OWL

Esercizi

Manuel Fiorelli

fiorelli@info.uniroma2.it

Esercizio #1

```
:Document a owl:Class ;
    owl:equivalentClass [ a owl:Restriction ;
        owl:onProperty :author ;
        owl:allValuesFrom :Person
    ] .
:Person a owl:Class .
:author a owl:ObjectProperty .

:doc a :Document :
    :author :pippo .
:pippo a owl:Thing.
```

→ **Document ≡ ∀author.Person**

Possiamo inferire `:pippo a :Person`, perché `pippo` è `author` di un `Document`, ed un `Document` ha solo `author` di tipo `Person`.

Manuel Fiorelli fiorelli@info.uniroma2.it
http://art.uniroma2.it/fiorelli

Esercizio #2

```
:Document2 a owl:Class ;
    owl:equivalentClass [ a owl:Restriction;
        owl:onProperty :author ;
        owl:someValuesFrom :Person
    ] .
:Person a owl:Class .
:author a owl:ObjectProperty .

:doc2 a :Document2 :
    :author :pippo .
:pippo a owl:Thing.
```

→ **Document ≡ ∃author.Person**

Sappiamo che `doc2` ha un `author` di tipo `Person`, ma non sappiamo se esso coincida con `pippo`: per l'ipotesi di mondo aperto `doc2` potrebbe avere altri `author`.

Non si può, quindi, inferire che `pippo` è di tipo `Person`.

Manuel Fiorelli fiorelli@info.uniroma2.it
http://art.uniroma2.it/fiorelli

Esercizio #3

```
:Document a owl:Class ;
      owl:equivalentClass [ a owl:Restriction;
            owl:onProperty :author ;
            owl:allValuesFrom :Person
      ] .
:Document2 a owl:Class ;
      owl:equivalentClass [ a owl:Restriction;
            owl:onProperty :author ;
            owl:someValuesFrom :Person
      ] .
:Document3 a owl:Class ;
      owl:equivalentClass [ a owl:Restriction;
            owl:onProperty :author ;
            owl:hasValue :pippo
      ] .
:Person a owl:Class .
:author a owl:ObjectProperty .
:doc a :Document :
      :author :pippo .
:pippo a owl:Thing.
```

Document ≡ ∀author.Person

Document2 ≡ ∃author.Person

Document3 ≡ author ∋ pippo

Esercizio #3 (cont)

Document ≡ ∀author.Person

Document2 ≡ ∃author.Person

Document3 ≡ author ∋ pippo

:doc a :Document .

:doc :author :pippo .

:pippo a owl:Thing .

Possiamo inferire:

`:pippo a :Person` (gli `author` si un `Document` sono `Person`)

`:doc a :Document3` (perché `pippo` è un `author` di `doc`)

`Document3 rdfs:subClassOf :Document2` . (perché ciò che ha `pippo` come `author` ha almeno un `author` di tipo `Person`).

`doc a Document2` . (segue dai due risultati precedenti)

Esercizio #4

```
:Document a owl:Class ;
    owl:equivalentClass [ a owl:Restriction;
        owl:onProperty :author ;
        owl:allValuesFrom :Person
    ] .
:Document2 a owl:Class ;
    owl:equivalentClass [ a owl:Restriction;
        owl:onProperty :author ;
        owl:someValuesFrom :Person
    ] .
:Document3 a owl:Class ;
    owl:equivalentClass [ a owl:Restriction;
        owl:onProperty :author ;
        owl:hasValue :pippo
    ] .
:Person a owl:Class .
:author a owl:ObjectProperty .
:doc a owl:Thing ;
    :author :pippo .
:pippo a :Person.
```

Document ≡ ∀author.Person

Document2 ≡ ∃author.Person

Document3 ≡ author ∋ pippo

Esercizio #4 (cont)

Document ≡ ∀author.Person

Document2 ≡ ∃author.Person

Document3 ≡ author ∋ pippo

:doc a ow:Thing .

:doc :author :pippo .

:pippo a :Person .

Possiamo inferire:

`:doc a :Document2` (perché ha un `author` di tipo `Person`)

`:doc a :Document3` (perché ha `pippo` come `author`)

`Document3 rdfs:subClassOf :Document2` . (perché ciò che ha `pippo` come `author` ha almeno un `author` di tipo `Person`).

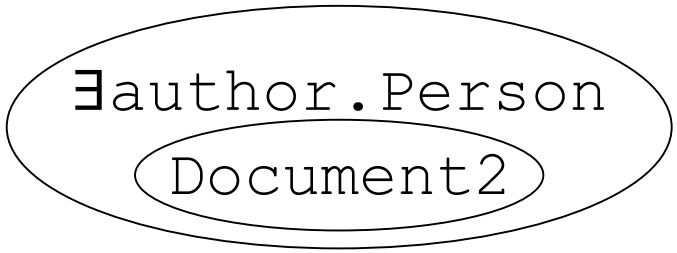
Esercizio #5

```
:Document2 a owl:Class ;
    rdfs:subClassOf [ a owl:Restriction;
        owl:onProperty :author ;
        owl:someValuesFrom :Person
    ] .
:Person a owl:Class .
:author a owl:ObjectProperty .

:doc a owl:Thing :
    :author :pippo .
:pippo a :Person.
```

→ **Document2 ⊑ ∃author.Person**

Non si può inferire che `doc` è un `Document2`, perché avere un `author` di tipo `Person` è condizione necessaria ma non sufficiente di appartenenza alla classe `Document2`.



Esercizio #6

```
:Document a owl:Class ;
    owl:equivalentClass [ a owl:Restriction;
        owl:onProperty :author ;
        owl:cardinality "2"^^xsd:nonNegativeInteger
    ] .

:doc a owl:Thing ;
    :author :a1 ;
    :author :a2 .
```

Document ≡ (= 2 author)

Non si può inferire che `doc` è di tipo `Document`, perché nulla esclude un terzo `author` (open world assumption).

Esercizio #7

```
:Document a owl:Class ;
    owl:equivalentClass [ a owl:Restriction;
        owl:onProperty :author ;
        owl:minCardinality "2"^^xsd:nonNegativeInteger
    ] .
```

Document ≡ (≥ 2 author)

```
:doc a owl:Thing ;
    :author :a1 ;
    :author :a2 .
```

Non si può inferire che `doc` è di tipo `Document`, perché `a1` e `a2` potrebbero denotare la stessa risorsa (per la *no unique name assumption*)

Esercizio #8

```
:Document a owl:Class ;
    owl:equivalentClass [ a owl:Restriction;
        owl:onProperty :author ;
        owl:minCardinality "2"^^xsd:nonNegativeInteger
    ] .

:doc a owl:Thing ;
    :author :a1 ;
    :author :a2 .

:a1 owl:differentFrom :a2 .
```

→ **Document ≡ (≥ 2 author)**

Possiamo inferire che

```
:doc a :Document
```

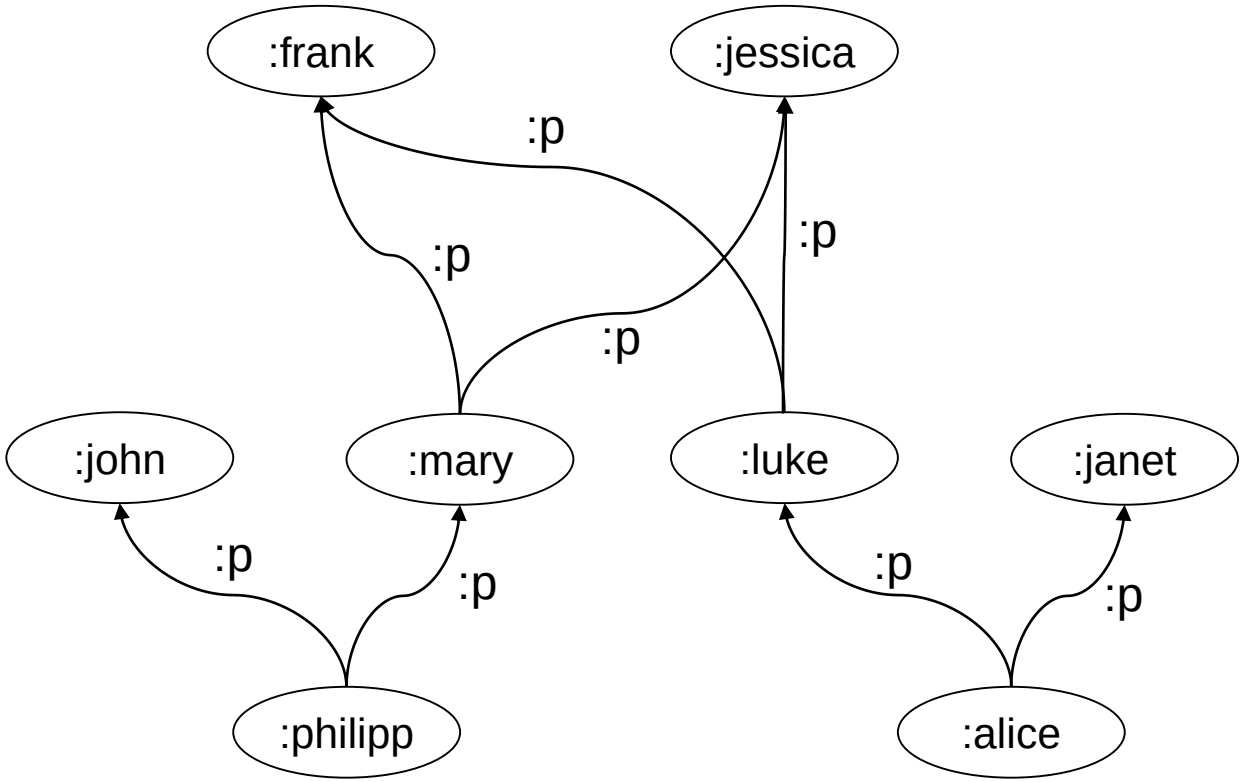
Esercizio #9 - TBOX

```
:relative a owl:ObjectProperty , owl:TransitiveProperty, owl:SymmetricProperty ;
    rdfs:domain :Person ;
    rdfs:range :Person .
:parent a owl:ObjectProperty ;
    rdfs:subPropertyOf :relative .
:Person a owl:Class ;
    rdfs:subClassOf [ a owl:Restriction ;
        owl:onProperty :relative ;
        owl:hasSelf true
    ] .
```

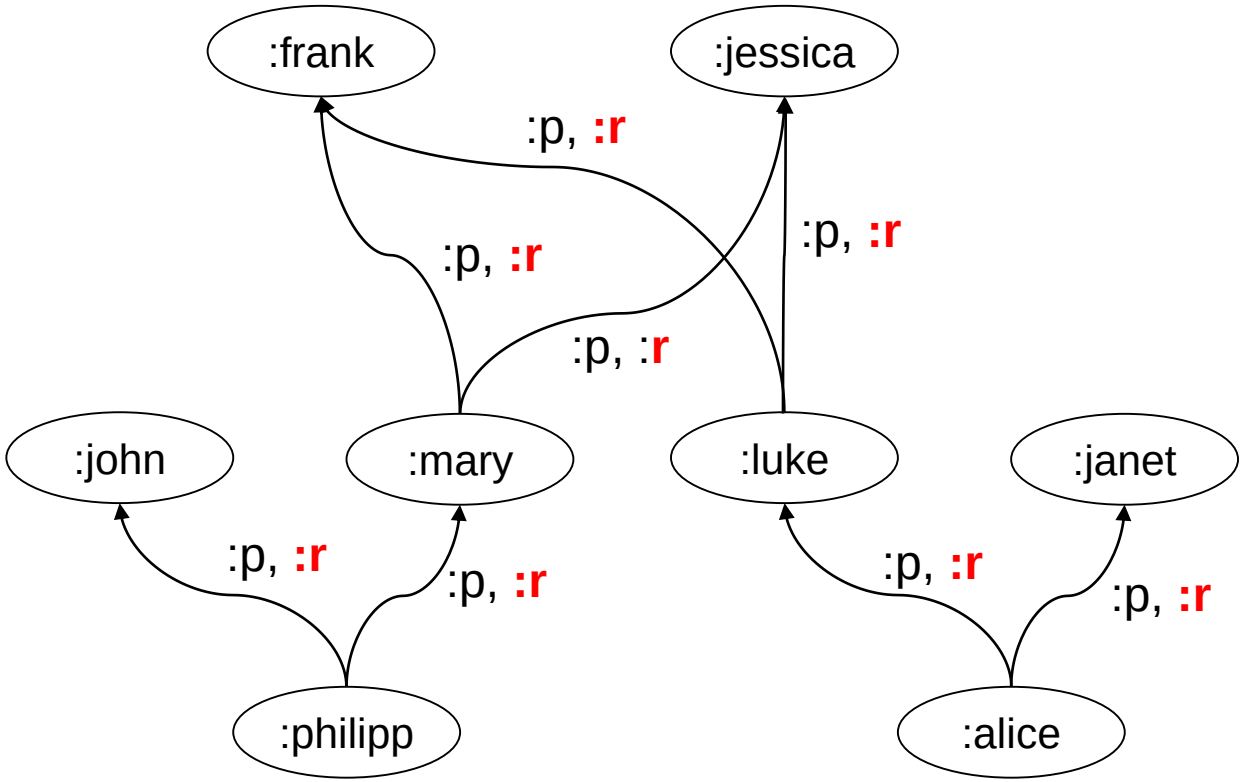
Esercizio #9 - ABOX

```
:philipp a owl:Thing ;
      :parent :john , :mary .
:alice a owl:Thing ;
      :parent :luke, :janet .
:mary a owl:Thing ;
      :parent :frank, :jessica .
:luke a owl:Thing ;
      :parent :frank, :jessica .
```

Esercizio #9 (cont)



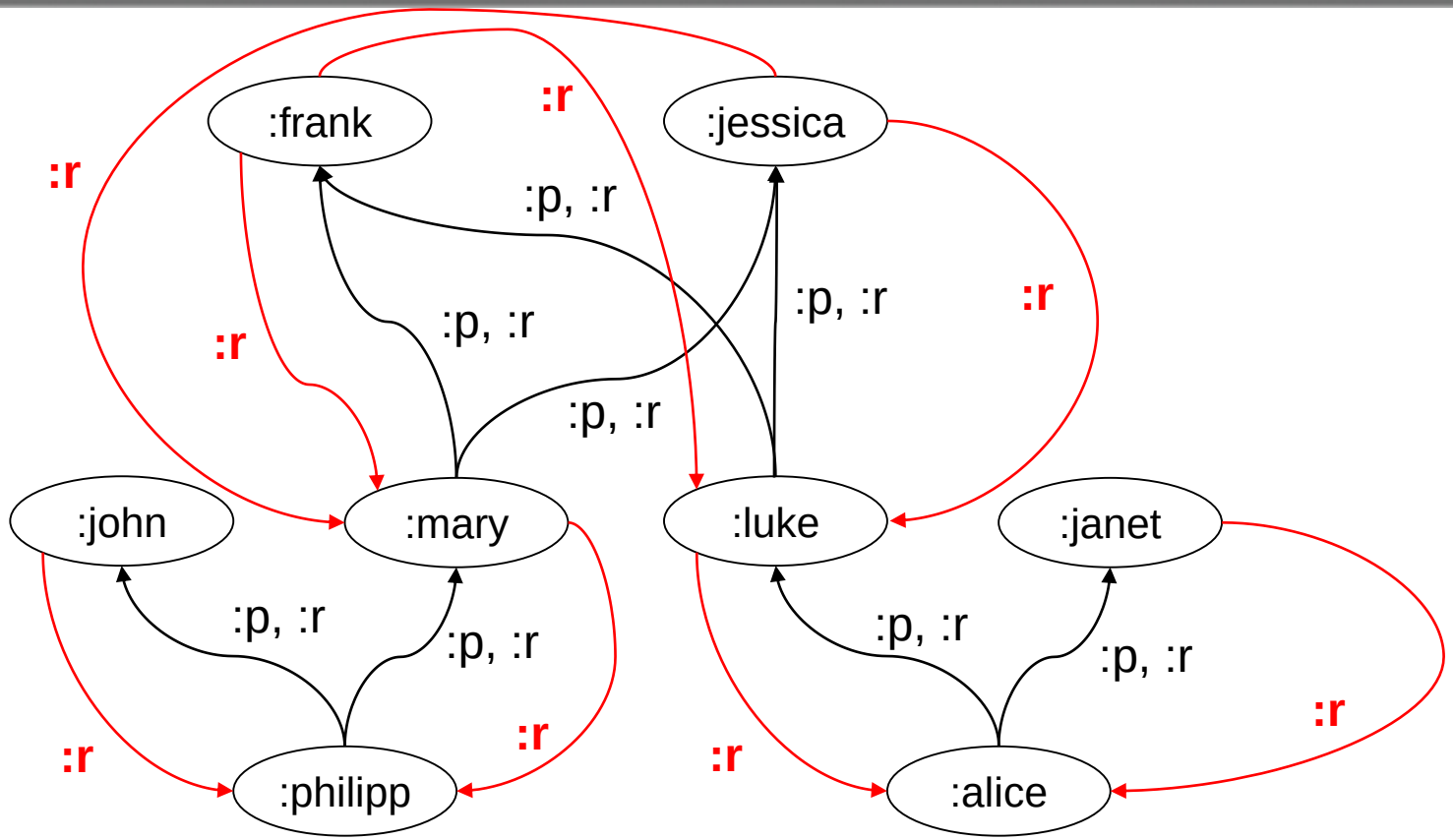
Esercizio #9 (cont 2)



Deduciamo che **x** ha `relative` **y,** se **x** ha `parent` **y**

Manuel Fiorelli fiorelli@info.uniroma2.it
http://art.uniroma2.it/fiorelli

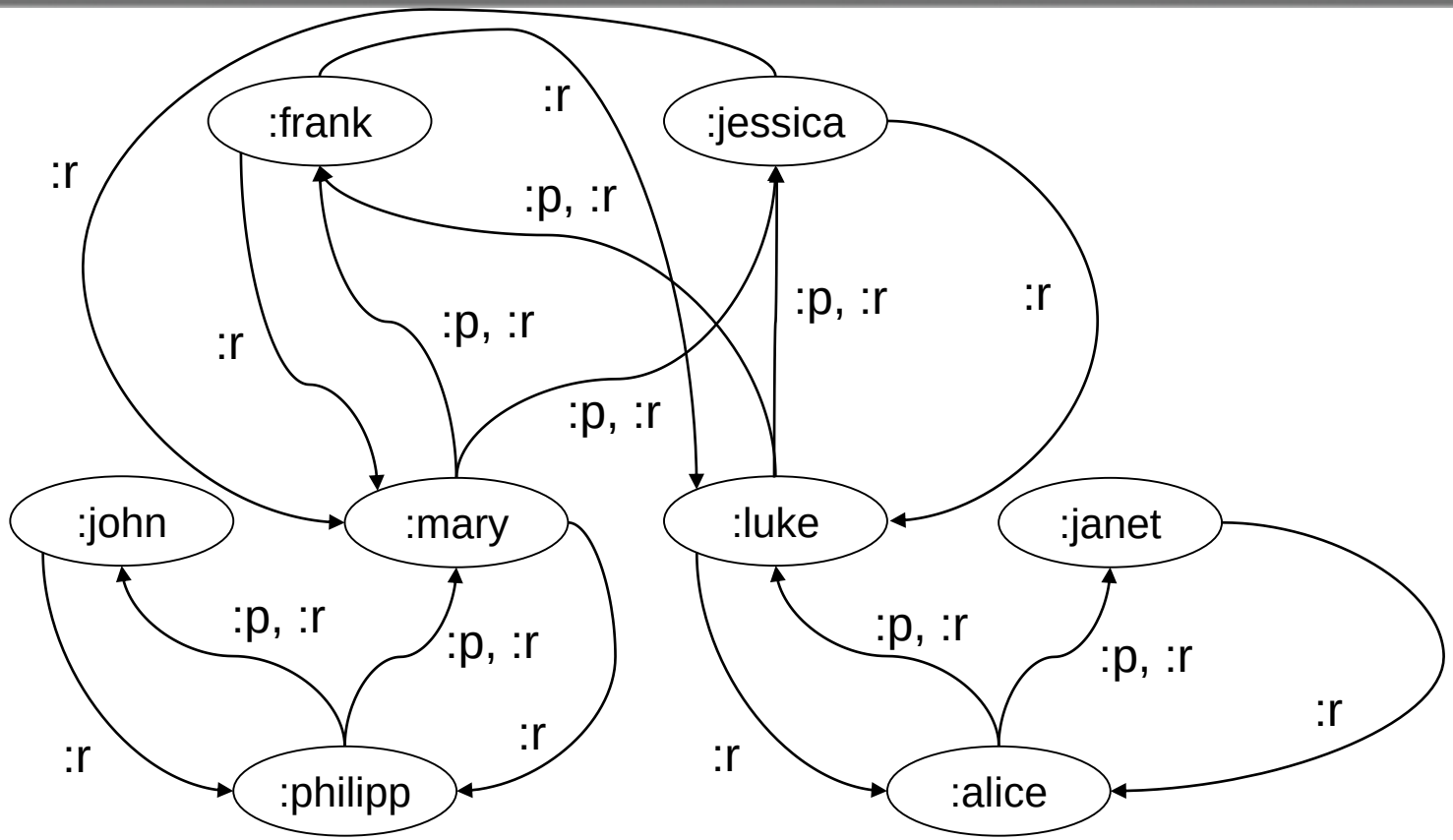
Esercizio #9 (cont 3)



Deduciamo che **x** ha `relative` **y,** se **y** ha `relative` **x**

Manuel Fiorelli fiorelli@info.uniroma2.it
http://art.uniroma2.it/fiorelli

Esercizio #9 (cont 4)



Siccome `relative` è transitiva, se da **x** possiamo arrivare ad **y** con un cammino di soli archi etichettati dalla proprietà `relative`, possiamo dedurre che **x** ha `relative` **y**. Benché non lo disegniamo, si può dedurre che ogni nodo ha relative ciascun altro nodo, incluso se stesso

Esercizio #9 (cont 5)

Abbiamo inferito che ogni nodo ha come `relative` ogni altro nodo, incluso se stesso. Siccome ogni nodo compare come soggetto o oggetto di una tripla col predicato `relative`, possiamo inferire che si tratta di una `:Person`.

Esercizio #10

Con riferimento alla base di conoscenza (TBOX + ABOX) dell'esercizio 9.

:grandparent a owl:ObjectProperty ;
owl:propertyChainAxiom (:parent :parent) .

Possiamo inferire

```
:pippo :grandparent :frank,:jessica .
:alice :grandparent :frank,:jessica .
```