' claims more than one thing, here $\neg O'$ means that *some* claim is negated.

- Does Ontology O entail O'?

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

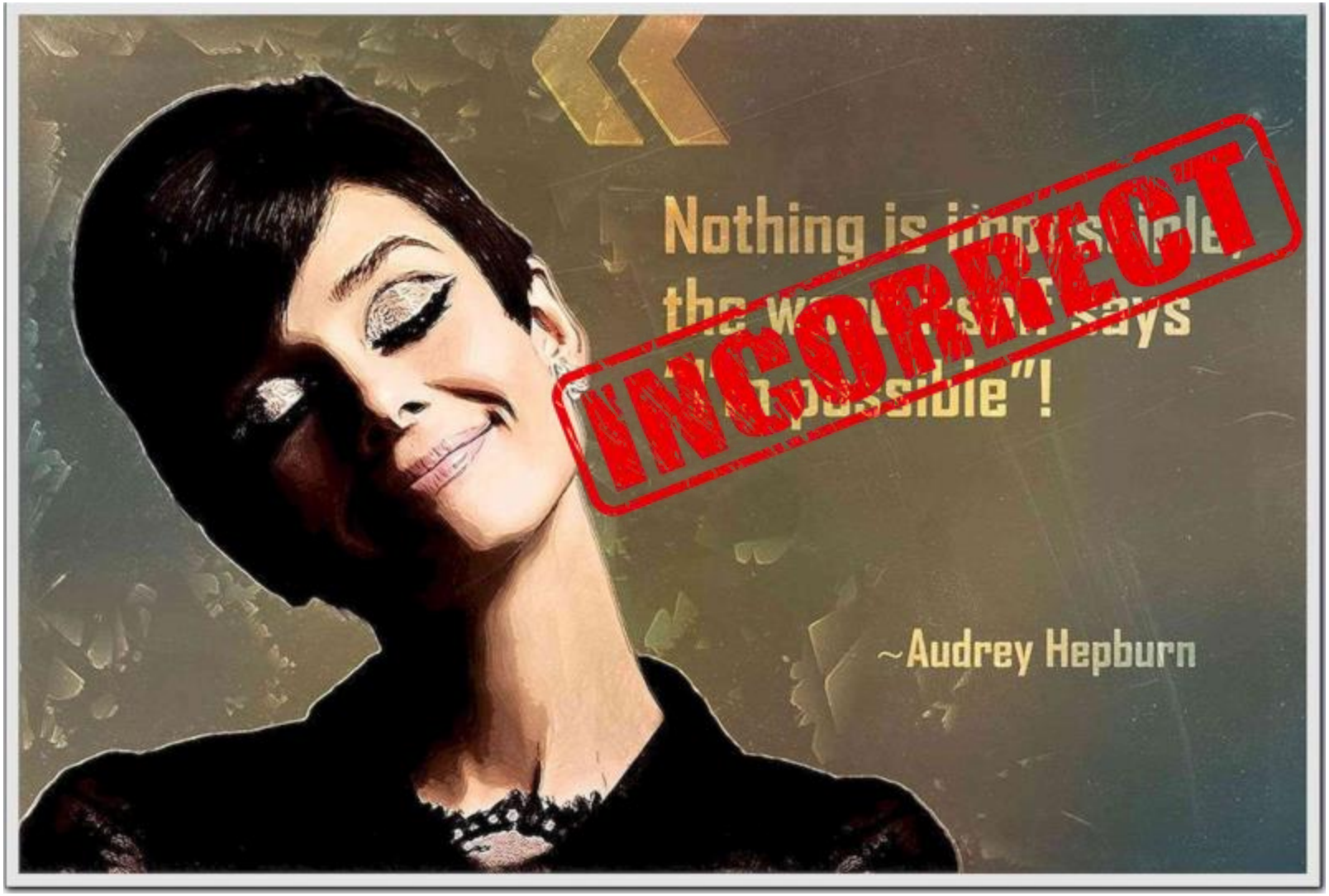
```
ex:Michael :hasChild ex:Mary .
```

- 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!

So how do we test unsatisfiability then?



Undecidability ...

A simple Java program ...

```
public class Collatz {
    public static void collatz(int n) {
        StdOut.print(n + " ");
        if (n == 1) return;
        else if (n % 2 == 0) collatz(n / 2);
        else collatz(3*n + 1);
    }

    public static void main(String[] args) {
        int N = Integer.parseInt(args[0]);
        collatz(N);
        StdOut.println();
    }

}
```

In: 6

```
3
10
5
16
8
4
2
1 (end)
```

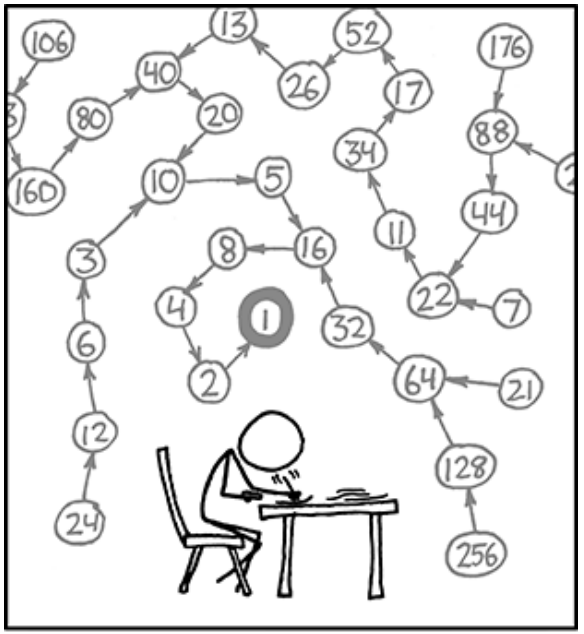
Does this Java program terminate on all possible (positive) inputs?

A simple Java program ...

```
public class Collatz {
    public static void collatz(int n) {
        StdOut.print(n + " ");
        if (n == 1) return;
        else if (n % 2 == 0) collatz(n / 2);
        else collatz(3*n + 1);
    }

    public static void main(String[] args) {
        int N = Integer.parseInt(args[0]);
        collatz(N);
        StdOut.println();
    }

}
```



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.

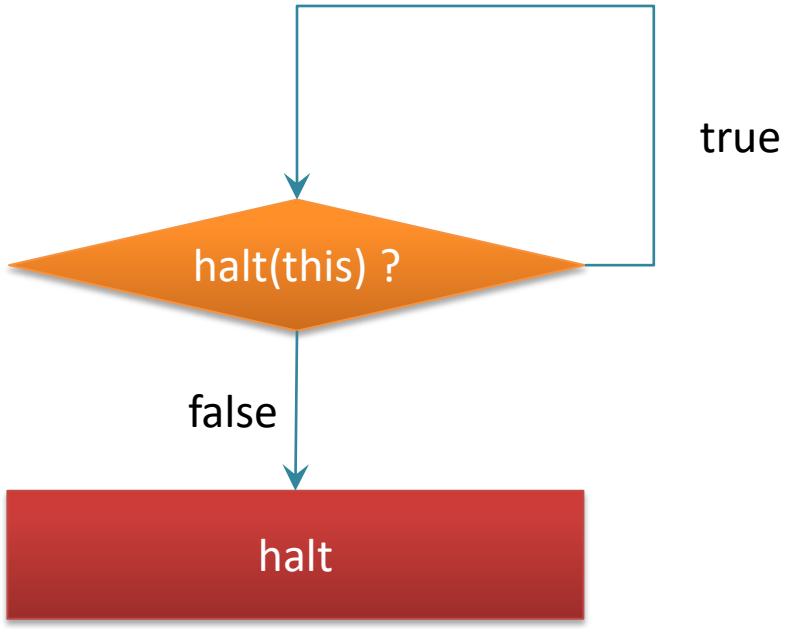
- Collatz conjecture: an unsolved problem in mathematics
- If we knew that this program terminates or does not terminate on all natural numbers (not just `int`s), this problem would be solved!

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 all program–input pairs!
  - It may halt for some program–input pairs

A quick *sketch* of Halting Problem proof



Problem: Collatz Halting Problem

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

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

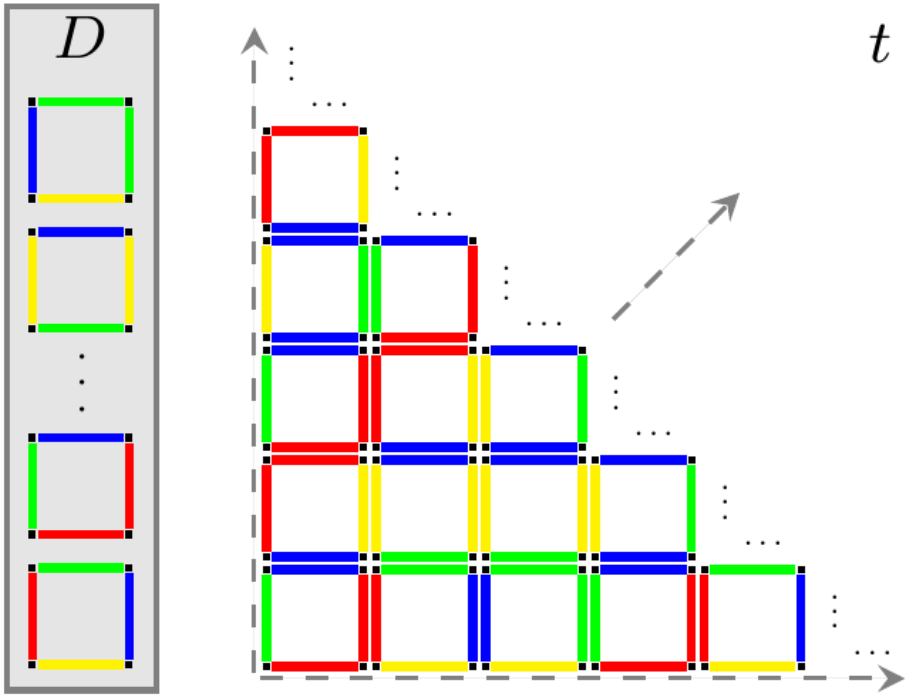
(P1) `return true;`

(P2) `return false;`

• either **(P1)** or **(P2)** must be correct
∴ there must exist a correct program that halts (even if we don't know it)
∴ problem is **DECIDABLE**!

Halting Problem **UNDECIDABLE** in the general case
(for all programs and inputs)

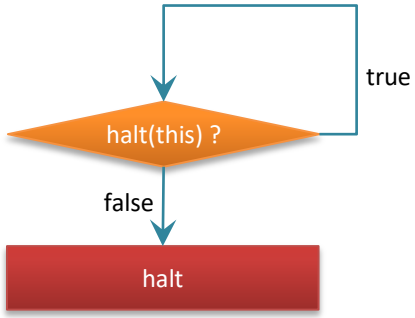
Domino Tiling Problem



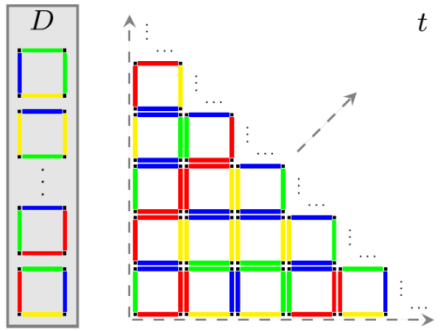
Is this problem
DECIDABLE or
UNDECIDABLE?

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

Can reduce from Halting to Tiling



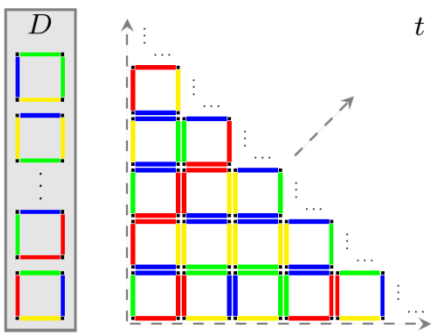
Turing reduction



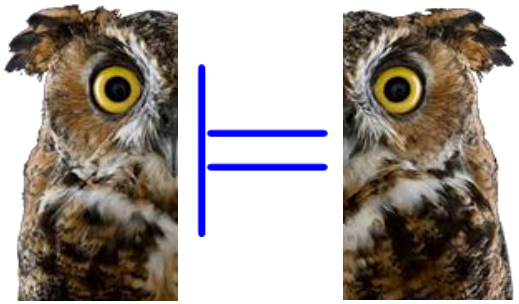
It has been shown that there exists a program that can reduce any Halting problem instance into a Domino Tiling problem instance

Thus the Tiling Problem is **UNDECIDABLE**

Reduce from Tiling to OWL entailment?



Turing 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?

Questions?