Previously on Introduction to Linked Data...

- In chapter 5 we have seen how to
  - translate a SPARQL query into an algebra expression; and
  - evaluate the algebra expression on a dataset; in particular,
  - check, given a solution mapping μ, whether μ is a solution for basic graph pattern matching

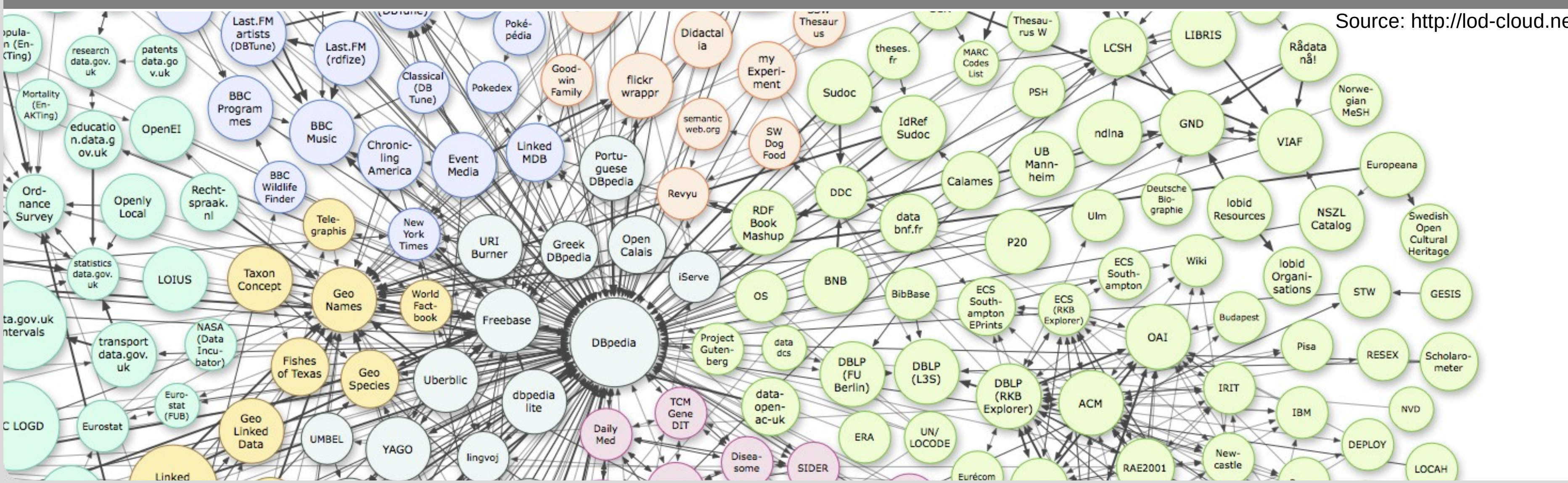
- In chapters 8, 9 and 10, we have seen how to
  - model data using RDF and RDFS terms;
  - define simple, D, RDF, RDFS and OWL LD entailment; and
  - check, given two graphs G1 and G2, whether

    G1 simply entails G2,
    G1 D-entails G2,
    G1 RDF-entails G2,
    G1 RDFS-entails G2,
    G1 OWL LD-entails G2.

C11 Combining Query Processing with Entailment
How to combine deductive reasoning with query processing?

Version 2022-07-11
Lecturer: Andreas Harth

CHAIR OF TECHNICAL INFORMATION SYSTEMS



Source: http://lod-cloud.net

CC - Creative Commons Licensing

- This set of slides is part of the lecture „Semantic Web Technologies“ held at Karlsruhe Institute of Technology
- The content of the lecture was prepared by PD Dr. Andreas Harth based on his book „Introduction to Linked Data“
- The slides were prepared by Andreas Harth and Maribel Acosta, with the help of a slideset by Birte Glimm.

Agenda

1. **Basic Graph Pattern Matching Revisited**
2. Entailment Revisited
3. Basic Graph Pattern Matching and Entailment Regimes
4. Open-World Assumption vs. Closed-World Assumption
5. Options for Implementation

Basic Graph Pattern Matching

Definition 14 (Basic Graph Pattern Matching). *Let P be a basic graph pattern and G be an RDF graph. A partial function μ is a solution of matching P on the graph G if:*

- *the domain of μ is the set of variables in P, and*
- *there exists a mapping σ of blank nodes in P to RDF terms $(\mathcal{U} \cup \mathcal{B} \cup \mathcal{L})$ in G, so that*
- *the graph $\mu(\sigma(P))$ is a subgraph of G.*

We write $\mu(P)$ to denote an RDF graph obtained from BGP P by replacing each variable x in P with $\mu(x)$.

[5] The mapping σ is similar to μ (but for blank nodes instead of variables); σ ensures the correct handling of blank nodes.

Think-Pair-Share

Given the following graph G available at `g.ttl`:

```
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix : <#> .
:Magneto a foaf:Person ;
         foaf:name "Max Eisenhardt" .
foaf:Person rdfs:subClassOf foaf:Agent .
```

What is the solution of matching the following basic graph pattern P?

```
?x a foaf:Agent . ?x foaf:name ?name .
```

Think-Pair-Share

Given the following graph G available at `g.ttl`:

```
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix : <#> .
:Magneto a foaf:Person ;
         foaf:name "Max Eisenhardt" .
foaf:Person rdfs:subClassOf foaf:Agent .
```

What is the solution of matching the following basic graph pattern P?

```
?x a foaf:Agent . ?x foaf:name ?name .
```

Solution applying **basic graph pattern matching**:

```
eval(G, BGP(P)) =
```

Wanted solution:

eval(G, BGP(P)) = { μ_1(?x) = :Magneto, μ_1(?name) = "Max Eisenhardt" }

Goal: Apply entailment during query evaluation

Agenda

1. Basic Graph Pattern Matching Revisited
2. **Entailment Revisited**
3. Basic Graph Pattern Matching and Entailment Regimes
4. Open-World Assumption vs. Closed-World Assumption
5. Options for Implementation

An Attempt to Update Basic Graph Pattern Matching

Definition 14 (Basic Graph Pattern Matching). *Let P be a basic graph pattern and G be an RDF graph. A partial function μ is a solution of matching P on the graph G if:*

- *the domain of μ is the set of variables in P, and*
- *there exists a mapping σ of blank nodes in P to RDF terms $(\mathcal{U} \cup \mathcal{B} \cup \mathcal{L})$ in G, so that*

$$G \models_{RDFS} \mu(\sigma(P))$$

- *the graph ~~$\mu(\sigma(P))$ is a subgraph of G.~~*

We write $\mu(P)$ to denote an RDF graph obtained from BGP P by replacing each variable x in P with $\mu(x)$.

Theory vs. Practice

- In theory, queries with entailment may have infinitely many solutions.
- In practice, most solutions in such cases are trivial.

https://twitter.com/disneypixar/status/675070238958972928

Simple Entailment – Blank Nodes

- Consider graph G:

```
_:a <#p> _:b .
```

- Consider graph E:

```
_:b <#p> _:a .
```

- G $\models_{simple}$ E

D-Entailment – Literals

- Consider graph G:

```
<#s> <#p> 23 .
```

- From G, we can *D*-entail the following triples, given that *D* includes xsd:integer and xsd:decimal:

```
<#s> <#p> 23.0 .
<#s> <#p> 23.00 .
<#s> <#p> 23.000 .
<#s> <#p> 23.0000 .
<#s> <#p> 23.00000 .
<#s> <#p> 23.000000 .
<#s> <#p> 23.0000000 .
<#s> <#p> 23.00000000 .
...
```

RDF Entailment – Container Membership Properties

- Consider the empty graph G.

- From G we can RDF-entail (via axiomatic triples):

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
rdf:_1 rdf:type rdf:Property .
rdf:_2 rdf:type rdf:Property .
rdf:_3 rdf:type rdf:Property .
rdf:_4 rdf:type rdf:Property .
rdf:_5 rdf:type rdf:Property .
rdf:_6 rdf:type rdf:Property .
rdf:_7 rdf:type rdf:Property .
rdf:_8 rdf:type rdf:Property .
# and so on for rdf:_9, rdf:_10…
```

RDFS Entailment – Container Membership Properties

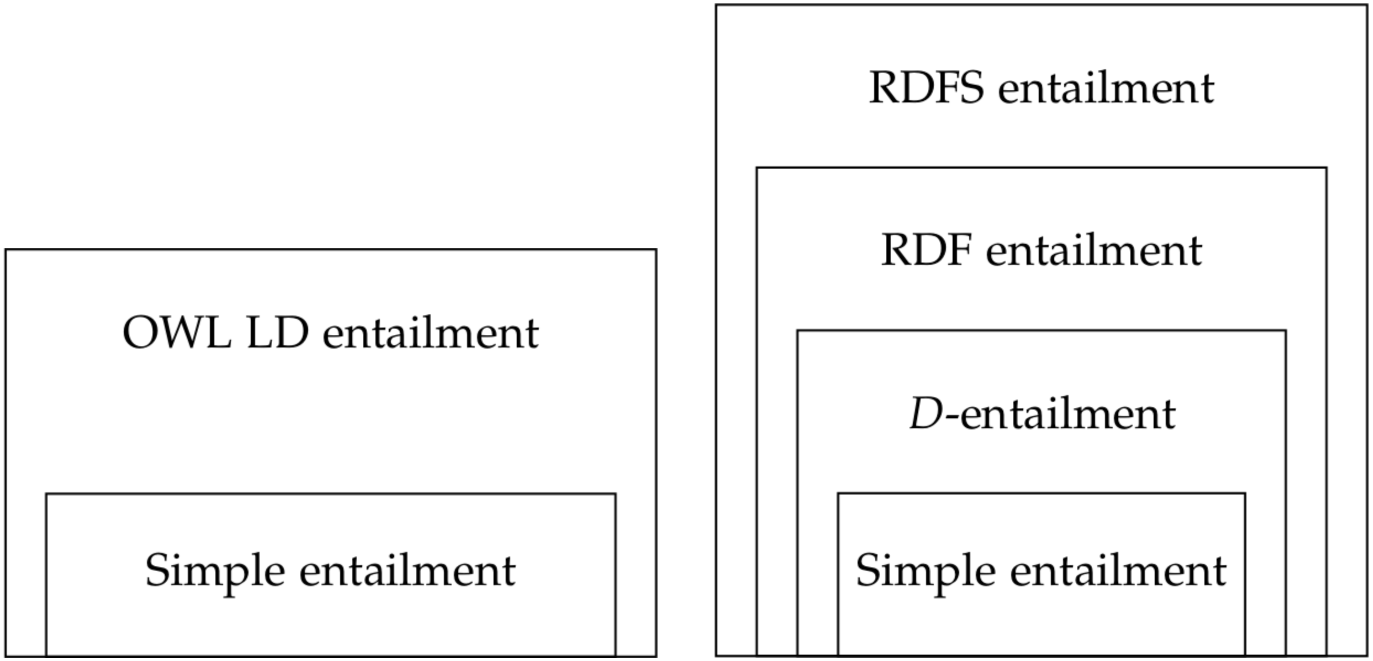
- Consider the empty graph G.
- From G we can RDFS-entail (via axiomatic triples):

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .

rdf:_1 a rdfs:ContainerMembershipProperty ; rdfs:domain rdfs:Resource ; rdfs:range rdfs:Resource .rdf:_1 rdf:type rdf:Property .
rdf:_2 a rdfs:ContainerMembershipProperty ; rdfs:domain rdfs:Resource ; rdfs:range rdfs:Resource .rdf:_1 rdf:type rdf:Property .
# and so on for rdf:_3, rdf:_4…
```

OWL LD Entailment – Literals

- Infinitely many lexical forms similar to D-entailment
- infinitely many axiomatic triples involving literals (e.g., owl:sameAs and owl:differentFrom axiomatic triples between the literals via entailment patterns dt-eq and dt-diff)



Layering of entailment.

Think-Pair-Share

Given the following graph G available at `g.ttl`:

```
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix : <#> .
:Magneto a foaf:Person ;
         foaf:name "Max Eisenhardt" .
foaf:Person rdfs:subClassOf foaf:Agent .
```

Based on the updated definition taking into account RDFS entailment, what is the solution of matching the following basic graph pattern P?

```
?x a rdf:Property .
```

Agenda

1. Basic Graph Pattern Matching Revisited
2. Entailment Revisited
3. **Basic Graph Pattern Matching and Entailment Regimes**
4. Open-World Assumption vs. Closed-World Assumption
5. Options for Implementation

Extending SPARQL with Entailment Regimes

- The goal is to define (and implement) the changes to SPARQL query evaluation that are necessary to include entailment relations.
- In order to extend SPARQL for an entailment relation, it suffices to **modify the evaluation of BGPs** accordingly.
- The remaining **algebra operators** (JOIN, UNION, OPTIONAL, etc.) can still be evaluated following the **SPARQL semantics**.
- We can apply entailment regimes based on the RDF semantics:
  - For example: simple, *D*-, RDF, RDFS, OWL LD
  - In the lecture, we do not consider more expressive OWL/OWL2 profiles

Evaluating Basic Graph Patterns

- In the variant of SPARQL we have seen, the BGP operator is the only one that accesses triples in a graph

$$\text{eval}(D(G), BGP(P)) = \Omega$$

- Now, we extend eval() with a parameter E to indicate the desired entailment regime

$$\text{eval-}E(D(G), BGP(P)) = \Omega$$

Entailment Regimes

- An entailment regime specifies:
  - A subset of RDF graphs called **well-formed** for the regime
  - An **entailment relation**

- We consider the following entailment relations, all built on the RDF-based semantics (not the OWL2 direct semantics):
  - $\models_{\text{simple}}$
  - $\models_{\text{D}}$
  - $\models_{\text{RDF}}$
  - $\models_{\text{RDFS}}$
  - $\models_{\text{OWL LD}}$

Well-formed Graphs

- Conditions for well-formed graphs could include:
  - RDF lists have to be properly closed
  - All the required triples are defined in the RDF graph
- We focus on the RDF-based semantics (RDF, RDFS, OWL LD), where all **RDF graphs are well-formed** (syntax restrictions for RDF apply)
- The graph over which we operate has to be **satisfiable** (i.e., does not lead to unsatisfiability)
- Only solution mappings that yield a set of well-formed triples when applied to CONSTRUCT templates T are legal solution mappings
- That is, only solution mappings applied to T (i.e., $\mu(\sigma(T))$) that yield well-formed RDF triples:
  - no literals in subject position, no blank nodes or literals on predicate position are legal.

Scoping Graph

- In SPARQL query evaluation, solution mappings can **bind a variable** in a BGP **to a blank node** in an RDF graph G.

- Blank nodes in query solutions **differ** from blank nodes in the queried graphs

- Suppose Q is a query, B is a BGP in Q, and G is the queried graph, solutions to Q are given by reference to GQ (the scoping graph), rather than G itself

- Given an entailment regime E, the scoping graph GQ of a graph G:
  - Is E-equivalent to G
  - Does not share blank nodes with G

Handling Blank Nodes: Skolemisation

- We have to contain the free reordering of blank nodes that $\models_{\text{simple}}$ allows (another source of infinity).
- We will use Skolemisation that ensures that blank nodes are treated like constants.
- Skolemisation replaces blank nodes with constants (URIs).

Definition 22 (Skolemisation). *Let* `skol` *be a prefix URI that is not used in the scoping graph or a query. The Skolemisation* $\mathtt{sk}(\_\mathtt{:}\mathit{id})$ *of a blank node* `_:id` *is defined as* $\mathtt{sk}(\_\mathtt{:}\mathit{id}) = \mathit{skol}\mathtt{:}\mathit{id}$. *We also allow to apply* $\mathtt{sk}()$ *to graphs.*

Handling Literals: Canonicalisation

- XSD 1.1 defines canonical representation for data values.

- For example, in the decimal datatype from the XML Schema Datatypes all of the following lexical forms represent the same value:
  - 100.5, +100.5, 0100.5, 100.50, 100.500, 100.5000
- For the above data values, the canonical lexical form is: 100.5.

- For the values
  - 100, +100, 0100, 100.0, 100.00, 100.000
- the canonical lexical form is: 100 according to XSD 1.1.

- The entailment regimes require using canonical values, and thus prevent infinite solutions for literals.

Preventing Infinite Solution Sequences: Container Membership Properties

- Due to the container membership properties rdf:_1, rdf:_2 …, the set of axiomatic triples for RDF and RDFS entailment is infinite
- Let G be a graph; Voc(G) denotes all URIs and literals in the graph
- We write Voc(RDF) to denote the RDF vocabulary, and Voc(RDFS) to denote the RDFS vocabulary
- Voc^{-}(RDF) denotes the RDF vocabulary without container membership properties
- Voc^{-}(RDFS) denotes the RDFS vocabulary without container membership properties

Extension of Basic Graph Pattern Matching with RDFS Entailment

- Skolemisation of blank nodes
- Restriction of vocabulary
- Entailment instead of subgraph check

Definition 23 (Basic Graph Pattern Matching under RDFS entailment). *Let P be a basic graph pattern and G be an RDF graph. A partial function μ is a solution of evaluating the expression $BGP(P)$ on the graph G under RDFS entailment if:*

- *the domain of μ is the set of variables in P, and*
- *RDF terms of μ appear in G or $\mathsf{Voc}^{-}(RDFS)$, and*
- *there exists a mapping σ of blank nodes in P to RDF terms in G, so that*
- *the graph $\mathsf{sk}(\mu(\sigma(P)))$ is well-formed and $\mathsf{sk}(G) \models_{RDFS} \mathsf{sk}(\mu(\sigma(P)))$.*

- BGP matching for RDF and OWL LD entailment is analogously defined

Think-Pair-Share

- Given the following RDF graph *G* available at `http://example.org/persons` and the SPARQL expression *E*

```
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix dbo: <http://dbpedia.org/ontology/> .
@prefix : <#> .
:Magneto a foaf:Person ;
         foaf:name "Max Eisenhardt" .
_:x a dbo:Person .
foaf:Person rdfs:subClassOf foaf:Agent .
foaf:Person owl:equivalentClass dbo:Person .
```

- What is the solution sequence of the following?

eval-RDFS(G, BGP (`?p a foaf:Agent .`))

Entailment on RDF Datasets

- We have defined entailment on graphs
- RDF datasets consist of multiple graphs (a single default graph and zero or many named graphs)

- The graphs in an RDF dataset are considered separately for entailment

Example (1)

- Consider document a.ttl:

```
@prefix ex: <http://example.org/bsp#> .
ex:p rdfs:domain ex:A .
```

- Consider document b.ttl:

```
@prefix ex: <http://example.org/bsp#> .
ex:s ex:p ex:o .
```

- The following query does not yield any solutions:

```
PREFIX : <http://example.org/bsp#>
SELECT ?g
FROM NAMED <a.ttl>
FROM NAMED <b.ttl>
WHERE {
  GRAPH ?g { ?x rdf:type :A . }
}
```

Example (2)

- Consider document a.ttl:

```
@prefix ex: <http://example.org/bsp#> .
ex:p rdfs:domain ex:A .
```

- Consider document b.ttl:

```
@prefix ex: <http://example.org/bsp#> .
ex:s ex:p ex:o .
```

- The following query does yield a solution:

```
PREFIX : <http://example.org/bsp#>
SELECT ?x
FROM <a.ttl>
FROM <b.ttl>
WHERE {
  ?x rdf:type :A .
}
```

Infinitely Many Infinities

- We have solved the issue of infinitely many solutions, via
  - Skolemisation and
  - restricting the vocabulary

- But… RDF entailment also allows infinitely many interpretations (i.e., interpretations that are models of a graph do not have to be minimal)

Agenda

1. Basic Graph Pattern Matching Revisited
2. Entailment Revisited
3. Basic Graph Pattern Matching and Entailment Regimes
4. **Open-World Assumption vs. Closed-World Assumption**
5. Options for Implementation

Queries with Negation

- Can we query for the absence of data?
- E.g., return all unmarried people

- E.g., `people.ttl`:

```
@prefix : <#> .
:Peter   :husbandOf  :Maria .
:Peter   :hasChild   :Tom .
:Peter   :friendOf   :Tim .

:Maria   :wifeOf     :Peter .
:Maria   :hasChild   :Tom .
:Maria   :friendOf   :Anna .
```

Closed-World Assumption

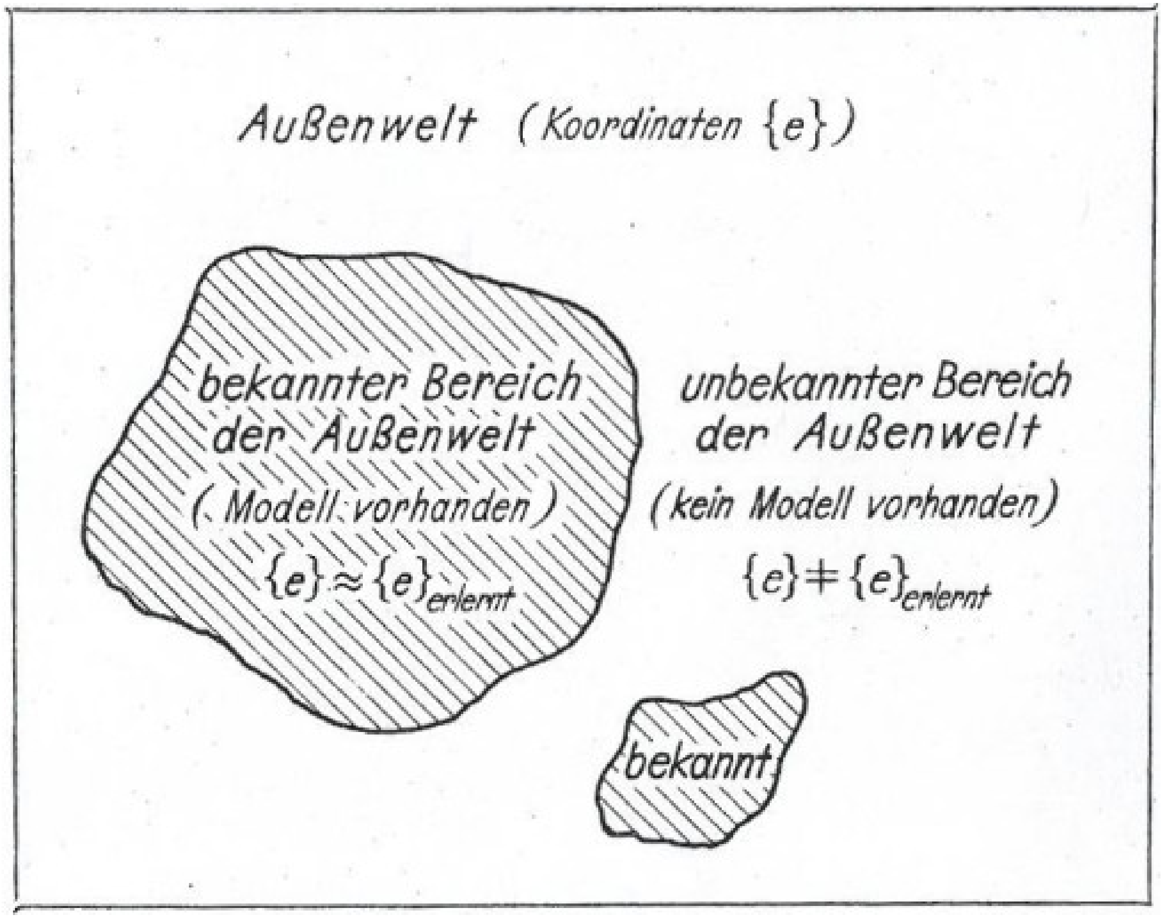
- To use negation in queries, we need to specify over which data we want to operate

- Query evaluation would roughly go as follows:

- Return the people from people.ttl
- Return all married people from people.ttl
- Unmarried people are all people minus the married people

- Philosophical: when we query the web, can we ever be sure we have all the data from the web?

Known World vs. Outer World



Karl Steinbuch: Automat und Mensch; über menschliche und maschinelle Intelligenz. Berlin, Springer, 1961.

Open-World Assumption

- We can never be sure to have all data from the web
- So, we cannot use negation
- That means that if we design a ontology language (RDFS, OWL…), we must use only positive rules (no negation!)

```
@prefix : <#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
:Peter     :husbandOf   :Maria .
:Bob       :husbandOf   :Maria .
:husbandOf a owl:InverseFunctionalProperty .
```

- What should happen?
- CWA: BOOM!
- OWA: it could still be that we find out that

```
:Peter     owl:sameAs :Bob .
```

Completeness and the Open-World Assumption

- Given that the web is huge, it is unreasonable to assume we can access all of the web
- Thus, the RDF dataset available locally is always a subset of the entire web
- We do processing or ask queries on the local RDF dataset
- The results of these queries are never complete, but always a subset of the complete results
- Thus, we can assume that the results of a query are correct, but we cannot assume that we get all results with respect to the whole web
  - Note: but the results are complete w.r.t. the local dataset
- We have to assume an „**open world**" when operating on data on the web:
  - We cannot assume that the data that is not recorded does not exist

Logical Monotonicity

- Introducing new data cannot invalidate existing data.
- So, if I have a result for entailment, I know that the result will be valid, no matter how much additional data I see.
- Logical monotonicity is guaranteed when considering the open-world assumption.

- E.g. from:

```
@prefix ex: <http://example.org/bsp#> .
ex:p rdfs:domain ex:A .
ex:s ex:p ex:o .
```

- follows under RDFS semantics:

```
@prefix ex: <http://example.org/bsp#> .
ex:s rdf:type ex:A .
```

- no matter what additional triples the RDF processor might see.

Monotonicity and SPARQL

- Certain SPARQL features introduce non-monotonicity.
- In the variant of SPARQL we have seen, OPTIONAL is the only non-monotonic feature.

Example for Non-Monotonicity

- Consider the query:

```
SELECT ?x ?y
WHERE {
   ?s :p ?x .
   OPTIONAL { ?s :q ?y . }
}
```

- Consider the following graph:

```
:s :p :o .
```

- The solution would be:

$\mu_1(?x) = :o$

- Now, if we learn another triple:

```
:s :q :o .
```

- The solution instead would be:

$\mu_1(?x) = :o, \mu_1(?y) = :o$

We would need to

retract the first μ_1 and

add the second μ_1.

That is, the results **do not** grow monotonically!

Entailment vs. SPARQL

- The definitions around entailment follow the open-world assumption, hence ensures logical monotonicity.

- Only a subset of SPARQL operators are monotonic, so features such as OPTIONAL (which is defined based on set difference) require the closed-world assumption (that models are minimal).

$$
\begin{array}{ll}
Diff(\Omega_1, \Omega_2, F) & \{\mu | \mu \in \Omega_1 \text{ such that for all } \mu' \in \Omega_2, \text{ either } \mu \text{ and } \mu' \text{ are not compatible} \\
& \text{or } \mu \text{ and } \mu' \text{ are compatible and } F(merge(\mu, \mu')) \text{ is } \texttt{false} \} \\
LeftJoin(\Omega_1, \Omega_2, F) & Filter(F, Join(\Omega_1, \Omega_2)) \cup Diff(\Omega_1, \Omega_2, F)
\end{array}
$$

- Current research: how to bring together the open-world assumption of entailment and the closed-world assumption of SPARQL?
  - Simple solution: disallow OPTIONAL in SPARQL ㅆ

Think-Pair-Share

- The unique name assumption is a simplifying assumption made in some logical languages. In logics with the unique name assumption, different names (URIs, literals) always refer to different entities in the world (resources).
- Is the unique name assumption compatible with the open-world assumption?
- With what entailment relations does the question become relevant?

Agenda

1. Basic Graph Pattern Matching Revisited
2. Entailment Revisited
3. Basic Graph Pattern Matching and Entailment Regimes
4. Open-World Assumption vs. Closed-World Assumption
5. **Options for Implementation**

Implementing Entailment

- The way how query solutions are defined allows for different implementation techniques:

- Materialisation/forward-chaining
- Query rewriting/backward-chaining
- Hybrid approaches

Materialisation

- Works from the available triples and generates all conclusions
- Then, the query is evaluated over the extended graph

- Drawbacks:
  - The queried graph increases in size
  - Every change on the graph requires a recomputation

Query Rewriting

- Instead of extending the graph, we extend the query
- Then, the extended query is evaluated on the graph

- Drawbacks:
  - Difficult to find all solutions (especially due to recursive application of rules), query might grow very large
  - Query rewriting has to be done per query, query processing takes more time

Hybrid Approaches

- Combining materialisation and query rewriting.

- E.g., do not materialise the instances of rdfs:Resource.
- E.g., do not materialise the semantics of owl:sameAs.

- Topic of current research.

Learning Goals

- G 11.1 Outline the necessary changes to the evaluation of BGP expressions to support RDF, RDFS and OWL LD entailment.
- G 11.2 Find solutions for SPARQL queries under simple, *D*-, RDF, RDFS or OWL LD entailment.
- G 11.3 Given multiple graphs, provide mappings between the terms of the graphs using RDFS and OWL LD, to allow for integrated querying.
- G 11.4 Explain the differences between the open-world assumption and the closed-world assumption.