Universität Bremen

Institute for Artificial Intelligence

Faculty 03

Mathematics / Computer science

SPARQL cheat sheet

Michaela Kümpel

Bremen, 11. Juni 2024

Artificial Intelligence

Starting a statement

- SELECT ?var WHERE { }
- SELCT * WHERE { }
- SELECT DISTINCT ... WHERE { }
- SELECT ... FROM <> WHERE { }
- SELECT (10 * ?a AS ?mult) WHERE { }

Ending a statement

- { } GROUP BY ... (e.g. group by article number)
- { } HAVING ... COUNT, SUM, AVG, MIN, MAX, SAMPLE, GROUP_CONCAT
- { } ORDER BY [ASC] (?variable)
- { } LIMIT ...
- { } OFFSET ...
- { } BINDINGS ...

... and in between

- specific data property:
  {?person foaf:name “Michael Beetz” .}
- all entities with a property:
  {?product lbl:hasLabel lbl:DerGruenePunkt.}

Graph Patterns

Conjunction
Join together the results of solving A and B by matching the values of any variables in common

A . B

{?person a foaf:person}
.
{?person foaf:depiction ?image}
→ all things that are a person and have a picture

Graph Patterns

Left join
Join together the results of solving A and B by matching the values of any variables in common. Allows for B being empty

A OPTIONAL { B }

```
?person a foaf:person.
OPTIONAL
{?person foaf:depiction ?image}
```

→ all things that are a person (not necessarily have a picture)

Graph Patterns

Disjunction
Include both the results of solving A and the results of solving B.

{ A } UNION { B }

```
{ ?person a foaf:person}
UNION
{?person foaf:depiction ?image}
```

→ all things that are a person and all things that have an image

Graph Patterns

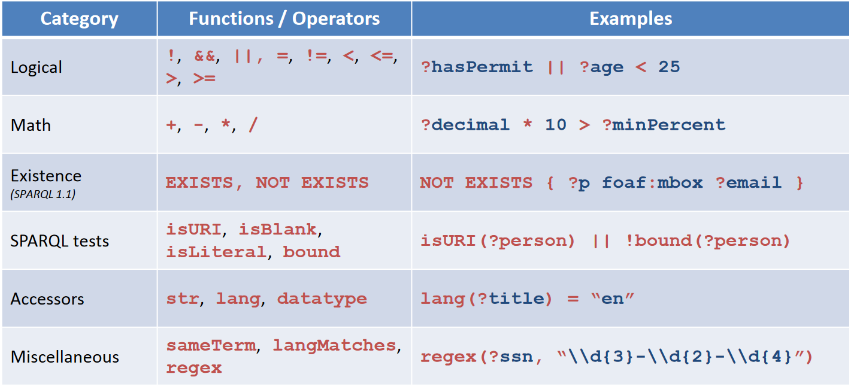
Negation

Solve A. Solve B. Include only those results from solving A that are not compatible with any of the results from B.

A MINUS { B }

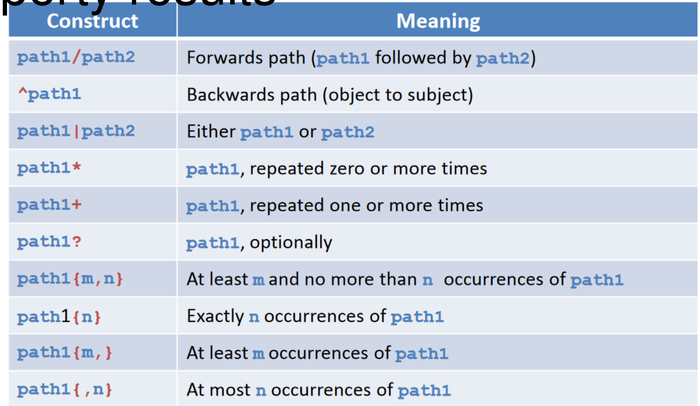
?person a foaf:person.
MINUS
{?person foaf:depiction ?image}
→ all things that are a person and NOT have an image

Filter



Category	Functions / Operators	Examples
Logical	!, &&, ||, =, !=, <, <=, >, >=	?hasPermit || ?age < 25
Math	+, -, *, /	?decimal * 10 > ?minPercent
Existence *(SPARQL 1.1)*	EXISTS, NOT EXISTS	NOT EXISTS { ?p foaf:mbox ?email }
SPARQL tests	isURI, isBlank, isLiteral, bound	isURI(?person) || !bound(?person)
Accessors	str, lang, datatype	lang(?title) = “en”
Miscellaneous	sameTerm, langMatches, regex	regex(?ssn, “\\d{3}-\\d{2}-\\d{4}”)

Filter property results



Construct	Meaning
path1/path2	Forwards path (path1 followed by path2)
^path1	Backwards path (object to subject)
path1|path2	Either path1 or path2
path1*	path1, repeated zero or more times
path1+	path1, repeated one or more times
path1?	path1, optionally
path1{m,n}	At least m and no more than n occurrences of path1
path1{n}	Exactly n occurrences of path1
path1{m,}	At least m occurrences of path1
path1{,n}	At most n occurrences of path1

Class Restriction <-> RDF Properties

The OWL expression is what you see when you model restrictions in Protègè
The RDF properties are how it is stored (when you open it in an editor) and queried

OWL expression	RDF property
some *anyClass*	owl:someValuesFrom
only *anyClass*	owl:allValuesFrom
value *anyDataValue*	owl:hasValue
anyProperty	owl:onProperty

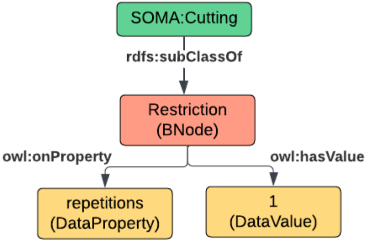
Cardinality Restriction <-> RDF Properties

Again, axiom expressions as in Protègè on the left and its translation to RDF on the right

OWL expression	RDF property
min x	owl:minQualifiedCardinality
exactly x	owl:qualifiedCardinality
max x	owl:maxQualifiedCardinality
anyProperty	owl:onProperty
anyClass	owl:onClass
anyDataType	owl:onDataRange

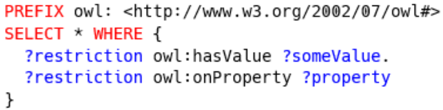
RDF Parsed Graph - Restriction

- Whenever an ontology is parsed, the parser creates blank nodes **BNodes** for OWL restrictions.
- Example: *Cutting subClassOf repetititons value 1*
- This OWL expression connects a class with one DataProperty and one Literal
- Between Cutting and its restriction one **blank node (BNode)** is used, to store this information



Querying Class Restrictions - 1

- get any value restriction, by querying its restriction
- the restriction is linked to a class
- the class can be queried with *?class rdfs:subClassOf ?restriction*
- extract the linked property with *owl:onProperty* and its value with *owl:hasValue*

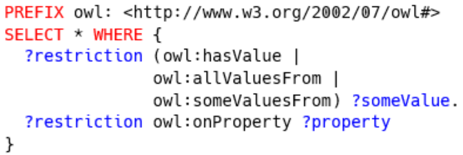


```
PREFIX owl: <http://www.w3.org/2002/07/owl#>
SELECT * WHERE {
  ?restriction owl:hasValue ?someValue.
  ?restriction owl:onProperty ?property
}
```

Abbildung: Query minimum, maximum and qualified cardinality restriction.

Querying Class Restrictions - 2

- get value, existential and quantified restrictions by using their respective properties
- Using | (Logical Or), the query matches the graph against all given properties

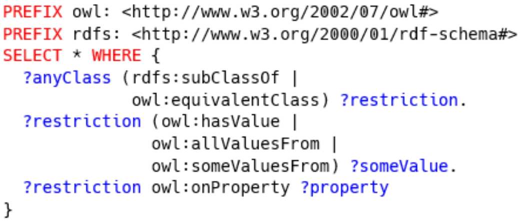


```
PREFIX owl: <http://www.w3.org/2002/07/owl#>
SELECT * WHERE {
  ?restriction (owl:hasValue |
                owl:allValuesFrom |
                owl:someValuesFrom) ?someValue.
  ?restriction owl:onProperty ?property
}
```

Abbildung: Query minimum, maximum and quantified cardinality restriction.

Querying Class Restrictions - 3

- Restrictions can be connected to some class via rdfs:subClassOf property or owl:equivalentClass property
- This queries doesn't work on nested restrictions

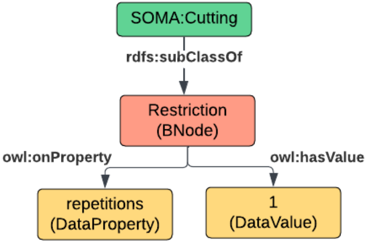


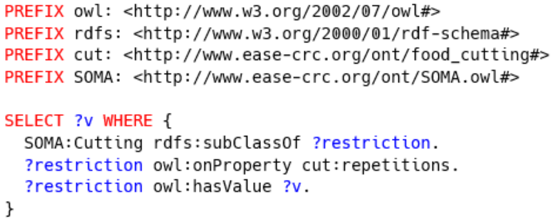
```
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT * WHERE {
  ?anyClass (rdfs:subClassOf |
             owl:equivalentClass) ?restriction.
  ?restriction (owl:hasValue |
                owl:allValuesFrom |
                owl:someValuesFrom) ?someValue.
  ?restriction owl:onProperty ?property
}
```

Abbildung: Connecting the query to a class.

Querying Class Restrictions - 4

Query the number of repetitions for cutting



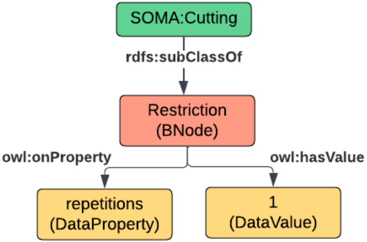


```
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX cut: <http://www.ease-crc.org/ont/food_cutting#>
PREFIX SOMA: <http://www.ease-crc.org/ont/SOMA.owl#>

SELECT ?v WHERE {
  SOMA:Cutting rdfs:subClassOf ?restriction.
  ?restriction owl:onProperty cut:repetitions.
  ?restriction owl:hasValue ?v.
}
```

Abbildung: Query how often an object should be cut

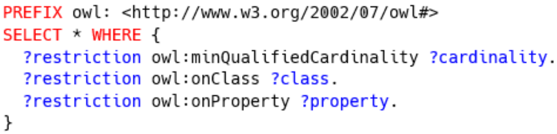
Cardinality Restrictions

- if we want to change a set number (like in the previous example)
- to include min 1, max 10 ...
- example: *Dicing rdfs:subClassOf repetitions min 1*



Querying Cardinality Restrictions - 1

- This query extracts all minimum cardinality restrictions and retrieves cardinality, class and property

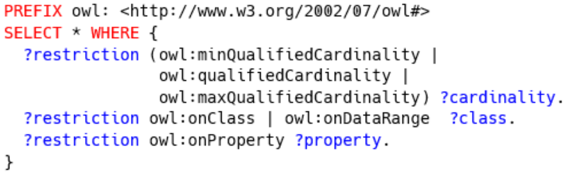


```
PREFIX owl: <http://www.w3.org/2002/07/owl#>
SELECT * WHERE {
  ?restriction owl:minQualifiedCardinality ?cardinality.
  ?restriction owl:onClass ?class.
  ?restriction owl:onProperty ?property.
}
```

Abbildung: Query minimum cardinality restriction.

Querying Cardinality Restrictions - 2

- Query all qualified and min/max cardinality restrictions by using their respective properties

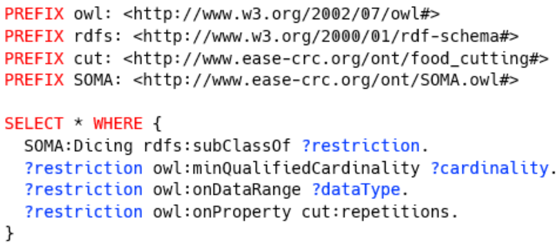


```
PREFIX owl: <http://www.w3.org/2002/07/owl#>
SELECT * WHERE {
  ?restriction (owl:minQualifiedCardinality |
                owl:qualifiedCardinality |
                owl:maxQualifiedCardinality) ?cardinality.
  ?restriction owl:onClass | owl:onDataRange  ?class.
  ?restriction owl:onProperty ?property.
}
```

Abbildung: Query minimum, maximum and qualified cardinality restriction.

Querying Cardinality Restrictions - 3

- Example: Query the following restriction: *Dicing subClassOf repetititons min 1*
- Using cut:repetitions instead of a variable forces the query to search for a restriction with repetitions included (otherwise all matching restrictions are returned)

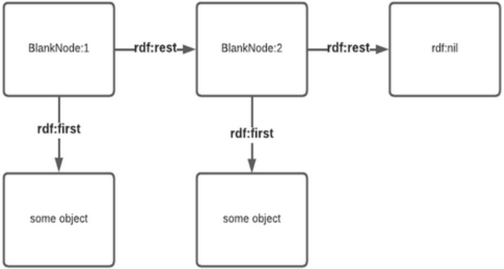


```
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX cut: <http://www.ease-crc.org/ont/food_cutting#>
PREFIX SOMA: <http://www.ease-crc.org/ont/SOMA.owl#>

SELECT * WHERE {
  SOMA:Dicing rdfs:subClassOf ?restriction.
  ?restriction owl:minQualifiedCardinality ?cardinality.
  ?restriction owl:onDataRange ?dataType.
  ?restriction owl:onProperty cut:repetitions.
}
```

Abbildung: Query the repetitions of dicing.

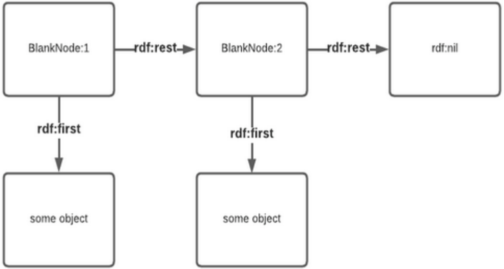
RDF Parsed Graph - Collections

- Collections in RDF are ordered lists
- Blank nodes connect an element of the list via rdf:first and another blank node via rdf:rest
- These kind of lists are parsed recurisvely, each list ends on rdf:nil
- In SPARQL you never have to explicitly query when a list ends



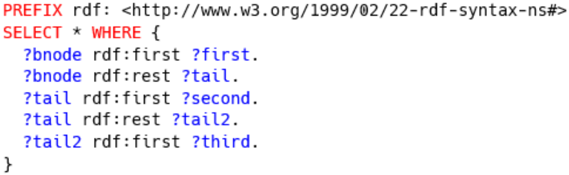
RDF Parsed Graph - Collections

- RDF collections can be queried isolated from all classes.
- they are never directly connected to any class as class restriction
- instead, collections are integrated in class expressions with **Intersections** and **Unions**



Querying RDF Collections

- RDF collections are queried with *rdf:first* and *rdf:rest*.
- Each list element connected via rdf:first contains either a blank node or a value/class.

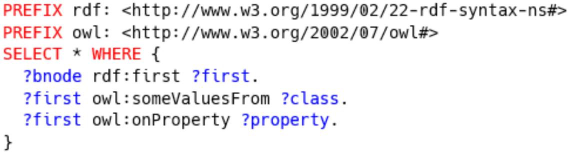


```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT * WHERE {
  ?bnode rdf:first ?first.
  ?bnode rdf:rest ?tail.
  ?tail rdf:first ?second.
  ?tail rdf:rest ?tail2.
  ?tail2 rdf:first ?third.
}
```

Abbildung: Query the first three list elements of a RDF-Collection

Querying RDF Collections - 2

- rdf:first returns the head of the list
- If the head also is a blank node, they can be queried like a class restriction, unless it is a nested class expression

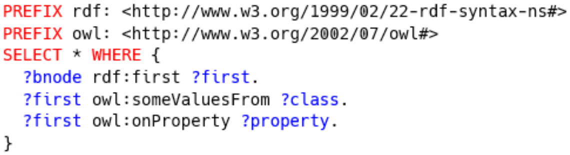


```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
SELECT * WHERE {
  ?bnode rdf:first ?first.
  ?first owl:someValuesFrom ?class.
  ?first owl:onProperty ?property.
}
```

Abbildung: Query all first elements of any collection with an existential restriction

Querying RDF Collections - 3

- Using the *rdf:rest* property matches against blank nodes containing the tail of the collection. The tail itself is connected to two nodes: the next list element and another tail of the collection.
- The list ends if *rdf:rest* matches against *rdf:nil*

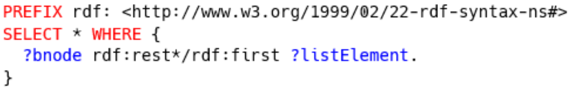


```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
SELECT * WHERE {
  ?bnode rdf:first ?first.
  ?first owl:someValuesFrom ?class.
  ?first owl:onProperty ?property.
}
```

Abbildung: Get existential restriction from collection

Querying RDF Collections - 4

- In SPARQL you can apply pattern matching in a recursive manner.
- rdf:rest/rdf:first: Take relations rdf:rest followed by rdf:first - You get the head of the sublist
- rdf:rest*: You take that path 0 to n times where n is how often that relation occurs.
- It has the disadvantage that you can't split your result into multiple variables

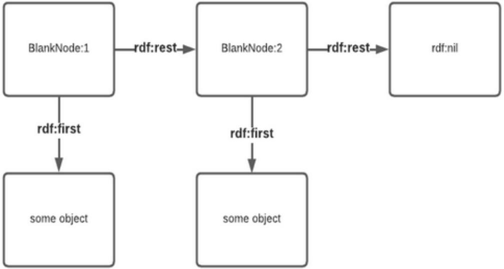


```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT * WHERE {
  ?bnode rdf:rest*/rdf:first ?listElement.
}
```

Abbildung: Query the first three list elements of a RDF-Collection

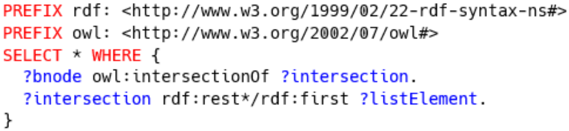
RDF Parsed Graph - Intersections

- RDF collections can be queried isolated from all classes.
- collections are integrated in class expressions with **Intersections** and **Unions**



Querying Intersections - 1

- An intersection is one type of collection in OWL
- Use owl:intersectionOf to query the Intersection of an restriction
- Intersections are identifiable by an **and** within OWL expressions
- Example: *Apple subclassOf (hasPart some (Core and hasEdibility ShouldBeAvoided))*

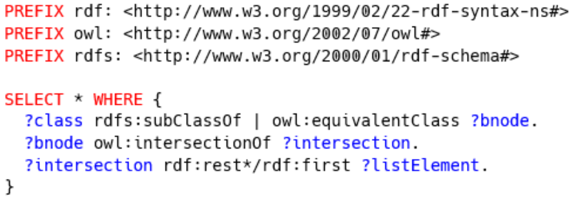


```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
SELECT * WHERE {
  ?bnode owl:intersectionOf ?intersection.
  ?intersection rdf:rest*/rdf:first ?listElement.
}
```

Abbildung: Query intersection and the RDF-Collection

Querying Intersections - 2

- An intersection is one type of collection in OWL
- the class is queried using rdfs:subClassOf or owl:EquivalentClass
- Example: *Apple subclassOf (hasPart some (Core and hasEdibility ShouldBeAvoided))*

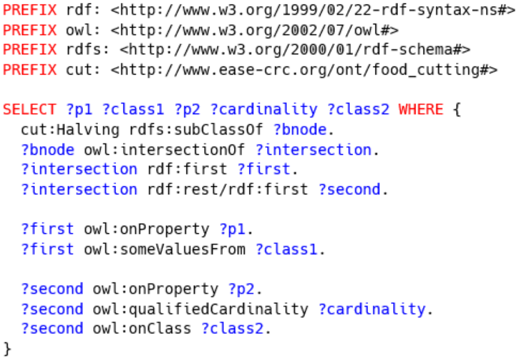


```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

SELECT * WHERE {
  ?class rdfs:subClassOf | owl:equivalentClass ?bnode.
  ?bnode owl:intersectionOf ?intersection.
  ?intersection rdf:rest*/rdf:first ?listElement.
}
```

Abbildung: Query intersection with its members directly attached to any class

Querying Intersections - Example

- Example: Query the following restriction: Halving subClassOf (hasInputObject some Food) and (hasResultObject exactly 2 Halve)
- Use owl:intersectionOf to query the intersection of restrictions.

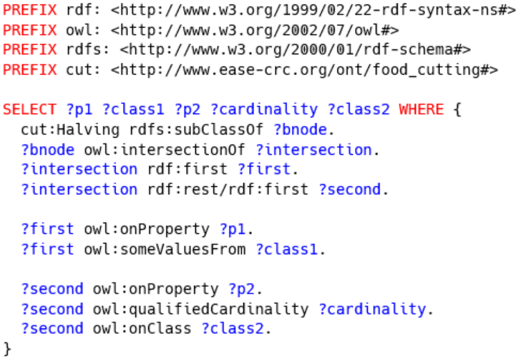


```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX cut: <http://www.ease-crc.org/ont/food_cutting#>

SELECT ?p1 ?class1 ?p2 ?cardinality ?class2 WHERE {
  cut:Halving rdfs:subClassOf ?bnode.
  ?bnode owl:intersectionOf ?intersection.
  ?intersection rdf:first ?first.
  ?intersection rdf:rest/rdf:first ?second.

  ?first owl:onProperty ?p1.
  ?first owl:someValuesFrom ?class1.

  ?second owl:onProperty ?p2.
  ?second owl:qualifiedCardinality ?cardinality.
  ?second owl:onClass ?class2.
}
```

Abbildung: Query input and output of *Halving* task

Querying Intersections - Example

- This intersection has two restrictions - An existential and a qualified cardinality restriction.
- Use owl:intersectionOf to query the intersection of restrictions.

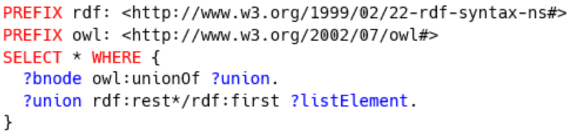


```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX cut: <http://www.ease-crc.org/ont/food_cutting#>

SELECT ?p1 ?class1 ?p2 ?cardinality ?class2 WHERE {
  cut:Halving rdfs:subClassOf ?bnode.
  ?bnode owl:intersectionOf ?intersection.
  ?intersection rdf:first ?first.
  ?intersection rdf:rest/rdf:first ?second.

  ?first owl:onProperty ?p1.
  ?first owl:someValuesFrom ?class1.

  ?second owl:onProperty ?p2.
  ?second owl:qualifiedCardinality ?cardinality.
  ?second owl:onClass ?class2.
}
```

Abbildung: Query intersection and the RDF-Collection

Querying Unions

- An union contains multiple class restrictions
- Use owl:unionOf to query an union of a restriction.
- Unions in an OWL expression can be identified by **or**

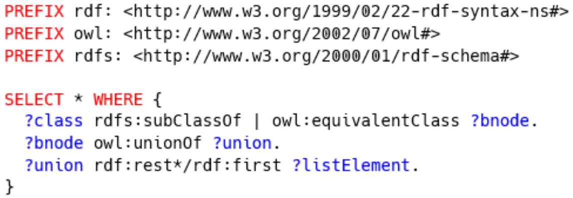


```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
SELECT * WHERE {
  ?bnode owl:unionOf ?union.
  ?union rdf:rest*/rdf:first ?listElement.
}
```

Abbildung: Query members of an OWL-Union

Querying Unions - 2

- Query an union of class restrictions
- Use owl:unionOf to query an union of a restriction

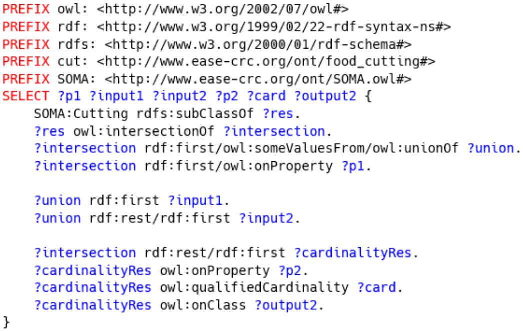


```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

SELECT * WHERE {
  ?class rdfs:subClassOf | owl:equivalentClass ?bnode.
  ?bnode owl:unionOf ?union.
  ?union rdf:rest*/rdf:first ?listElement.
}
```

Abbildung: Query members of OWL-Union directly attached to any class

Example: Querying Nested Restrictions

- Example: Cutting subClassOf (hasInputObject some (Food or FoodPart)) and (hasResultObject exactly 1 FoodPart) and (hasResultObject exactly 1 Slice)



```
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX cut: <http://www.ease-crc.org/ont/food_cutting#>
PREFIX SOMA: <http://www.ease-crc.org/ont/SOMA.owl#>
SELECT ?p1 ?input1 ?input2 ?p2 ?card ?output2 {
    SOMA:Cutting rdfs:subClassOf ?res.
    ?res owl:intersectionOf ?intersection.
    ?intersection rdf:first/owl:someValuesFrom/owl:unionOf ?union.
    ?intersection rdf:first/owl:onProperty ?p1.

    ?union rdf:first ?input1.
    ?union rdf:rest/rdf:first ?input2.

    ?intersection rdf:rest/rdf:first ?cardinalityRes.
    ?cardinalityRes owl:onProperty ?p2.
    ?cardinalityRes owl:qualifiedCardinality ?card.
    ?cardinalityRes owl:onClass ?output2.
}
```

Abbildung: Example union query

Example: Querying Nested Restrictions

- The first 2 triples query the Intersection of the OWL expression, consisting of the these elements:
  1. *(hasInputObject some (Food or FoodPart))*
  2. *(hasResultObject exactly 1 FoodPart)*
  3. *(hasResultObject exactly 1 Slice)*

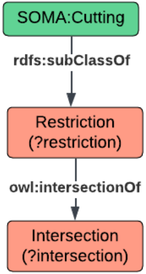


Abbildung: Graph - Intersection of SOMA:Cutting

Example: Querying Nested Restrictions

- The next 4 triples retrieve the first part of the OWL expression

  (hasInputObject some (Food or FoodPart)

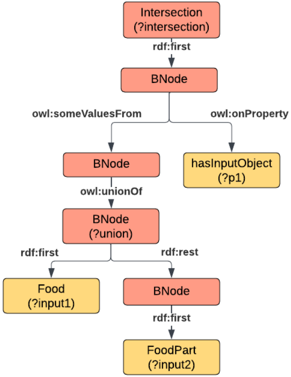


Abbildung: Graph - First member of intersection

Example: Querying Nested Restrictions

- The remaining triples query the second element of the OWL expression:

  (hasResultObject exactly 1 FoodPart)

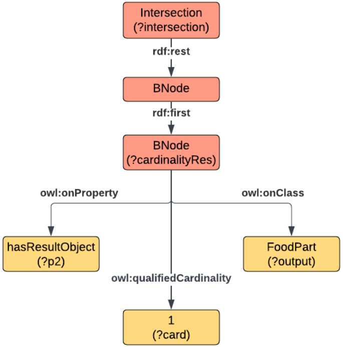


Abbildung: Graph - Second member of intersection

Example: Querying Nested Restrictions

- How is the third element of the intersection queried?
- Query the knowledge graph via this link `https://krr.triply.cc/mkumpel/FruitCuttingKG/sparql`