Enabling Completeness-aware Querying in SPARQL

Luis Galárraga, Katja Hose, Simon Razniewski

May 14th, 2017

WebDB, Chicago

Outline

- Completeness in RDF knowledge bases
- Completeness oracles
- Our vision
  - Representations for completeness oracles
  - Reasoning with completeness oracles
  - Enabling completeness in SPARQL
- Summary & conclusions

Outline

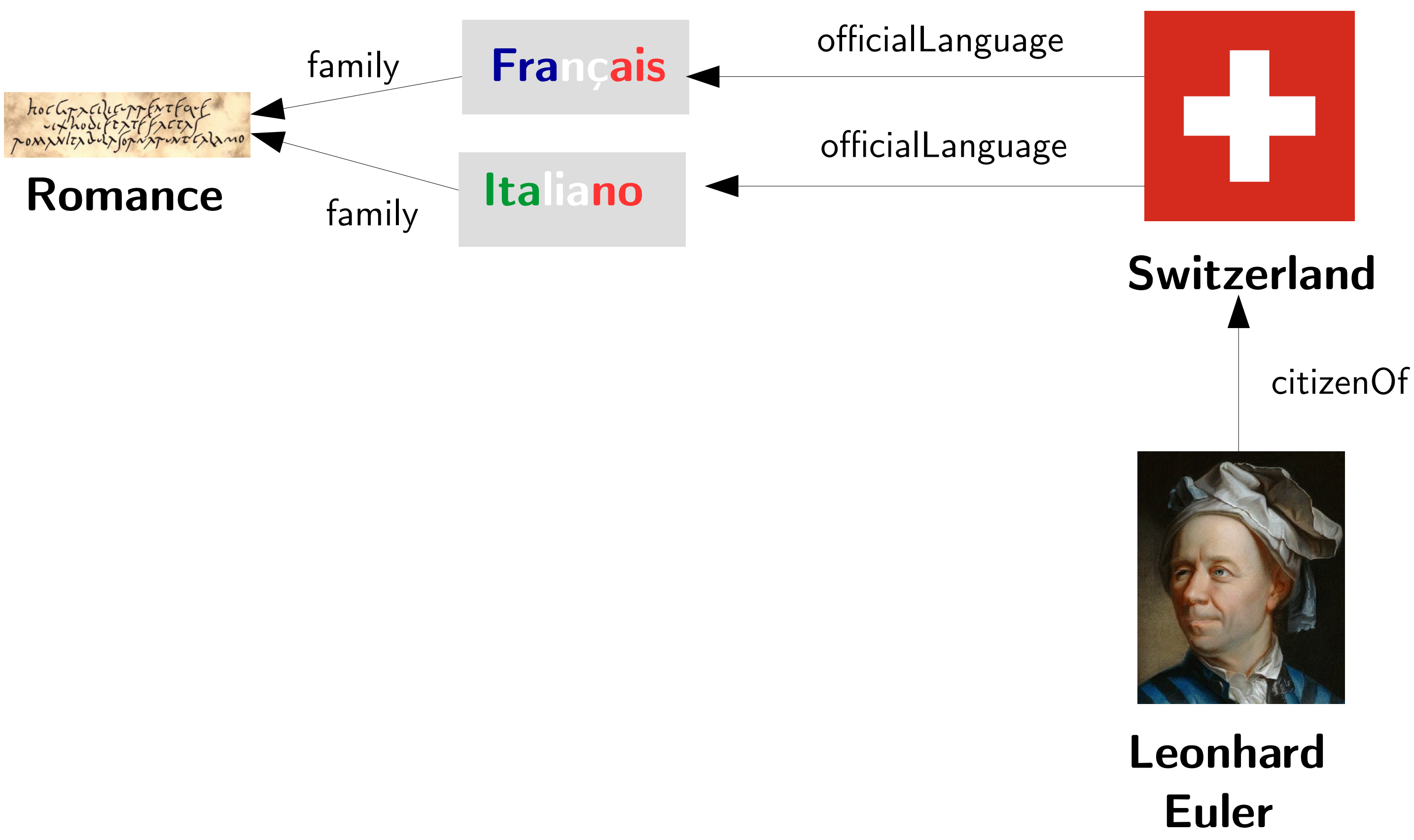
- Completeness in RDF knowledge bases
- Completeness oracles
- Our vision
  - Representations for completeness oracles
  - Reasoning with completeness oracles
  - Enabling completeness in SPARQL
- Summary & conclusions

Outline

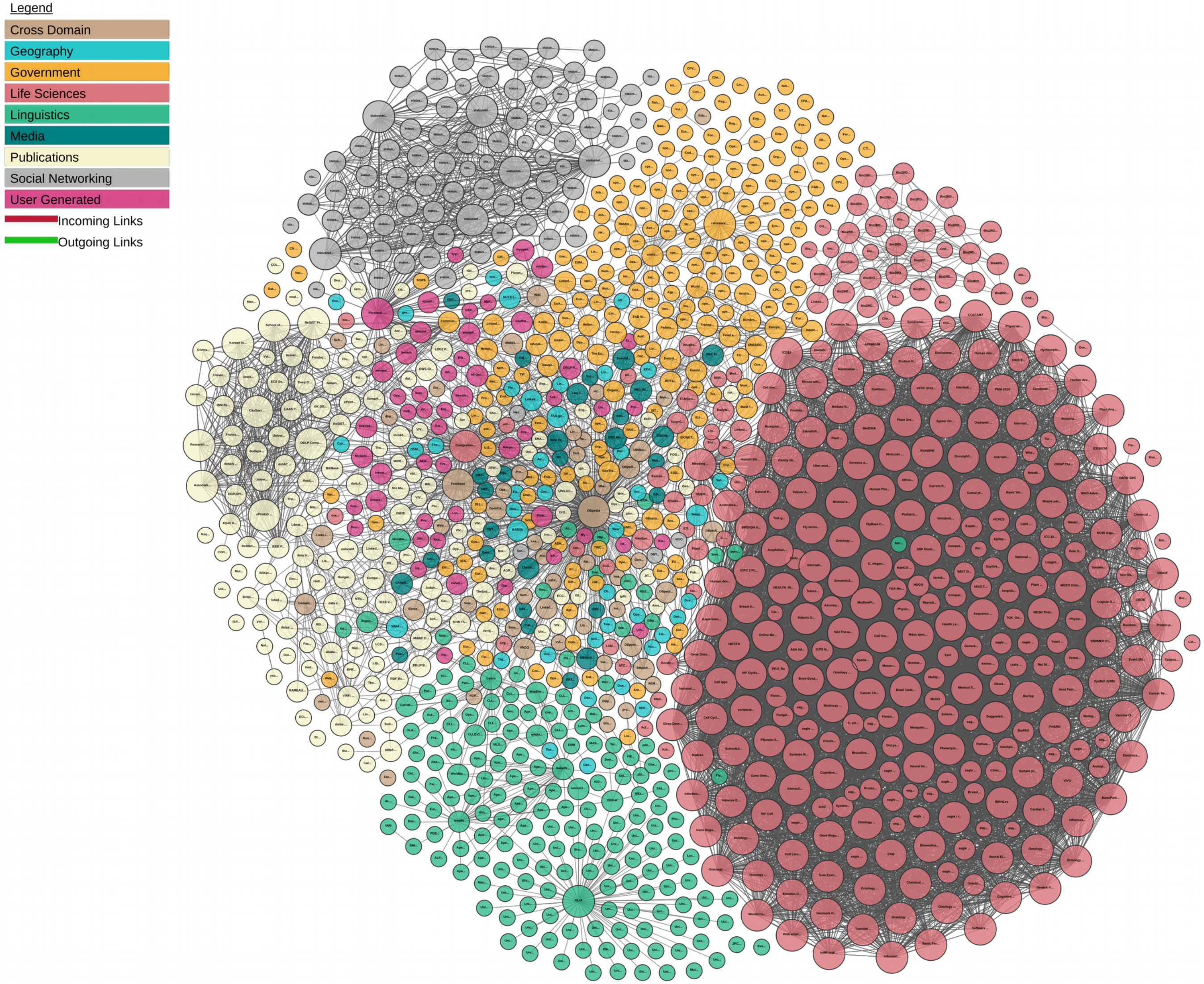
- Completeness in RDF knowledge bases
- Completeness oracles
- Our vision
  - Representations for completeness oracles
  - Reasoning with completeness oracles
  - Enabling completeness in SPARQL
- Summary & conclusions

RDF Knowledge Bases (KBs)

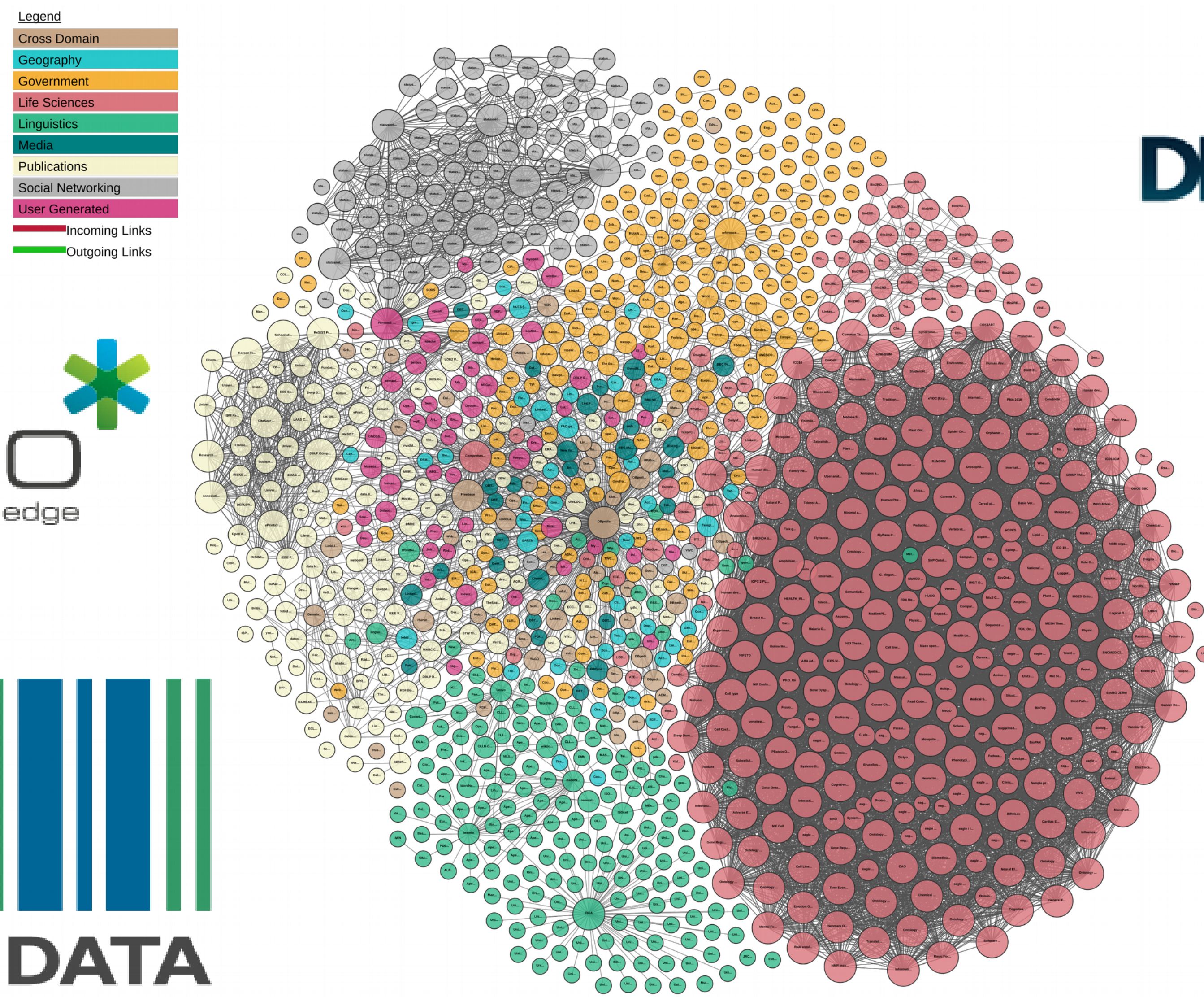
Collection of structured knowledge



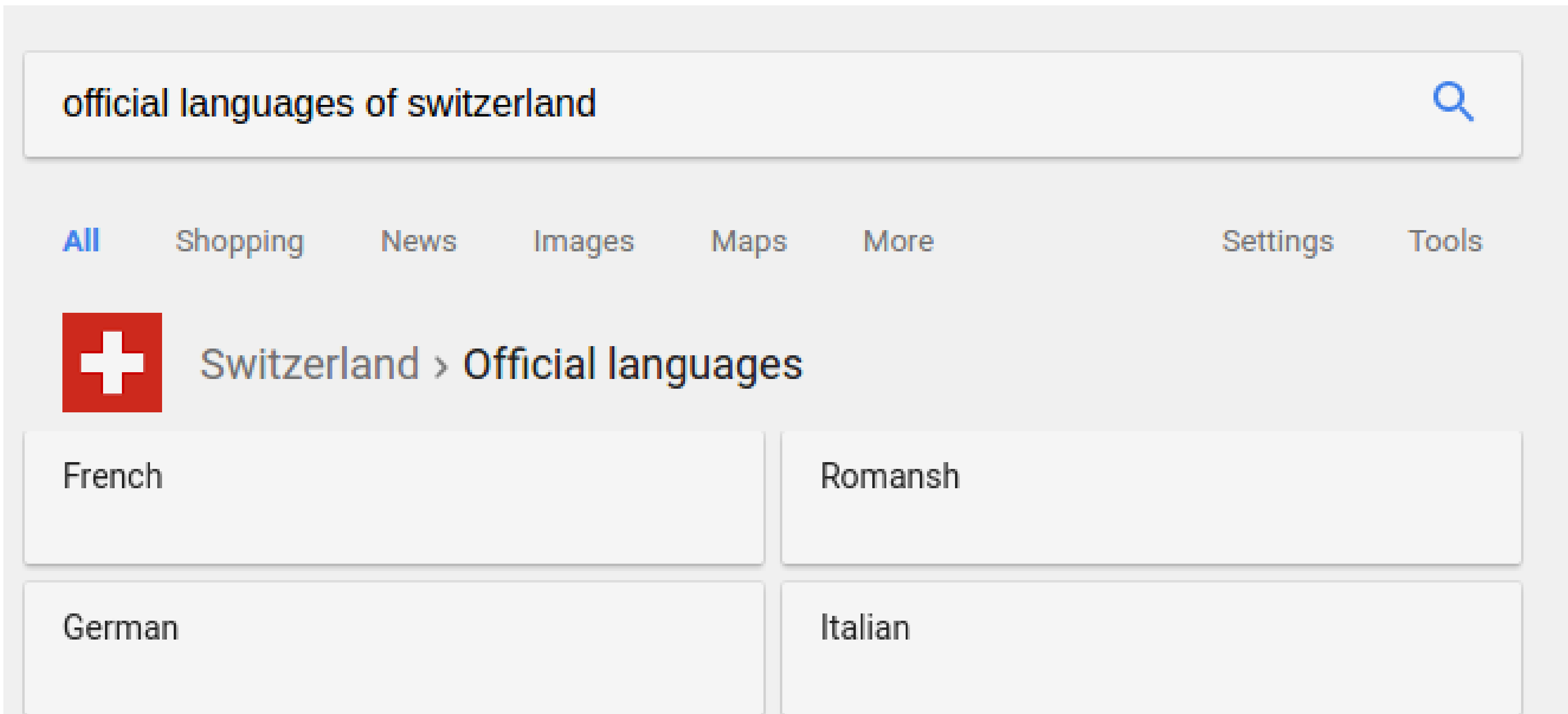
Plenty of KBs out there!



Plenty of KBs out there!



KBs in action



Outline

- Completeness in RDF knowledge bases
- Completeness oracles
- Our vision
  - Representations for completeness oracles
  - Reasoning with completeness oracles
  - Enabling completeness in SPARQL
- Summary & conclusions

Completeness in RDF KBs

- KBs are highly incomplete
  - 1% of people have a citizenship in YAGO

Completeness in RDF KBs

- KBs are highly incomplete
  - 1% of people have a citizenship in YAGO
- We do not know where the incompleteness lies

Completeness in RDF KBs

- KBs are highly incomplete
  - 1% of people have a citizenship in YAGO
- We do not know where the incompleteness lies
  - A single person in the KB could be actually single or the KB may be incomplete

Completeness in RDF KBs

- KBs are highly incomplete
  - 1% of people have a citizenship in YAGO
- We do not know where the incompleteness lies
  - A single person in the KB could be actually single or the KB may be incomplete
- Problems for data producers and consumers

Completeness in RDF KBs

- KBs are highly incomplete
  - 1% of people have a citizenship in YAGO
- We do not know where the incompleteness lies
  - A single person in the KB could be actually single or the KB may be incomplete
- Problems for data producers and consumers
  - Consumers: no completeness guarantees for queries.
  - Producers: which parts of the KB need to be populated?

Completeness

- Defined with respect to a **query q** via a complete hypothetical KB K*.

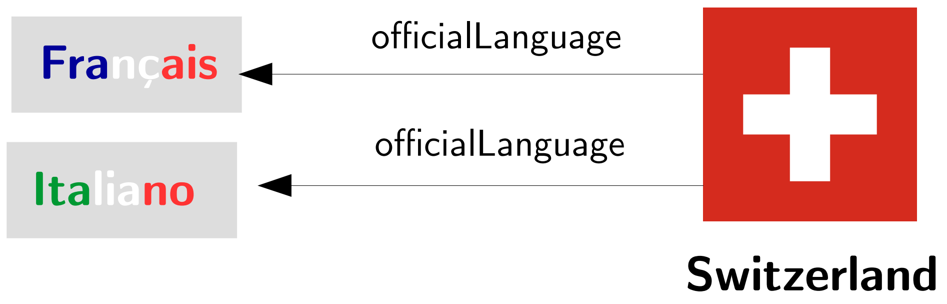
Completeness

- Defined with respect to a **query q** via a complete hypothetical KB K*.
  - A query q is complete in K, iff q(K*) $\subseteq$ q(K).

Completeness

- Defined with respect to a **query q** via a complete hypothetical KB K*.
  - A query q is complete in K, iff q(K*) ⊆ q(K).

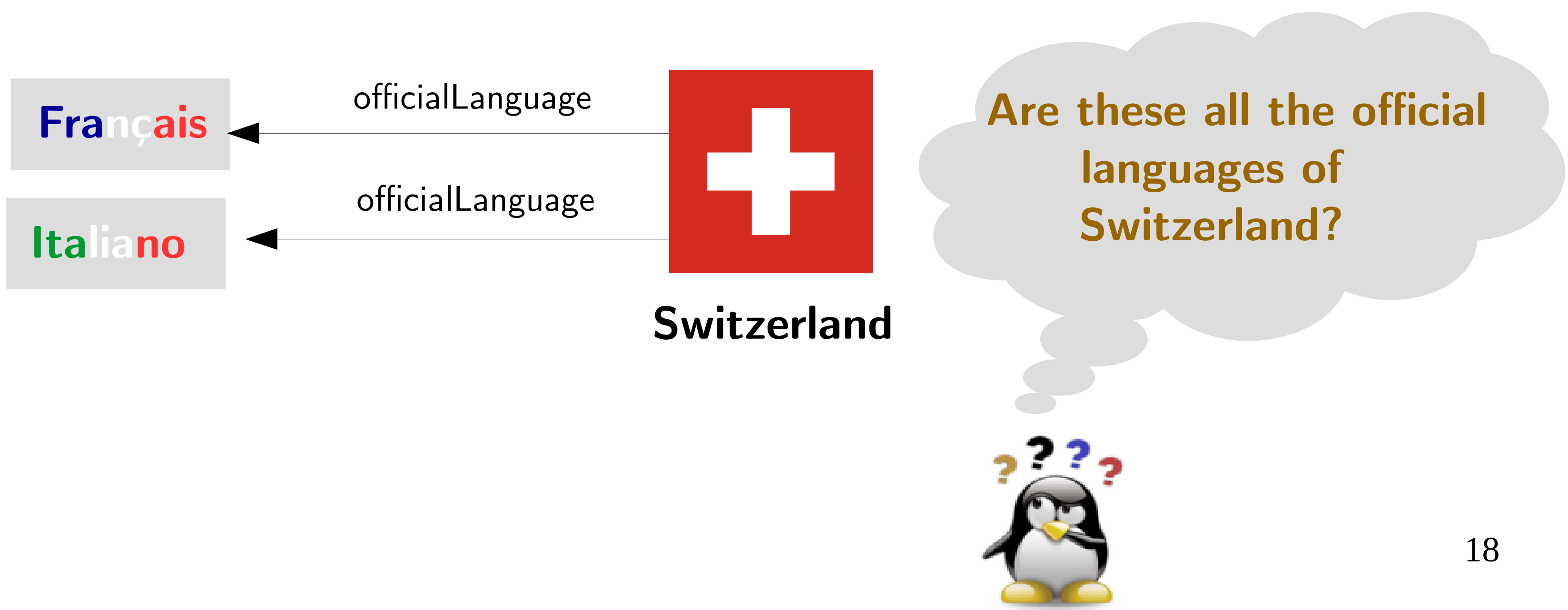
SELECT ?x WHERE { Switzerland officialLang ?x }



Completeness

- Defined with respect to a **query q** via a complete hypothetical KB K*.
  - A query q is complete in K, iff q(K*) ⊆ q(K).

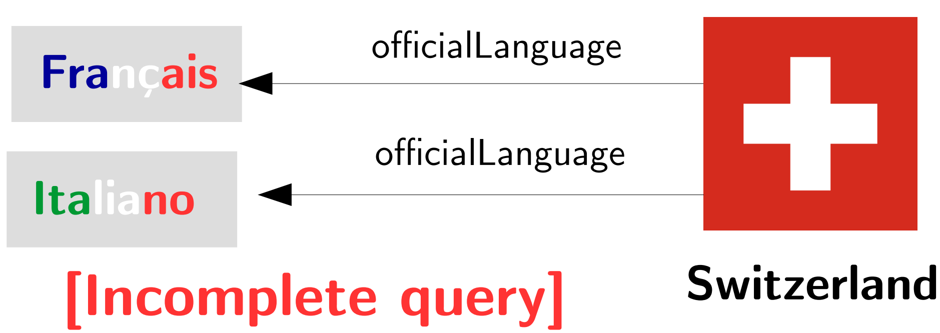
SELECT ?x WHERE { Switzerland officialLang ?x }

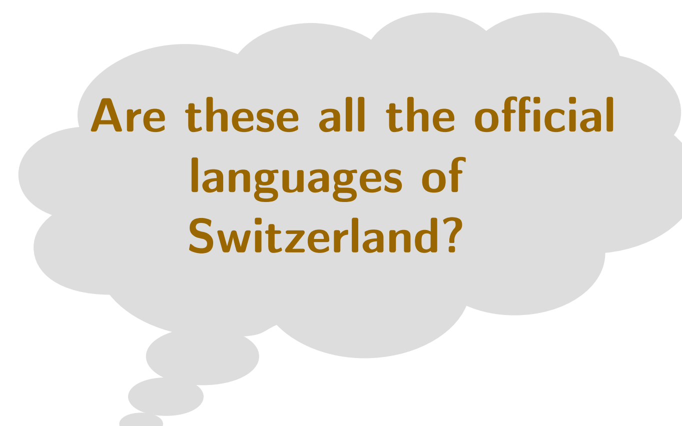


Completeness

- Defined with respect to a **query q** via a complete hypothetical KB K*.
  - A query q is complete in K, iff q(K*) ⊆ q(K).

SELECT ?x WHERE { Switzerland officialLang ?x }





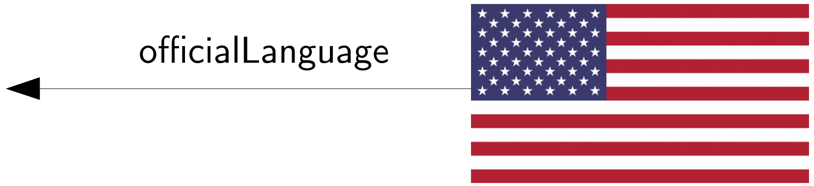
Completeness in RDF data

- Wikidata provides *no value annotations*

Completeness in RDF data

- Wikidata provides *no value annotations*

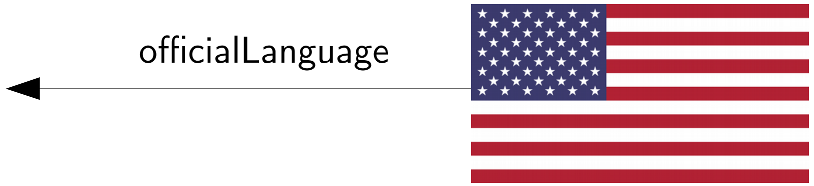
SELECT ?x WHERE { USA officialLang ?x }



Completeness in RDF data

- Wikidata provides *no value annotations*

SELECT ?x WHERE { USA officialLang ?x }

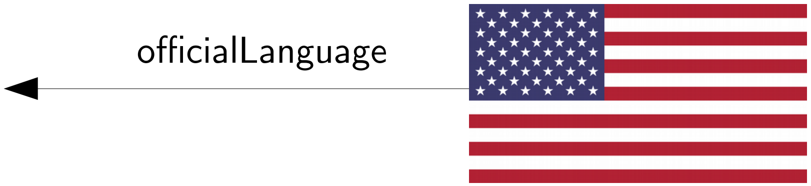


[Complete query]

Completeness in RDF data

- Wikidata provides *no value annotations*

SELECT ?x WHERE { USA officialLang ?x }



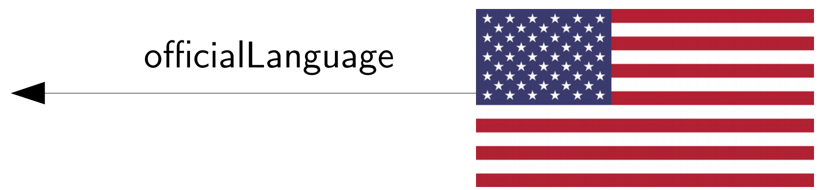
[Complete query]

- Not applicable if we know some official language

Completeness in RDF data

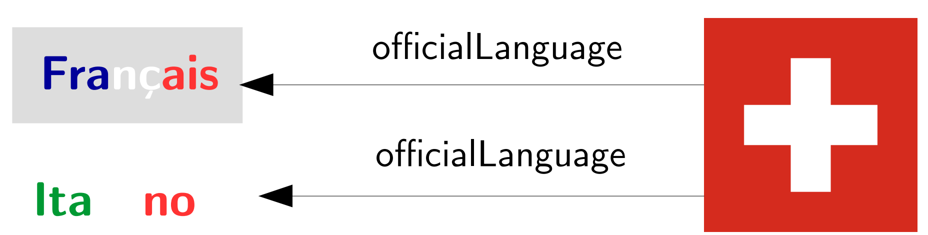
- Wikidata provides *no value annotations*

SELECT ?x WHERE { USA officialLang ?x }



[Complete query]

- Not applicable if we know some official language



Switzerland

Outline

- Completeness in RDF knowledge bases
- Completeness oracles
- Our vision
  - Representations for completeness oracles
  - Reasoning with completeness oracles
  - Enabling completeness in SPARQL
- Summary & conclusions

Completeness oracle

- Boolean function $\boldsymbol{\omega}(\mathbf{q}, \mathbf{K})$ that guesses the completeness of a query q in a KB K.

SR completeness oracle

- Function $\boldsymbol{\varpi}$ that guesses the completeness of queries of the form [Galárraga et. al, 2017]:

SELECT ?x WHERE { subject relation ?x }

SR completeness oracle

- Function $\boldsymbol{\omega}$ that guesses the completeness of queries of the form [Galárraga et. al, 2017]:

  SELECT ?x WHERE { subject relation ?x }

- We use the notation ω*(subject, relation)*

SR completeness oracle

- Function $\boldsymbol{\varpi}$ that guesses the completeness of queries of the form [Galárraga et. al, 2017]:

SELECT ?x WHERE { subject relation ?x }

- We use the notation $\varpi(subject, relation)$
- $\varpi = pca(s, r) =$ partial completeness assumption

SR completeness oracle

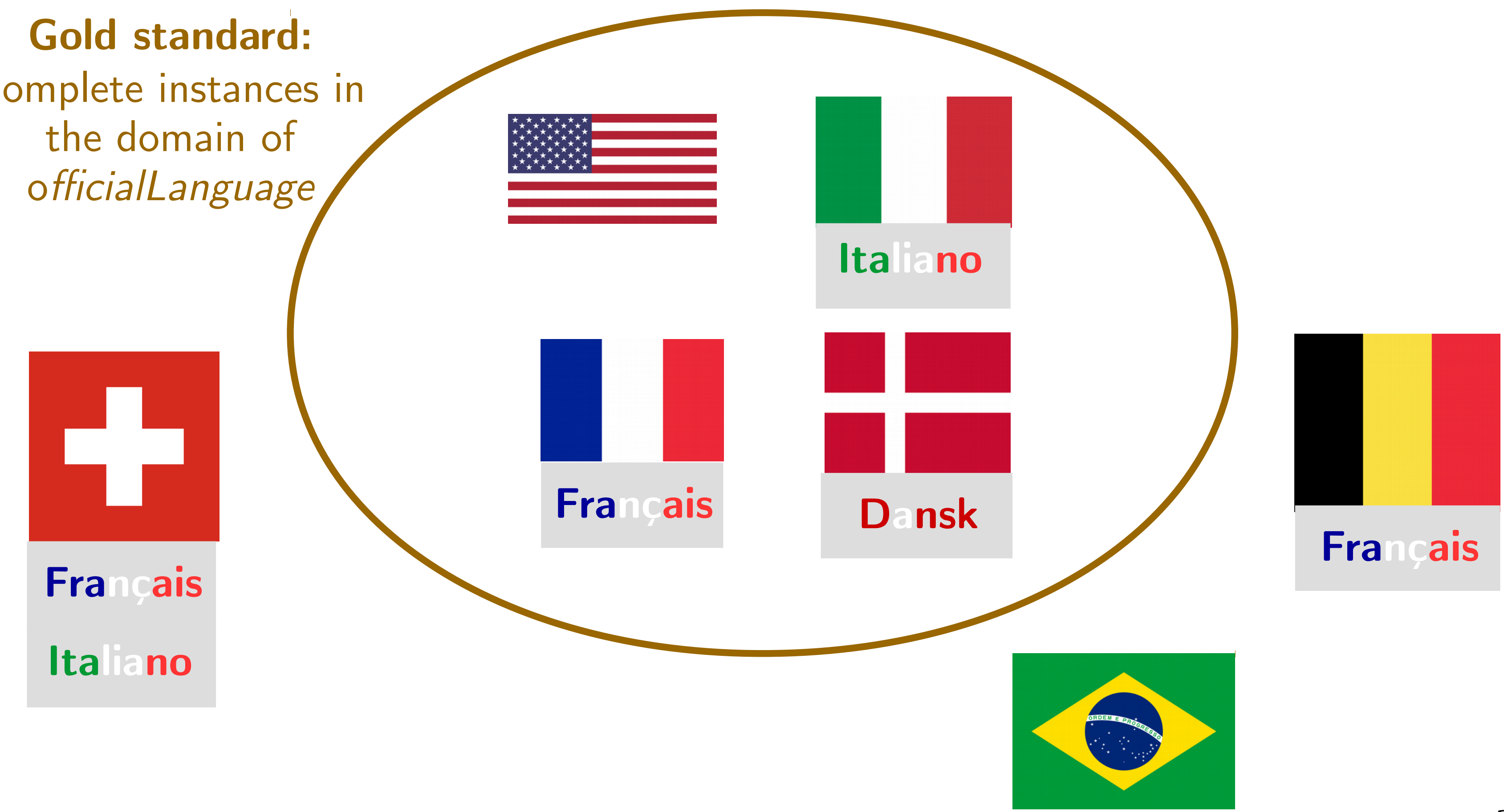
- Function $\boldsymbol{\varpi}$ that guesses the completeness of queries of the form [Galárraga et. al, 2017]:

  SELECT ?x WHERE { subject relation ?x }

- We use the notation $\varpi(subject, relation)$
- $\varpi = pca(s, r) =$ partial completeness assumption
  - Query is **complete** in KB if at least one answer is known

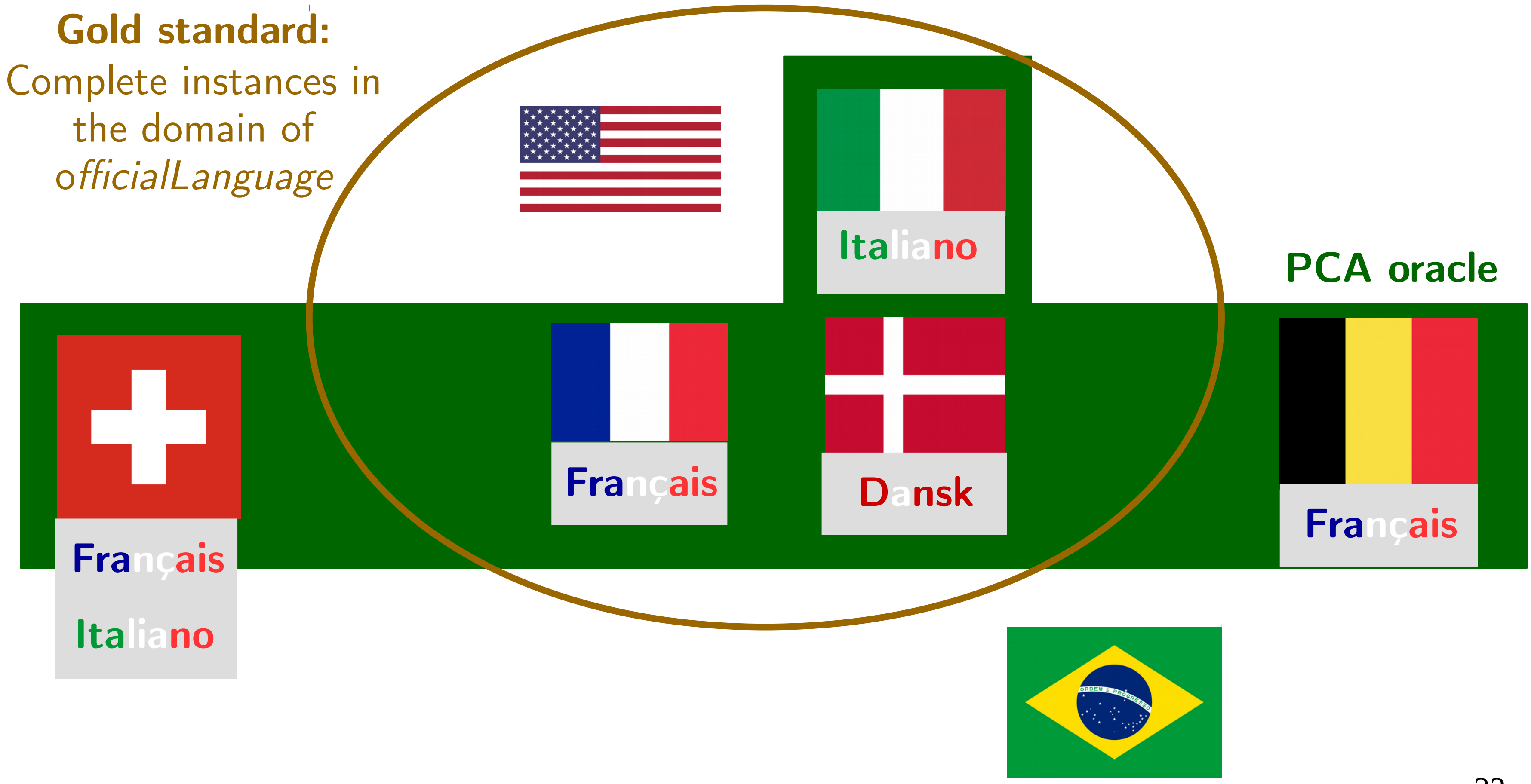
Evaluating SR oracles

$\varpi = pca(s, r) =$ partial completeness assumption



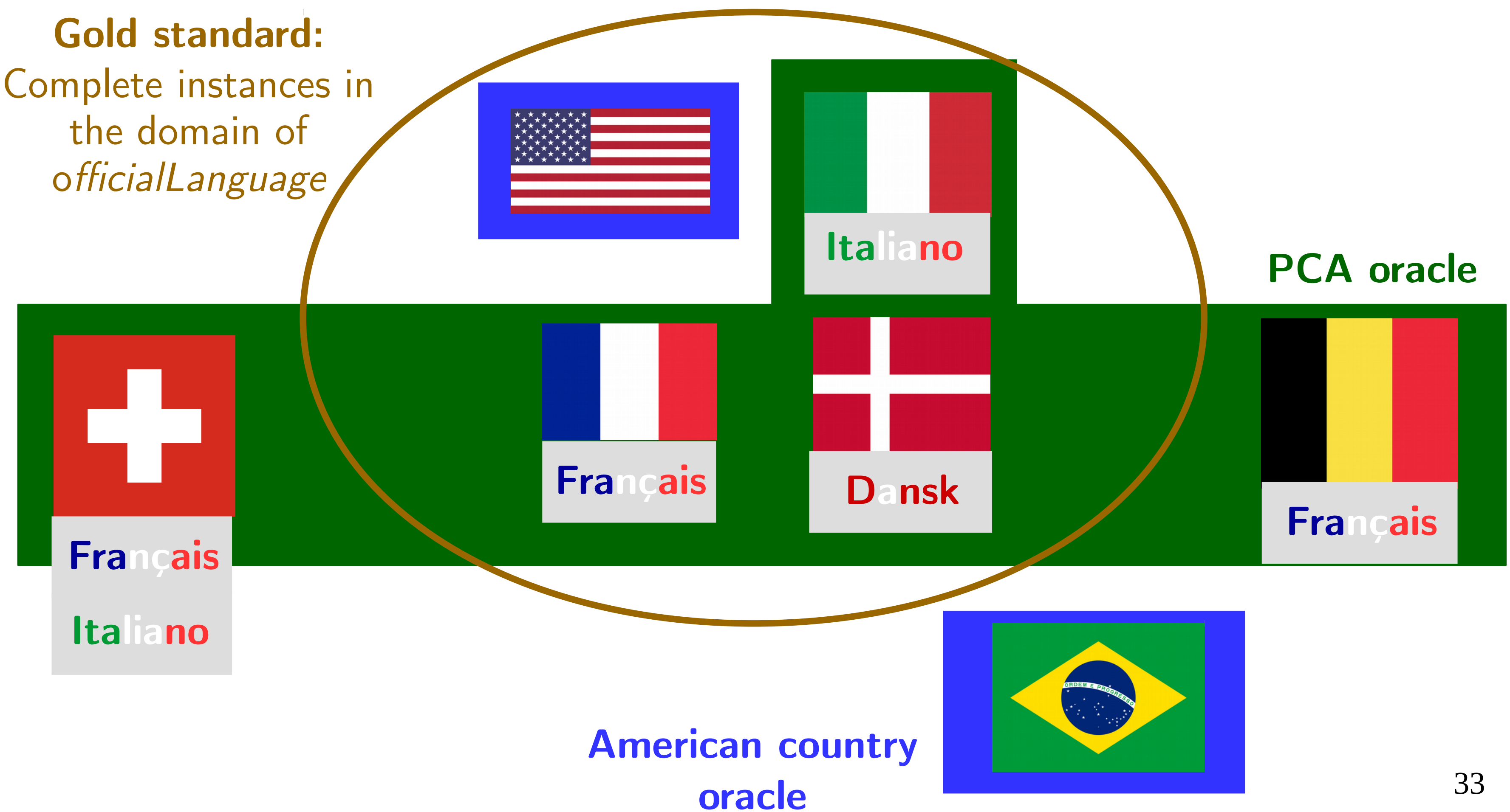
Evaluating SR oracles

$\varpi = pca(s, r) =$ partial completeness assumption



Evaluating SR oracles

$$\varpi = \textit{american-country-oracle}(s, r)$$



Evaluating SR oracles

PCA oracle

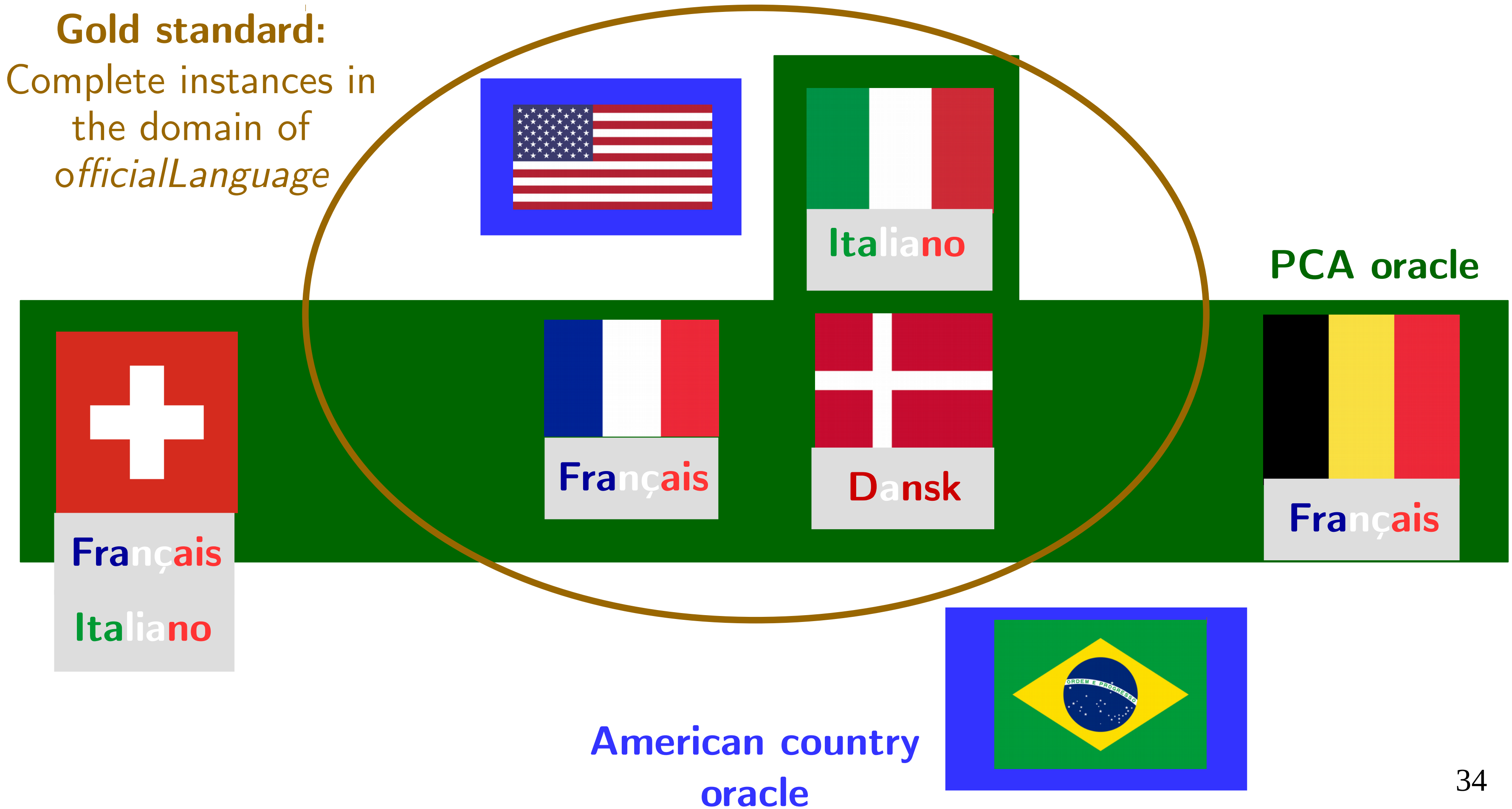
Precision = 3/5

Recall = 3/4

American country oracle

Precision = 1/2

Recall = 1/4



SR completeness oracles

- Closed World Assumption: $cwa(s, r) = true$
- PCA: $pca(s, r) = \exists\ o : r(s, o)$
- Cardinality: $card(s, r) = \#(o : r(s, o)) \geq k$
- Popular entities: $popularity_{pop}(s, r) = pop(s)$
- No-chg over time: $nochange_{chg}(s, r) = \sim chg(s, r)$
- Star : $star_{r_1,..,r_n}(s, r) = \forall\ i \in \{1,..,n\} : \exists\ o : r_i(s, o)$
- Class: $class_c(s, r) = type(s, c)$
- Rule mining oracle

Rule mining SR oracle

- Based on completeness rules

notype(x, Adult), type(x, Person) ⇒ complete(x, hasChild)

dateOfDeath(x, y), lessThan_1(x, placeOfDeath)⇒ incomplete(x, placeOfDeath)

Rule mining SR oracle

- Based on completeness rules

  notype(x, Adult), type(x, Person) ⇒ complete(x, hasChild)

  dateOfDeath(x, y), lessThan_1(x, placeOfDeath)⇒ incomplete(x, placeOfDeath)

- Learned using the AMIE [Galárraga et. al, 2013] rule mining system
  - On gold standard built via crowdsourcing

Rule mining SR oracle

- Based on completeness rules

  notype(x, Adult), type(x, Person) ⇒ complete(x, hasChild)

  dateOfDeath(x, y), lessThan_1(x, placeOfDeath)⇒ incomplete(x, placeOfDeath)

- Learned using the AMIE [Galárraga et. al, 2013] rule mining system
  - On gold standard built via crowdsourcing
  - 100% F1-measure for functional relations, quite good for relations *hasChild*, *graduatedFrom*

Outline

- Completeness in RDF knowledge bases
- Completeness oracles
- Our vision
  - Representations for completeness oracles
  - Reasoning with completeness oracles
  - Enabling completeness in SPARQL
- Summary & conclusions

Representing completeness oracles

- Extensional approach [Darari, et. Al, 2013]
  - An oracle is a collection of completeness statements about queries

Representing completeness oracles

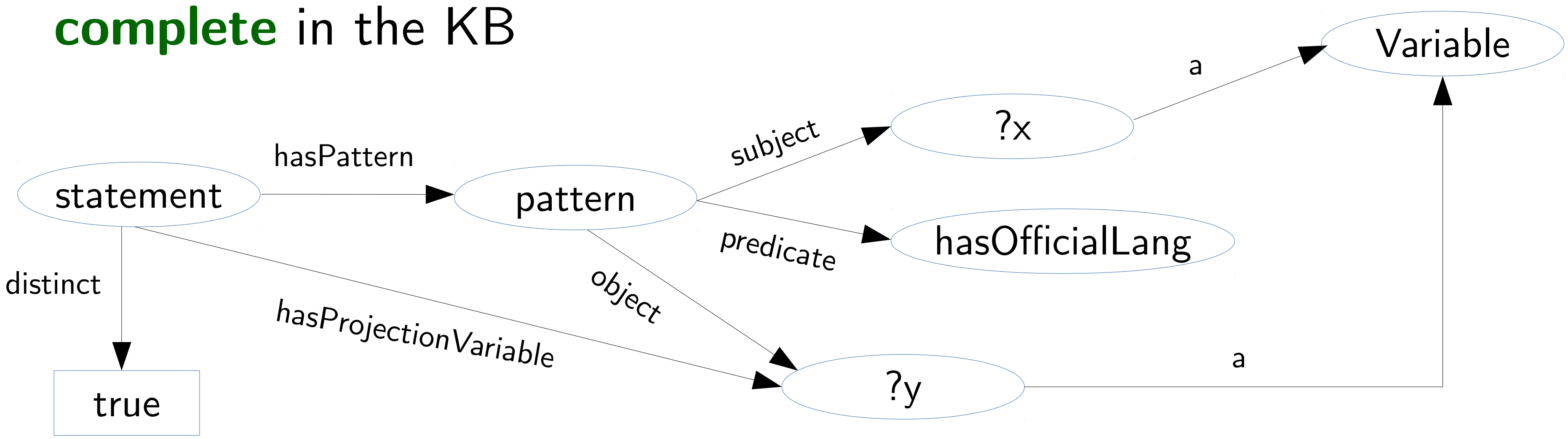
- Extensional approach [Darari, et. Al, 2013]
  - An oracle is a collection of completeness statements about queries

SELECT DISTINCT ?y WHERE { ?x hasOfficialLang ?y } is **complete** in the KB

Representing completeness oracles

- Extensional approach [Darari, et. Al, 2013]
  - An oracle is a collection of completeness statements about queries

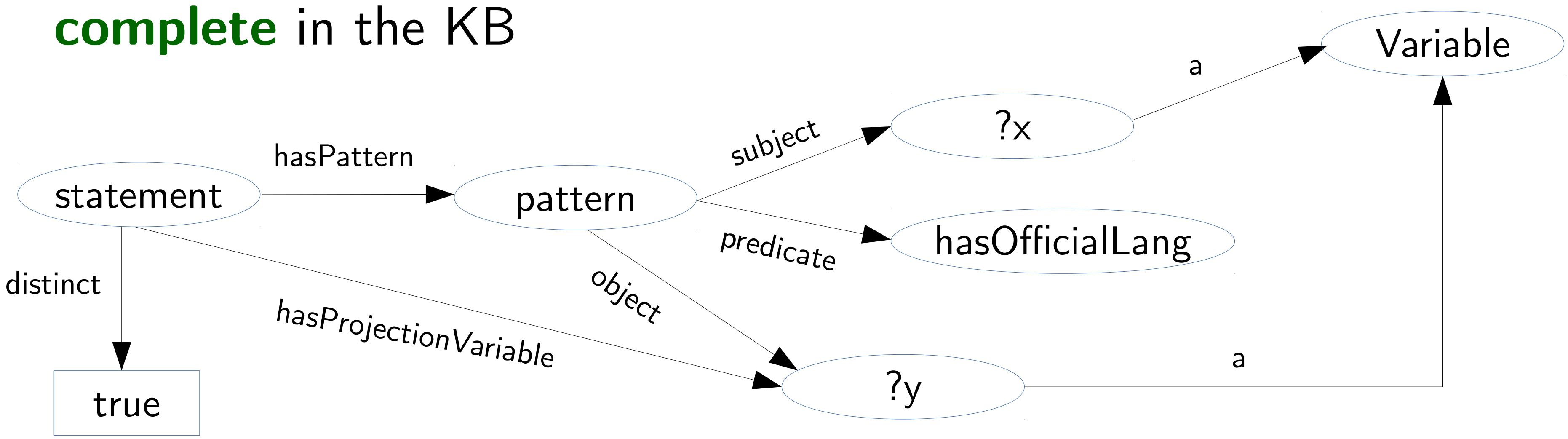
SELECT DISTINCT ?y WHERE { ?x hasOfficialLang ?y } is **complete** in the KB



Representing completeness oracles

- Extensional approach [Darari, et. Al, 2013]
  - A call to the oracle asks for the existence of the query in the graph

SELECT DISTINCT ?y WHERE { ?x hasOfficialLang ?y } is **complete** in the KB

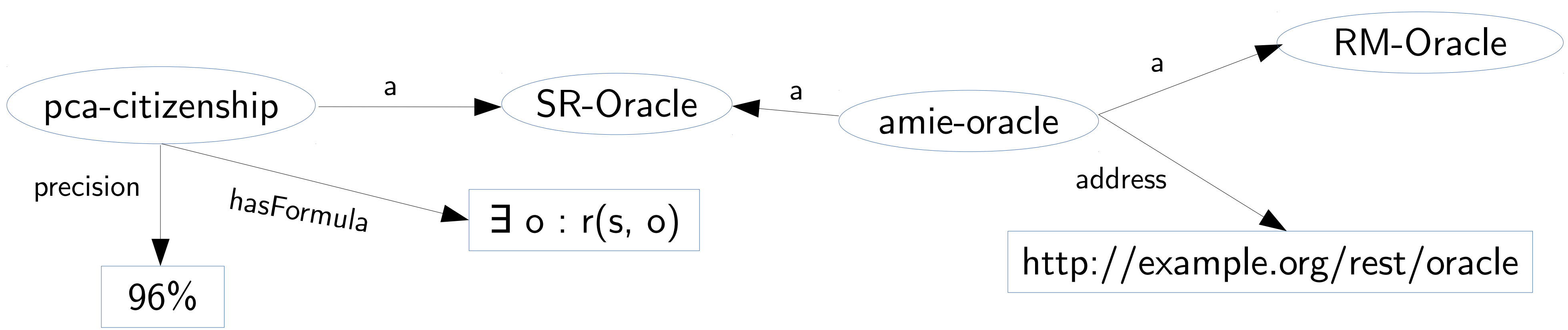


Representing completeness oracles

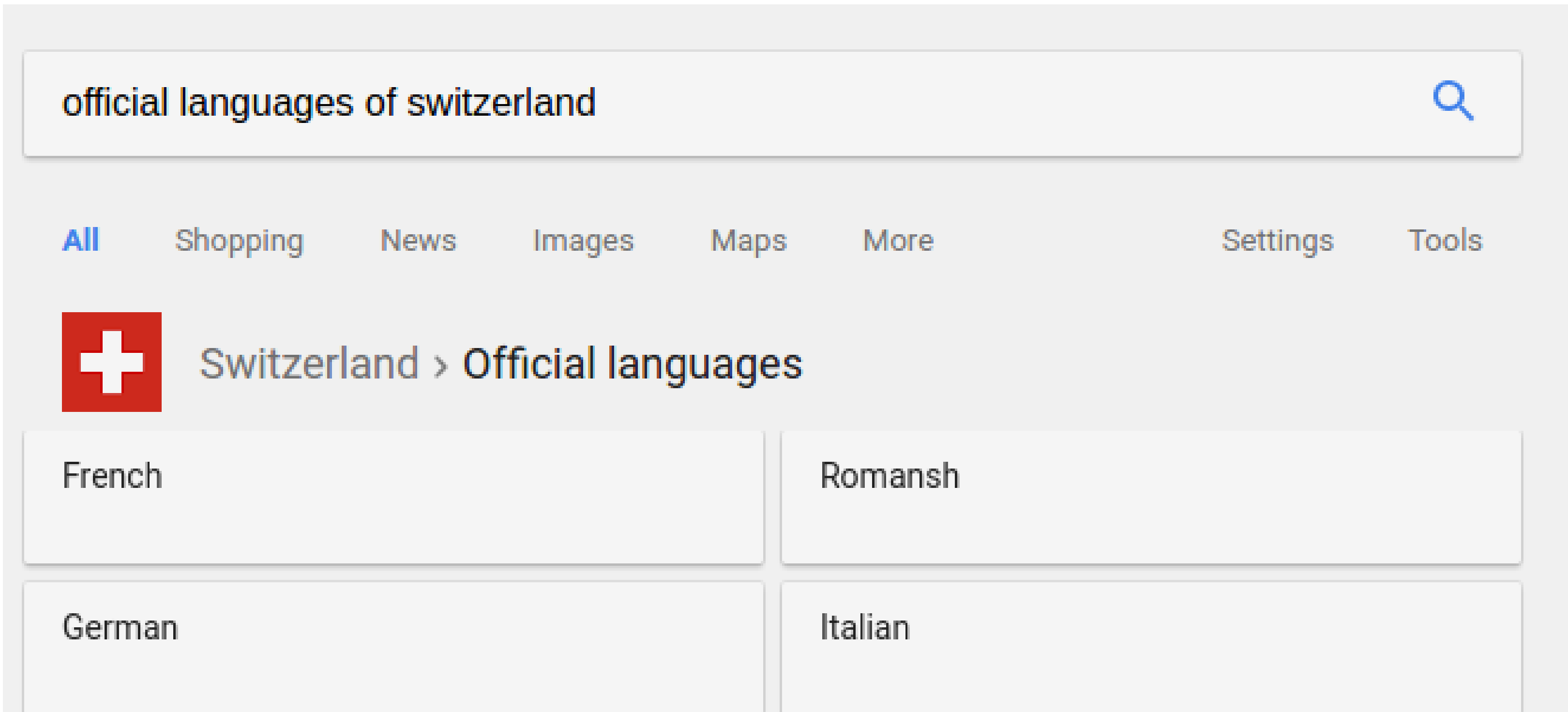
- Intensional approach
  - The oracle logic is embedded as a lambda function or a link to a program or resource

Representing completeness oracles

- Intensional approach
  - The oracle logic is embedded as a lambda function or a link to a program or resource

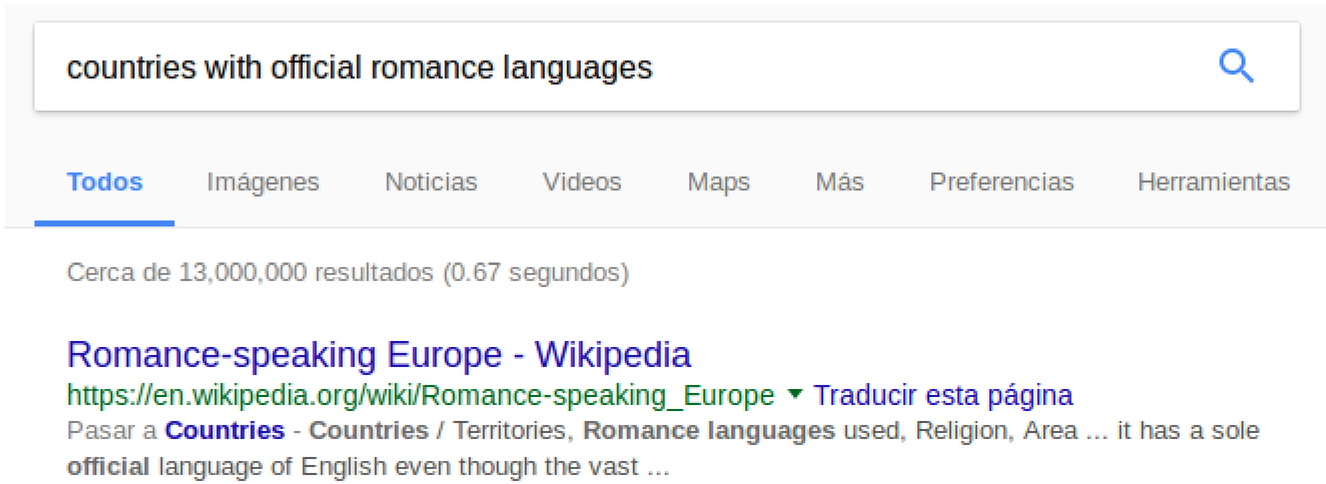


Providing completeness guarantees

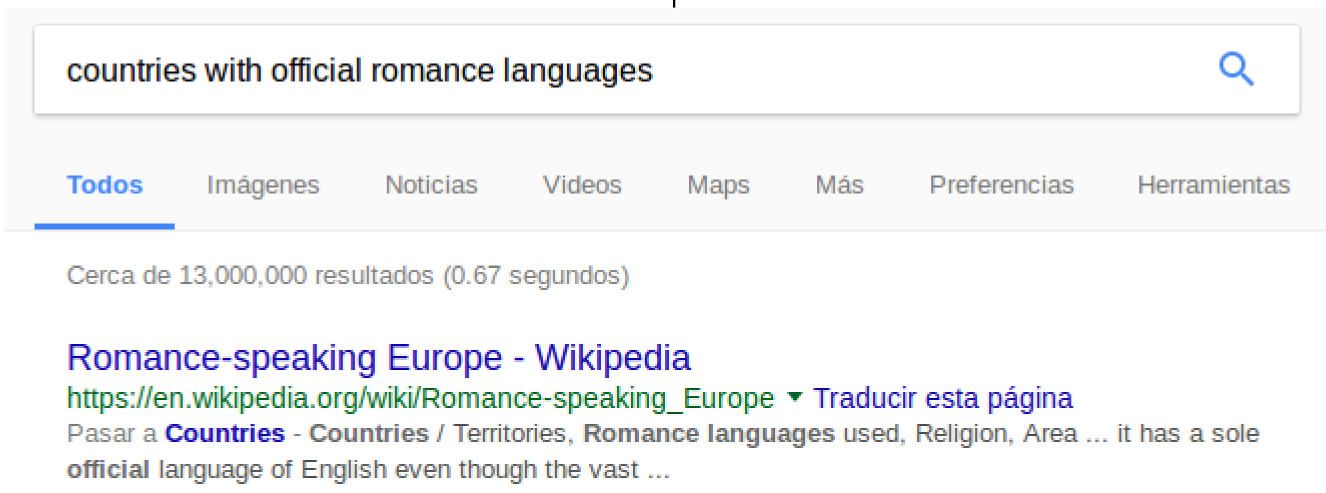


List of results is complete according to oracle ϖ with confidence X

Providing completeness guarantees

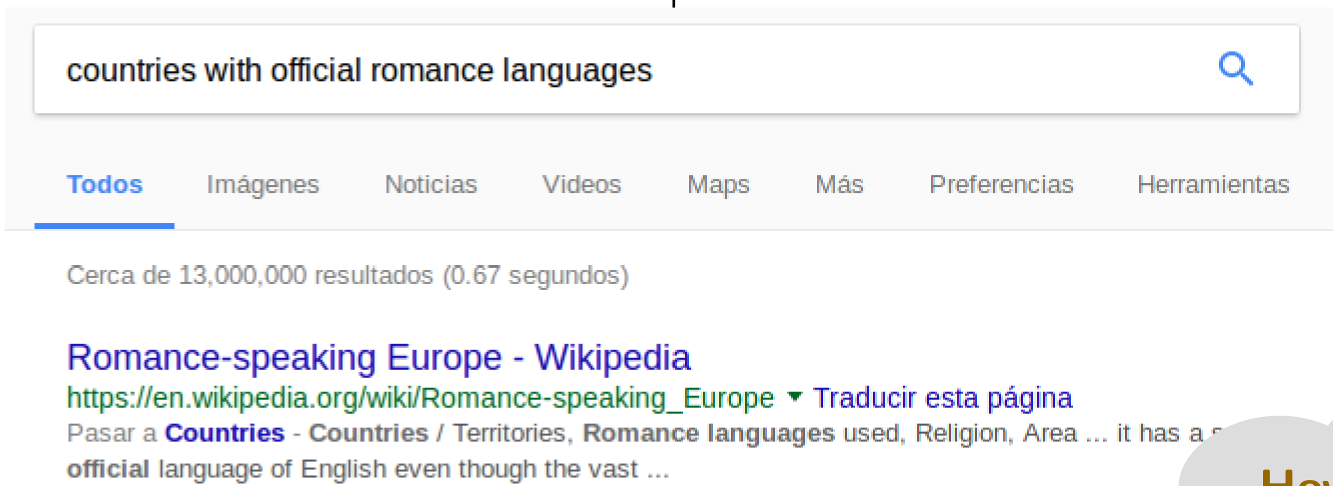


Providing completeness guarantees




```
SELECT ?country WHERE {
    ?country officialLang ?lang .
    ?lang family Romance .
}
```

Providing completeness guarantees




```
SELECT ?country WHERE {
    ?country officialLang ?lang .
    ?lang family Romance .
}
```

How to provide completeness guarantees for arbitrary queries?

Outline

- Completeness in RDF knowledge bases
- Completeness oracles
- Our vision
  - Representations for completeness oracles
  - Reasoning with completeness oracles
  - Enabling completeness in SPARQL

D completeness oracles

- Oracle $\boldsymbol{\omega}_{\mathbf{d}}$ for the completeness of queries:

  SELECT DISTINCT ?x WHERE { ?x relation ?y }

  SELECT DISTINCT ?y WHERE { ?x relation ?y }

D completeness oracles

- Oracle $\boldsymbol{\omega}_{\mathbf{d}}$ for the completeness of queries:

SELECT DISTINCT ?x WHERE { ?x relation ?y }

SELECT DISTINCT ?y WHERE { ?x relation ?y }

- We use the notation $\omega_d(relation)$ *or* $\omega_d(relation^{-1})$

SELECT DISTINCT ?y WHERE { ?x officialLang ?y }

D completeness oracles

- Oracle $\boldsymbol{\omega}_{\mathbf{d}}$ for the completeness of queries:

SELECT DISTINCT ?x WHERE { ?x relation ?y }

SELECT DISTINCT ?y WHERE { ?x relation ?y }

- We use the notation ω_d*(relation) or* ω_d*(relation*$^{-1}$*)*

SELECT DISTINCT ?y WHERE { ?x officialLang ?y }

- If $\boldsymbol{\omega}_{\mathbf{d}}$ returns true, $\boldsymbol{\omega}_{\mathbf{d}}$ states that the KB knows all languages that are official in some country

Completeness guarantees for arbitrary queries

- Write completeness annotations for every possible type of query
  - It requires a large amount of effort

Completeness guarantees for arbitrary queries

- Write completeness annotations for every possible type of query
  - It requires a large amount of effort
- Reuse existing SR and D oracles

Completeness guarantees for arbitrary queries

- Write completeness annotations for every possible type of query
  - It requires a large amount of effort
- Reuse existing SR and D oracles

```
SELECT ?country WHERE {
    ?country officialLang ?lang .
    ?lang family Romance .
}
```

Completeness guarantees for arbitrary queries

- Write completeness annotations for every possible type of query
  - It requires a large amount of effort
- Reuse existing SR and D oracles

```
SELECT ?country WHERE {
    ?country officialLang ?lang .
    ?lang family Romance .
}
```

ϖ' =

Completeness guarantees for arbitrary queries

- Write completeness annotations for every possible type of query
  - It requires a large amount of effort
- Reuse existing SR and D oracles

```
SELECT ?country WHERE {
    ?country officialLang ?lang .
    ?lang family Romance .
}
```

$\omega' = \omega(\text{Romance}, \text{family}^{-1})$

Completeness guarantees for arbitrary queries

- Write completeness annotations for every possible type of query
  - It requires a large amount of effort
- Reuse existing SR and D oracles

SELECT ?country WHERE {
    ?country officialLang ?lang .
    ?lang family Romance .
}

$$\omega' = \omega(\text{Romance}, \text{family}^{-1}) \wedge (\bigwedge_{l:\text{family}(l,\ \text{Romance})} \omega(l, \text{officialLang}^{-1}))$$

Completeness guarantees for arbitrary queries

- Write completeness annotations for every possible type of query
  - It requires a large amount of effort
- Reuse existing SR and D oracles

```
SELECT ?country WHERE {
    ?country officialLang ?lang
    ?lang family Romance .
}
```

It will generate false negatives

$$\omega' = \omega(\text{Romance}, \text{family}^{-1}) \wedge \left(\bigwedge_{l:\text{family}(l,\ \text{Romance})} \omega(l, \text{officialLang}^{-1}) \right)$$

Completeness guarantees for arbitrary queries

- Write completeness annotations for every possible type of query
  - It requires a large amount of effort
- Reuse existing SR and D oracles

If the KB misses Ligurian, this term returns false

```
SELECT ?country WHERE {
    ?country officialLang ?lang .
    ?lang family Romance .
}
```

$$\omega' = \omega(\text{Romance}, \text{family}^{-1}) \wedge \left(\bigwedge_{l:\text{family}(l,\ \text{Romance})} \omega(l, \text{officialLang}^{-1})\right)$$

Completeness guarantees for arbitrary queries

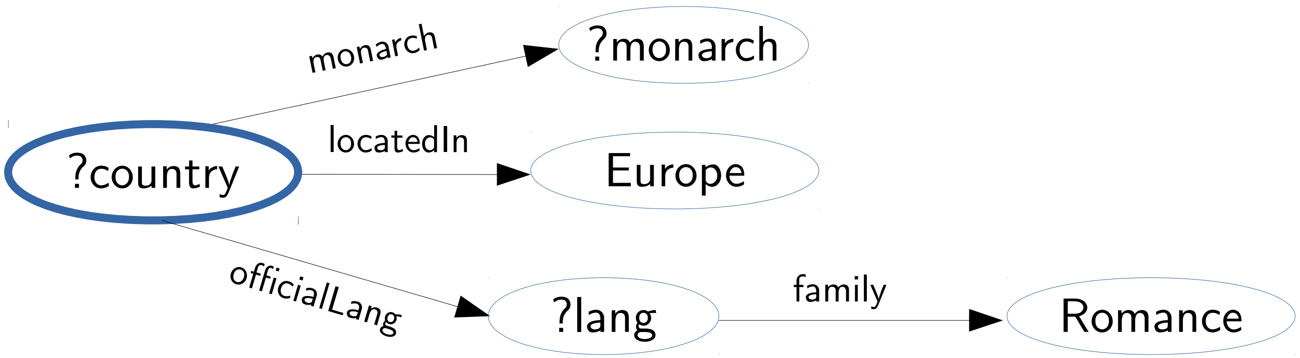
- Write completeness annotations for every possible type of query
  - It requires a large amount of effort
- Reuse existing SR and D oracles

SELECT ?country WHERE {
    ?country officialLang ?lang .
    ?lang family Romance .
}

Even though this term does not care, because Ligurian is not official in any country

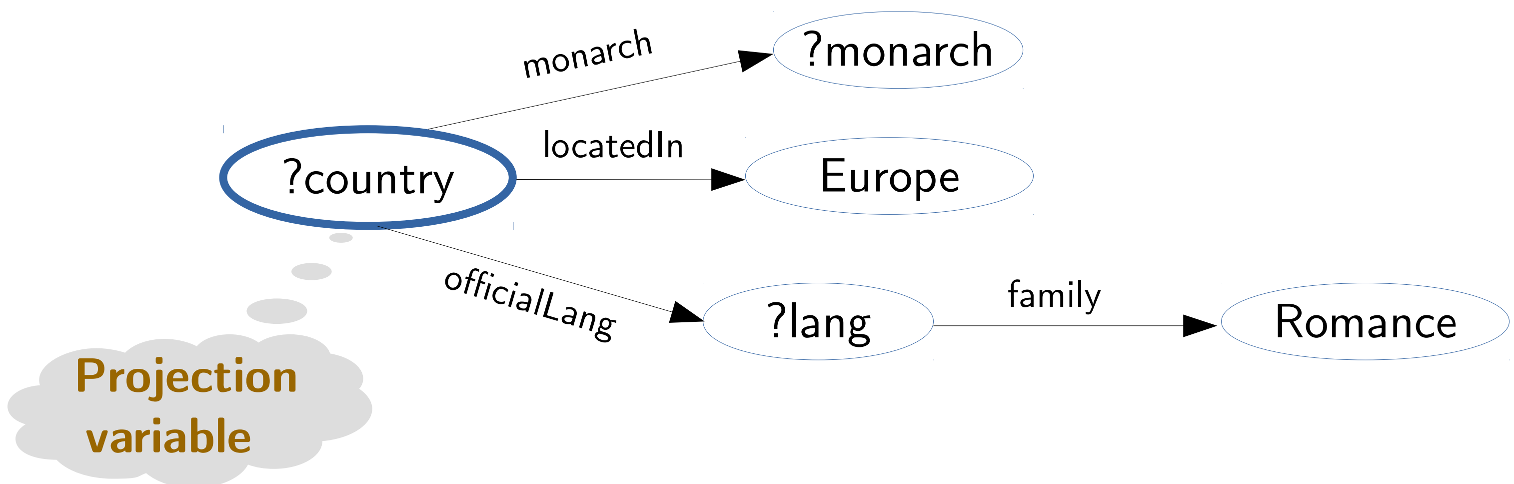
$$\omega' = \omega(\text{Romance}, \text{family}^{-1}) \wedge \left(\bigwedge_{l:\text{family}(l,\ \text{Romance})} \omega(l, \text{officialLang}^{-1})\right)$$

Automatic oracle composition



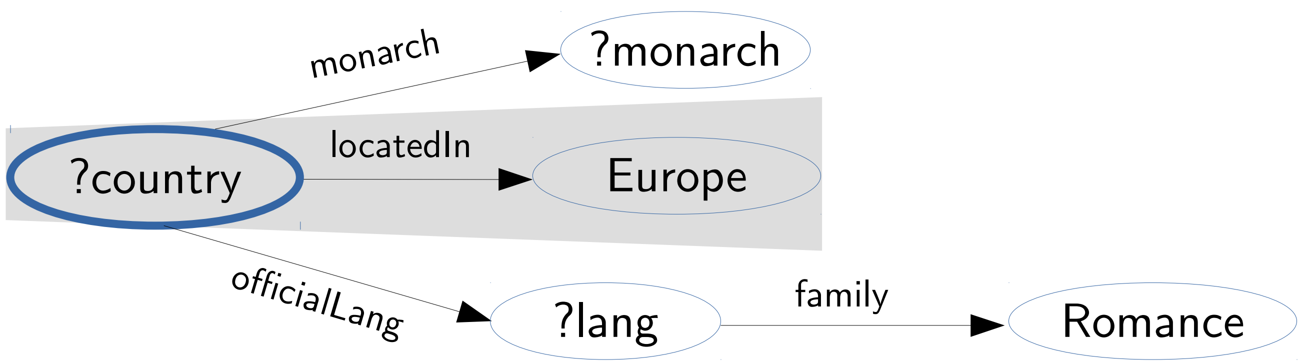

```
SELECT ?country WHERE {
    ?country monarch ?monarch .
    ?country locatedIn Europe .
    ?country officialLang ?lang .
    ?lang family Romance .
}
```

Automatic oracle composition



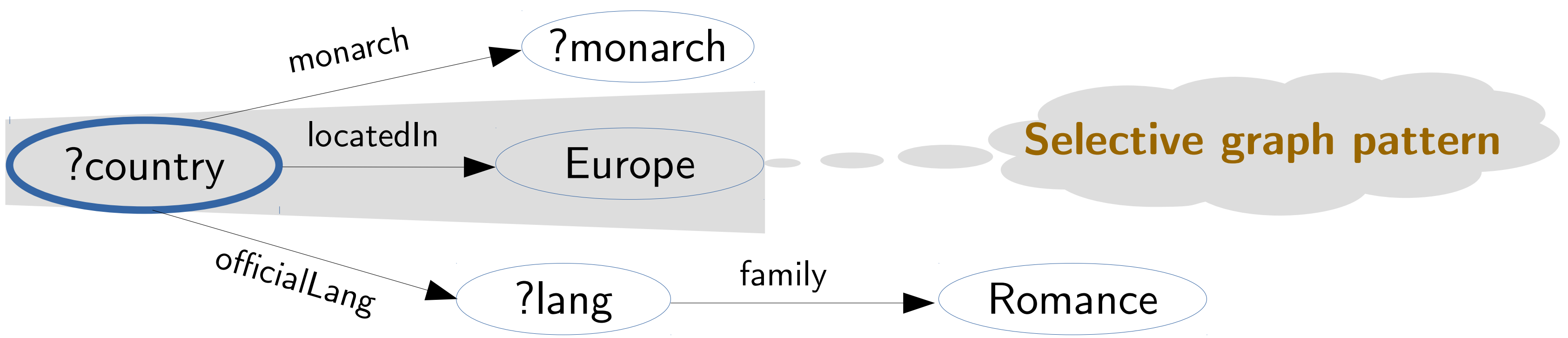
SELECT ?country WHERE {
 ?country monarch ?monarch .
 ?country locatedIn Europe .
 ?country officialLang ?lang .
 ?lang family Romance .
}

Automatic oracle composition



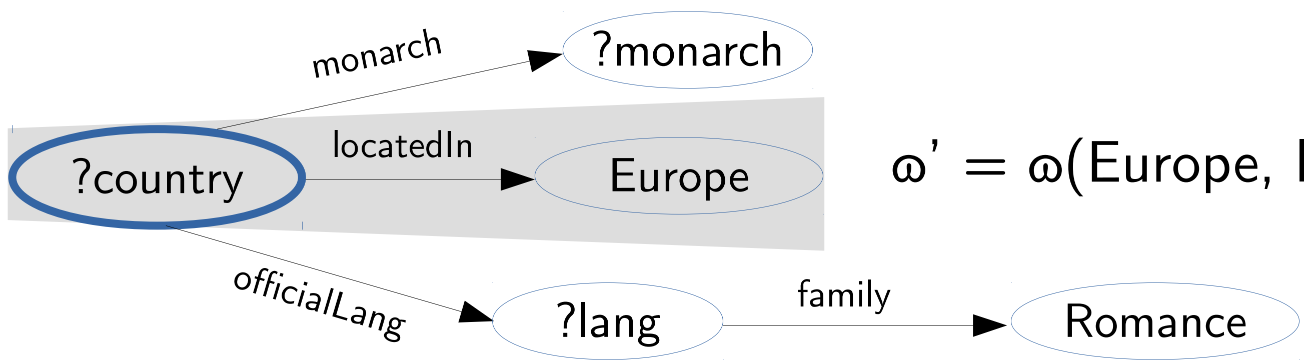
SELECT ?country WHERE {
    ?country monarch ?monarch .
    ?country locatedIn Europe .
    ?country officialLang ?lang .
    ?lang family Romance .
}

Automatic oracle composition



SELECT ?country WHERE {
    ?country monarch ?monarch .
    ?country locatedIn Europe .
    ?country officialLang ?lang .
    ?lang family Romance .
}

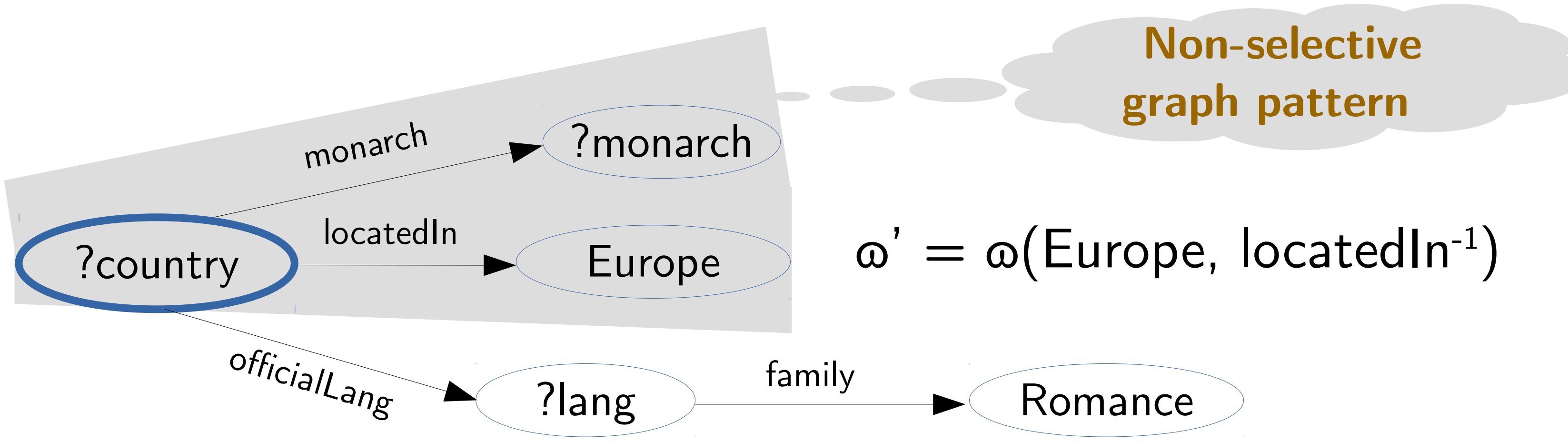
Automatic oracle composition



$\varpi' = \varpi(\text{Europe, locatedIn}^{-1})$

```
SELECT ?country WHERE {
    ?country monarch ?monarch .
    ?country locatedIn Europe .
    ?country officialLang ?lang .
    ?lang family Romance .
}
```

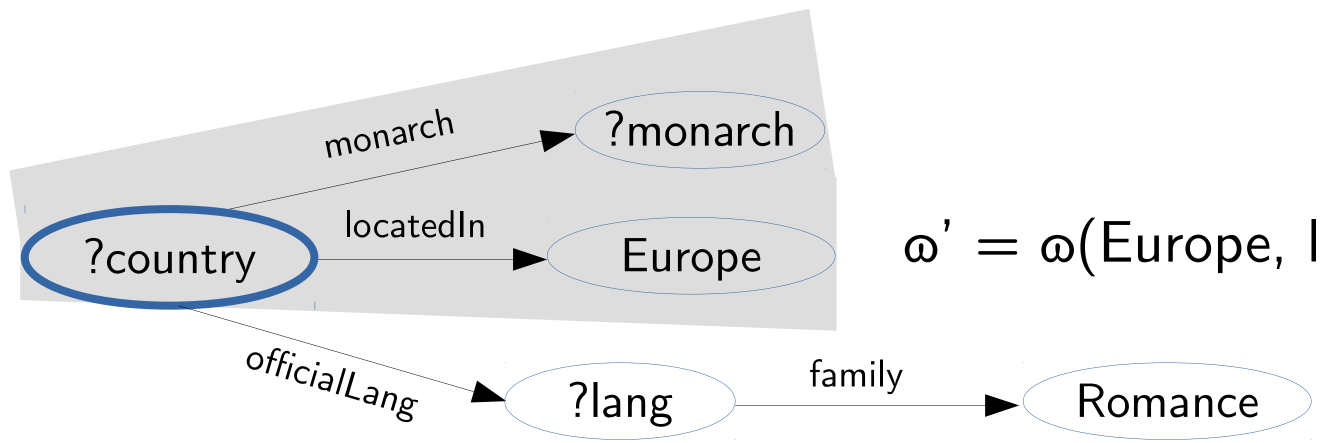
Automatic oracle composition



$\omega' = \omega(\text{Europe, locatedIn}^{-1})$

SELECT ?country WHERE {
    ?country monarch ?monarch .
    ?country locatedIn Europe .
    ?country officialLang ?lang .
    ?lang family Romance .
}

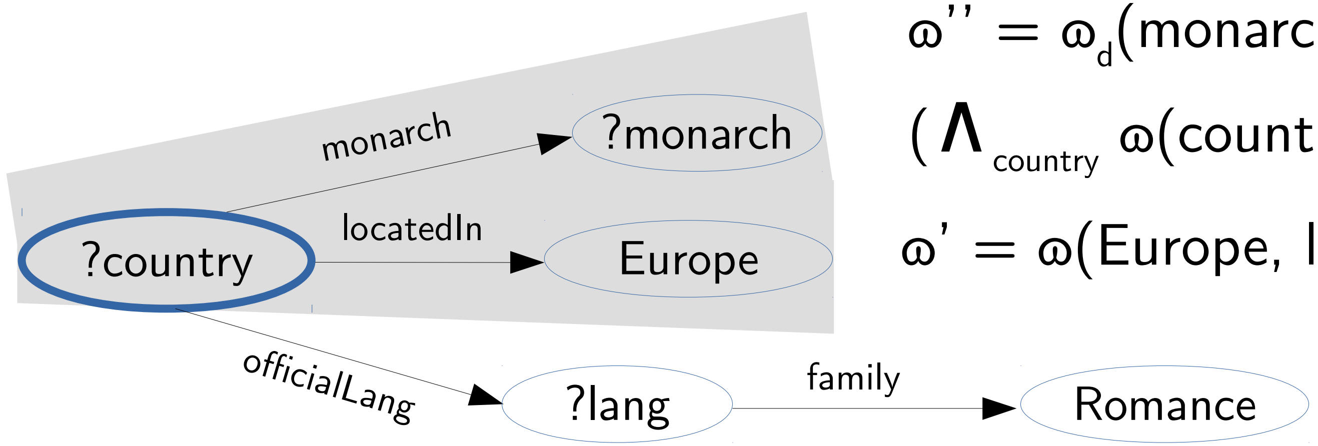
Automatic oracle composition



$\omega' = \omega(\text{Europe, locatedIn}^{-1})$

SELECT ?country WHERE {
  ?country monarch ?monarch .
  ?country locatedIn Europe .
  ?country officialLang ?lang .
  ?lang family Romance .
}

Automatic oracle composition



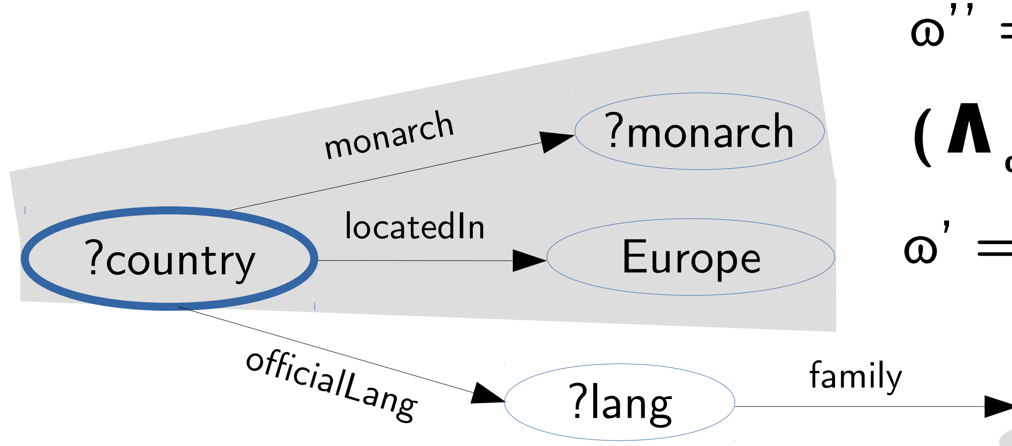
$\omega'' = \omega_d(\text{monarch}) \wedge$

$(\bigwedge_{\text{country}} \omega(\text{country, monarch}))$

$\omega' = \omega(\text{Europe, locatedIn}^{-1})$

```
SELECT ?country WHERE {
    ?country monarch ?monarch .
    ?country locatedIn Europe .
    ?country officialLang ?lang .
    ?lang family Romance .
}
```

Automatic oracle composition



$\omega'' = \omega_d(\text{monarch}) \wedge$

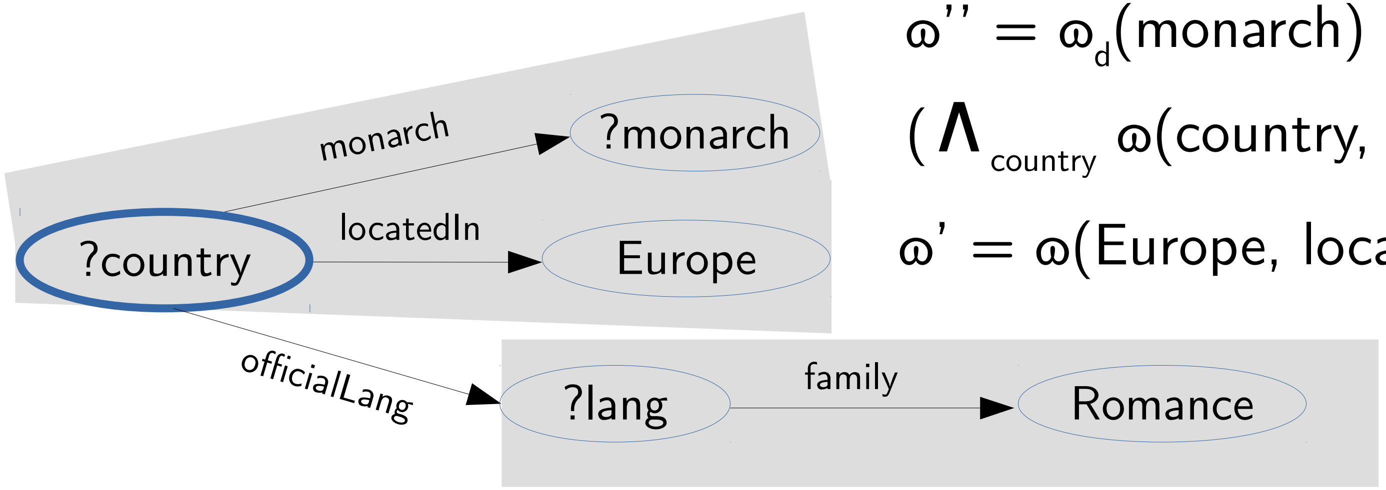
$\left(\bigwedge_{\text{country}} \boldsymbol{\omega(\text{country, monarch})}\right)$

$\omega' = \omega(\text{Europe}, \text{locatedIn}^{-1})$

not needed in case of Set semantics

```
SELECT ?country WHERE {
    ?country monarch ?monarch .
    ?country locatedIn Europe .
    ?country officialLang ?lang .
    ?lang family Romance .
}
```

Automatic oracle composition



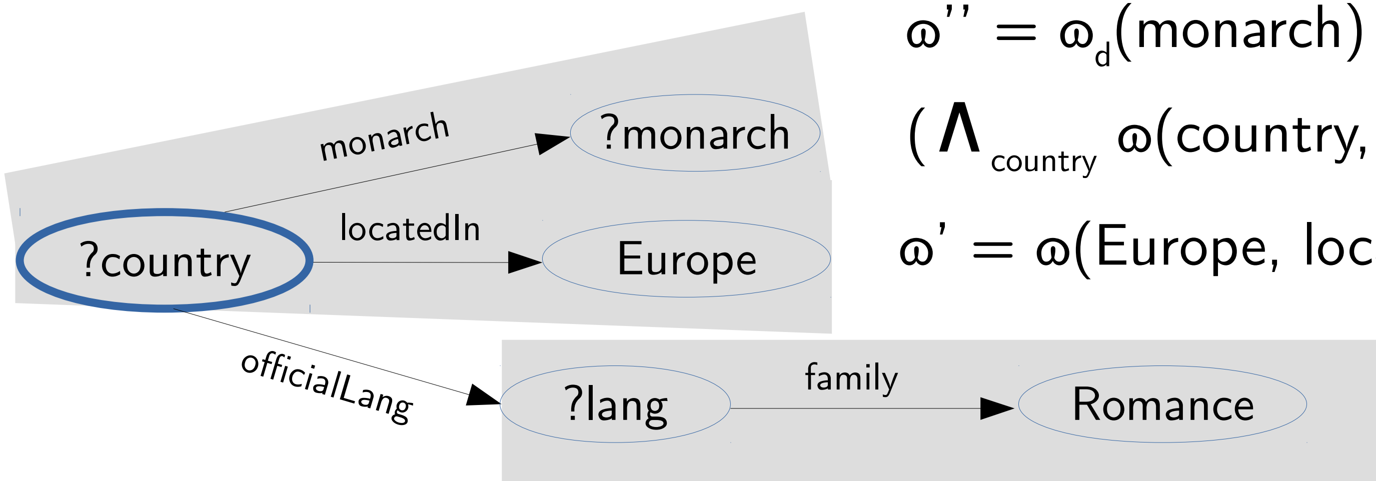
$\omega'' = \omega_d(\text{monarch}) \wedge$

$(\bigwedge_{\text{country}} \omega(\text{country, monarch}))$

$\omega' = \omega(\text{Europe, locatedIn}^{-1})$

```
SELECT ?country WHERE {
    ?country monarch ?monarch .
    ?country locatedIn Europe .
    ?country officialLang ?lang .
    ?lang family Romance .
}
```

Automatic oracle composition



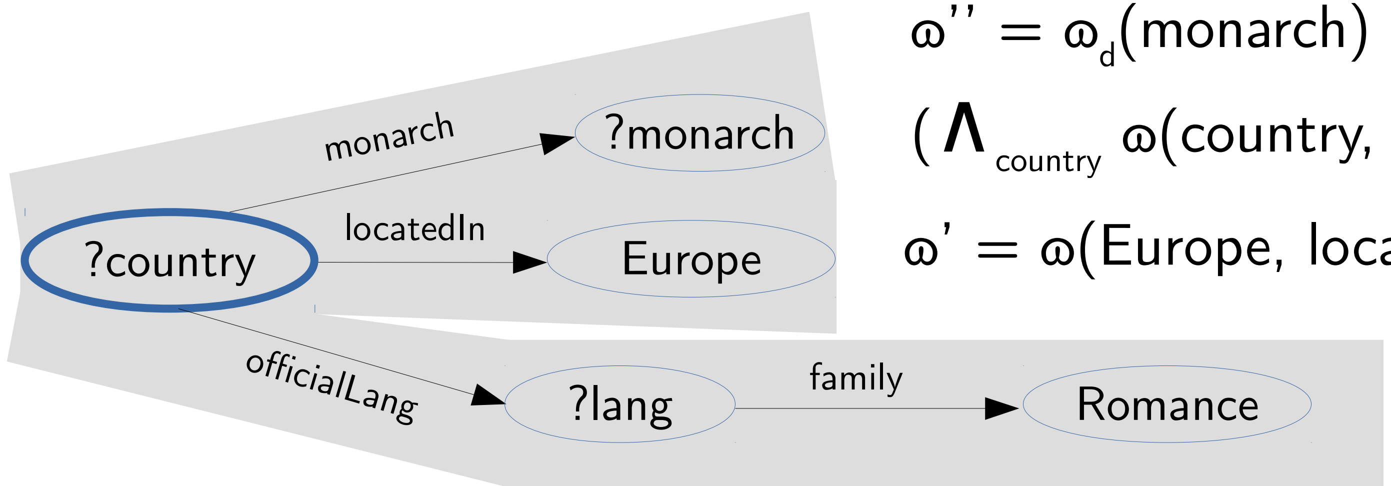
$$\varpi'' = \varpi_d(\text{monarch}) \wedge \left(\bigwedge_{\text{country}} \varpi(\text{country, monarch})\right)$$

$$\varpi' = \varpi(\text{Europe, locatedIn}^{-1})$$

$$\varpi^* = \varpi(\text{Romance, family}^{-1})$$

```
SELECT ?country WHERE {
    ?country monarch ?monarch .
    ?country locatedIn Europe .
    ?country officialLang ?lang .
    ?lang family Romance .
}
```

Automatic oracle composition



$\varpi'' = \varpi_d(\text{monarch}) \wedge$

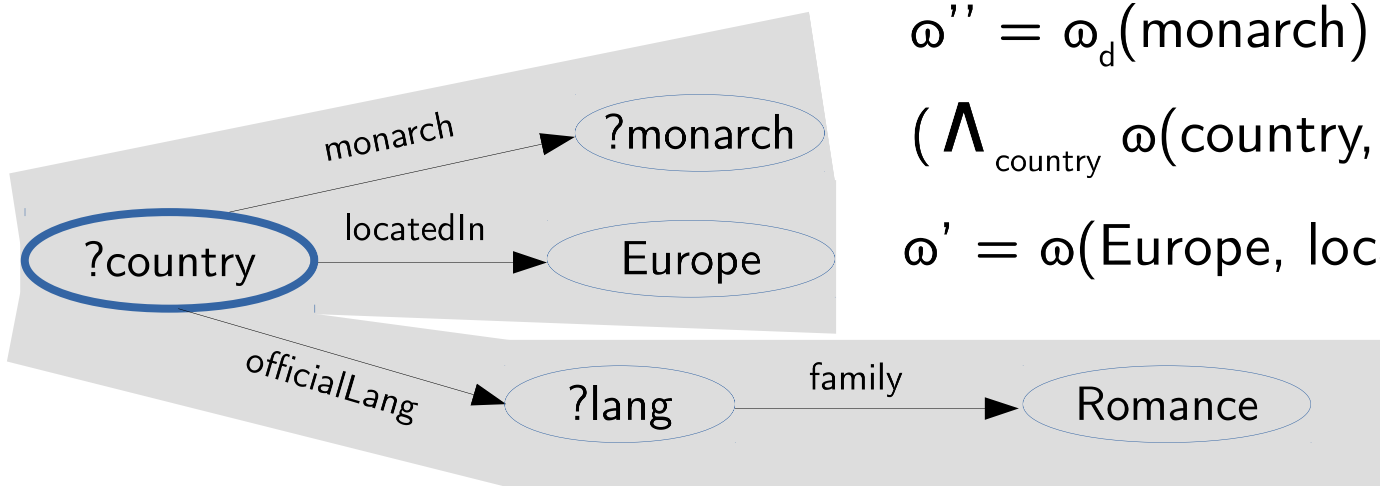
$(\bigwedge_{\text{country}} \varpi(\text{country, monarch}))$

$\varpi' = \varpi(\text{Europe, locatedIn}^{-1})$

$\varpi^* = \varpi(\text{Romance, family}^{-1})$

```
SELECT ?country WHERE {
    ?country monarch ?monarch .
    ?country locatedIn Europe .
    ?country officialLang ?lang .
    ?lang family Romance .
}
```

Automatic oracle composition



$\varpi'' = \varpi_d(\text{monarch}) \wedge$

$(\bigwedge_{\text{country}} \varpi(\text{country, monarch}))$

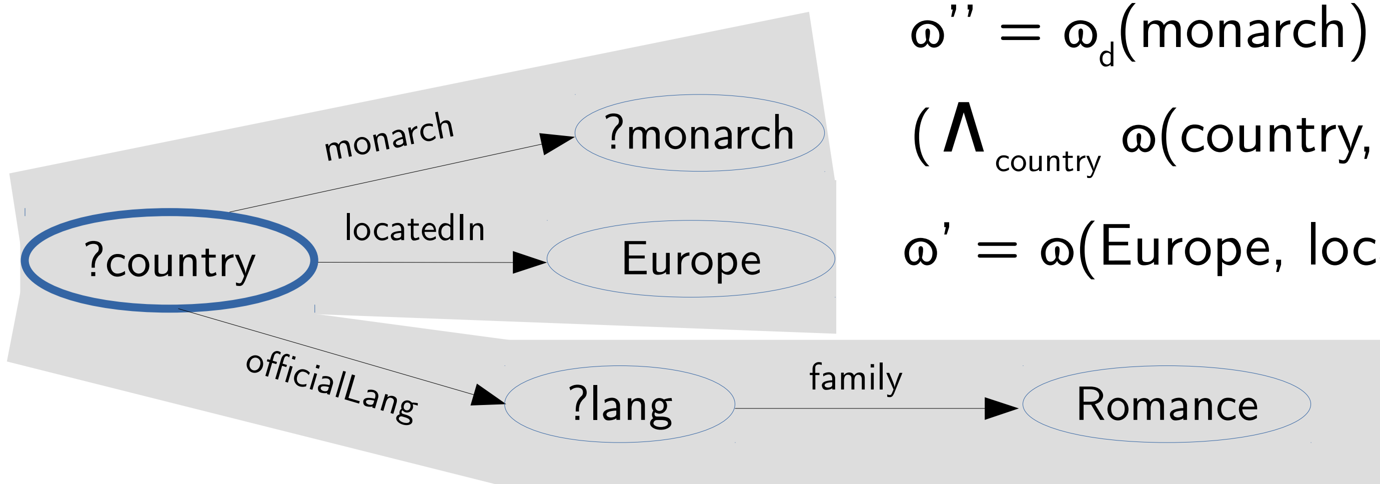
$\varpi' = \varpi(\text{Europe, locatedIn}^{-1})$

$\varpi^{**} = \bigwedge_{\text{l : family(l, Romance)}} \varpi(\text{l, officialLang}^{-1}))$

$\varpi^{*} = \varpi(\text{Romance, family}^{-1})$

```
SELECT ?country WHERE {
    ?country monarch ?monarch .
    ?country locatedIn Europe .
    ?country officialLang ?lang .
    ?lang family Romance .
}
```

Automatic oracle composition



$\varpi'' = \varpi_d(\text{monarch}) \wedge$

$(\bigwedge_{\text{country}} \varpi(\text{country, monarch}))$

$\varpi' = \varpi(\text{Europe, locatedIn}^{-1})$

$\varpi^{**} = \bigwedge_{l\,:\,\text{family}(l,\ \text{Romance})} \varpi(l, \text{officialLang}^{-1}))$

$\varpi^{*} = \varpi(\text{Romance, family}^{-1})$

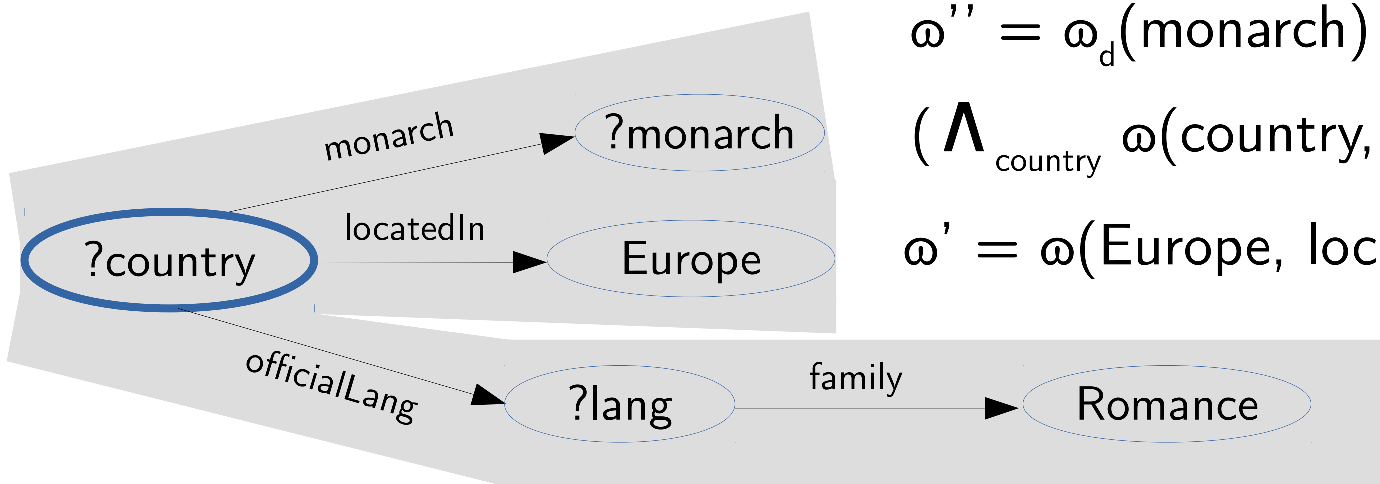
```
SELECT ?country WHERE {
    ?country monarch ?monarch .
    ?country locatedIn Europe .
    ?country officialLang ?lang .
    ?lang family Romance .
}
```

Completeness veredict $= \varpi^{**} \wedge \varpi^{*} \wedge \varpi' \wedge \varpi''$

Automatic oracle composition

$\varpi'' = \varpi_d(\text{monarch}) \wedge$

$(\bigwedge_{\text{country}} \varpi(\text{country, monarch}))$



$\varpi' = \varpi(\text{Europe, locatedIn}^{-1})$

$\varpi^{**} = \bigwedge_{\text{l : family(l, Romance)}} \varpi(\text{l, officialLang}^{-1}))$

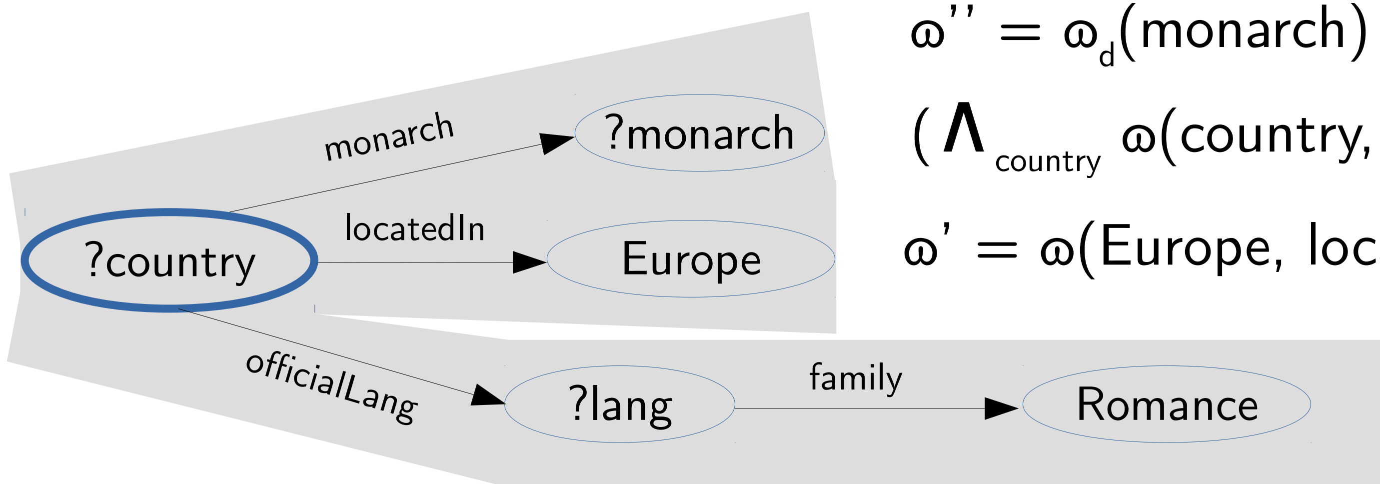
$\varpi^{*} = \varpi(\text{Romance, family}^{-1})$

```
SELECT ?country WHERE {
    ?country monarch ?monarch .
    ?country locatedIn Europe .
    ?country officialLang ?lang .
    ?lang family Romance .
}
```

Completeness veredict $= \varpi^{**} \wedge \varpi^{*} \wedge \varpi' \wedge \varpi''$

Confidence $= \text{prec}(\varpi^{**}) \times \text{prec}(\varpi^{*}) \times \text{prec}(\varpi') \times \text{prec}(\varpi'')$

Automatic oracle composition



$\varpi'' = \varpi_d(\text{monarch}) \wedge (\bigwedge_{\text{country}} \varpi(\text{country, monarch}))$

$\varpi' = \varpi(\text{Europe, locatedIn}^{-1})$

$\varpi^{**} = \bigwedge_{l\,:\,\text{family}(l,\ \text{Romance})} \varpi(l, \text{officialLang}^{-1}))$

$\varpi^{*} = \varpi(\text{Romance, family}^{-1})$

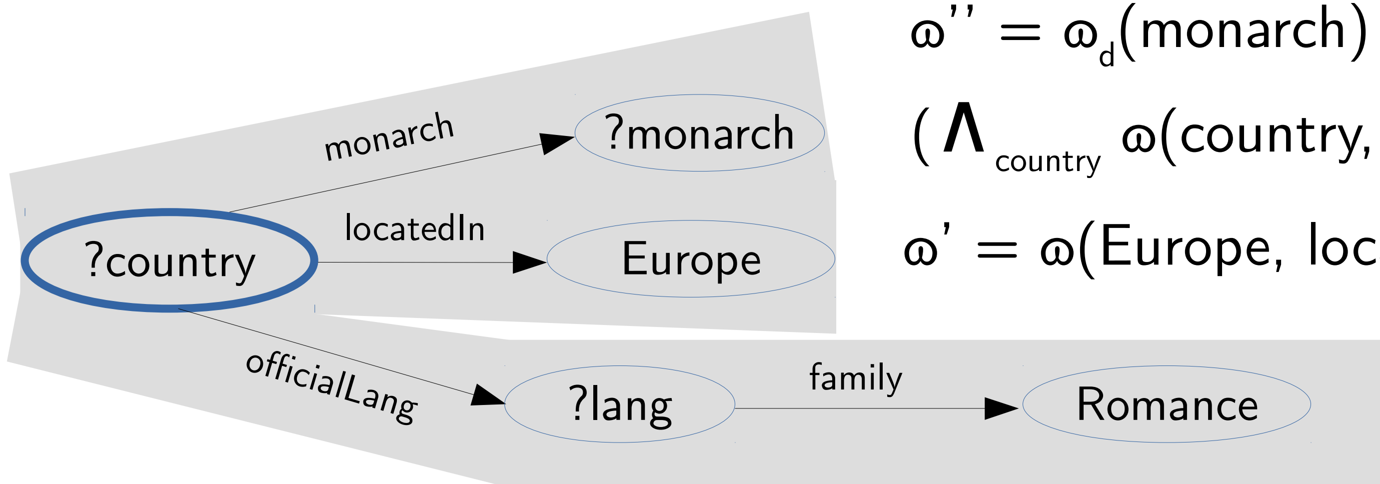
```
SELECT ?country WHERE {
    ?country monarch ?monarch .
    ?country locatedIn Europe .
    ?country officialLang ?lan
    ?lang family Romance .
}
```

It could easily lead to false negatives

Completeness veredict = $\varpi^{**} \wedge \varpi^{*} \wedge \varpi' \wedge \varpi''$

Confidence = prec(ϖ^{**}) × prec(ϖ^{*}) × prec(ϖ') × prec(ϖ'')

Automatic oracle composition



$\varpi'' = \varpi_d(\text{monarch}) \wedge (\bigwedge_{\text{country}} \varpi(\text{country, monarch}))$

$\varpi' = \varpi(\text{Europe, locatedIn}^{-1})$

$\varpi^{**} = \bigwedge_{l\,:\,\text{family}(l,\ \text{Romance})} \varpi(l, \text{officialLang}^{-1}))$

$\varpi^{*} = \varpi(\text{Romance, family}^{-1})$

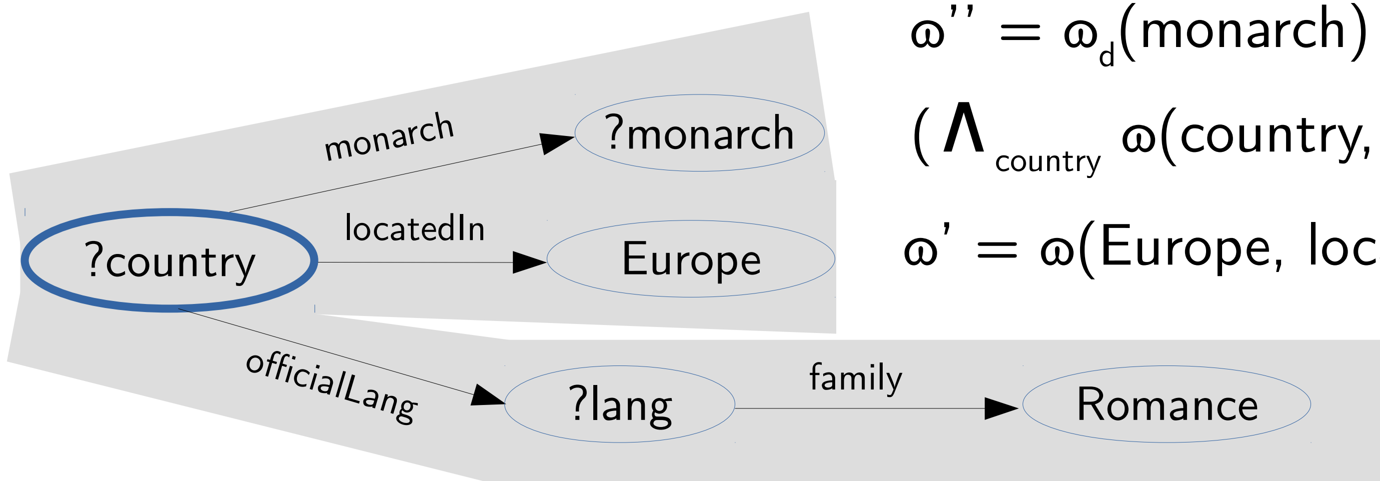
```
SELECT ?country WHERE {
    ?country monarch ?monarch
    ?country locatedIn Europ
    ?country officialLang ?lan
    ?lang family Romance .
}
```

We would like to minimize the number of used oracles

Completeness veredict $= \varpi^{**} \wedge \varpi^{*} \wedge \varpi' \wedge \varpi''$

Confidence $= \text{prec}(\varpi^{**}) \times \text{prec}(\varpi^{*}) \times \text{prec}(\varpi') \times \text{prec}(\varpi'')$

Automatic oracle composition



$\varpi'' = \varpi_d(\text{monarch}) \wedge (\bigwedge_{\text{country}} \varpi(\text{country, monarch}))$

$\varpi' = \varpi(\text{Europe, locatedIn}^{-1})$

$\varpi^{**} = \bigwedge_{l\,:\,\text{family}(l,\ \text{Romance})} \varpi(l, \text{officialLang}^{-1}))$

$\varpi^{*} = \varpi(\text{Romance, family}^{-1})$

SELECT ?country WHERE {
untry monarch ?monarch
untry locatedIn Europ
?country officialLang ?lan
?lang family Romance .
}

Use more complex oracles that cover larger parts of the query graph at once

We would like to minimize the number of used oracles

Completeness veredict $= \varpi^{**} \wedge \varpi^{*} \wedge \varpi' \wedge \varpi''$

Confidence $= \text{prec}(\varpi^{**}) \times \text{prec}(\varpi^{*}) \times \text{prec}(\varpi') \times \text{prec}(\varpi'')$

Outline

- Completeness in RDF knowledge bases
- Completeness oracles
- Our vision
  - Representations for completeness oracles
  - Reasoning with completeness oracles
  - Enabling completeness in SPARQL
- Summary & conclusions

Enabling completeness in SPARQL

- Calls to completeness oracles could be embedded in the query language

Enabling completeness in SPARQL

- Calls to completeness oracles could be embedded in the query language
  - Example: aggregated number of Spanish speakers in a county per state, only for *those states with complete information*

Enabling completeness in SPARQL

- Calls to completeness oracles could be embedded in the query language
  - Example: aggregated number of Spanish speakers in a county per state, only for *those states with complete information*

```
SELECT ?state sum(?nspeak) WHERE {
    ?county inState ?state .
    ?county spanishSpeakers ?nspeak .
} GROUP BY ?state HAVING (complete(?nspeak))
```

Enabling completeness in SPARQL

- Calls to completeness oracles could be embedded in the query language
  - Example: aggregated number of Spanish speakers in a county per state, only for *those states with complete information*

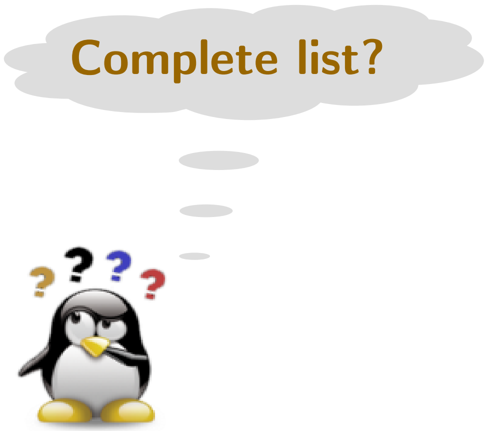
Boolean aggregation function on sets of bindings

```
SELECT ?state sum(?nspeak) WHERE {
    ?county inState ?state .
    ?county spanishSpeakers ?nspeak .
} GROUP BY ?state HAVING (complete(?nspeak))
```

Enabling completeness in SPARQL

- For each value of *?state* check if the bindings for *?nspeak* are complete

?state	?county	?nspeak
Delaware	New Castle Kent Sussex	2000 4300 1200
Hawaii	Hawaii Kalawao	30000 1200



SELECT ?state sum(?nspeak) WHERE {
  ?county inState ?state .
  ?county spanishSpeakers ?nspeak .
} GROUP BY ?state HAVING (**complete**(?nspeak))

Enabling completeness in SPARQL

- For each value of *?state* check if the bindings for *?nspeak* are complete

?state	?county	?nspeak
Delaware	New Castle	2000
	Kent	4300
	Sussex	1200
Hawaii	Hawaii	30000
	Kalawao	1200

SELECT **complete**(?nspeak) WHERE {
  ?county inState Delaware .
  ?county spanishSpeakers ?nspeak .
}

SELECT ?state sum(?nspeak) WHERE {
  ?county inState ?state .
  ?county spanishSpeakers ?nspeak .
} GROUP BY ?state HAVING (**complete**(?nspeak))

Enabling completeness in SPARQL

- For each value of *?state* check if … for *?nspeak* are complete

Completeness oracles to the rescue!

?state	?county	?nspeak
Delaware	New Castle	2000
	Kent	4300
	Sussex	1200
Hawaii	Hawaii	30000
	Kalawao	1200

```
SELECT complete(?nspeak) WHERE {
    ?county inState Delaware .
    ?county spanishSpeakers ?nspeak .
}
```

```
SELECT ?state sum(?nspeak) WHERE {
    ?county inState ?state .
    ?county spanishSpeakers ?nspeak .
} GROUP BY ?state HAVING (complete(?nspeak))
```

Outline

- Completeness in RDF knowledge bases
- Completeness oracles
- Our vision
  - Representations for completeness oracles
  - Reasoning with completeness oracles
  - Enabling completeness in SPARQL
- Summary & conclusions

Summary

- Completeness is a dimension of data quality
  - It determines the value and reliability of the data
  - Existing work provides only completeness statements and oracles for simple queries
- Semantic Web is not completeness-aware
  - **Vision**
    - Use completeness oracles for simpler queries to infer completeness for arbitrary queries
    - Embed completeness in the SPARQL query language
  - **Goal:** Increase the value of the results delivered by queries

Future work

- Augment existing RDF data with completeness statements and oracles
- Implement reasoning with completeness oracles in SPARQL query engines
  - Extend the SPARQL query language to support the *complete* aggregation function