BUILDING AND QUERYING A KNOWLEDGE BASE FOR E-COMMERCE

Knowledge Representation and Semantic Technologies
a.y. 2020-2021
Engineering in Computer Science (Sapienza)

Alessandra Monaco

TABLE OF CONTENTS

THE TOOL

+ **Popular**
+ **Many features and plugins**
  - Different visualizations (class hierarchy, graph, inference, definitions, …)
  - Many query languages supported (DL, SPARQL, SWQRL)
  - Many reasoners available (HermiT, Pellet, Mastro, Ontop, ELK, …)
  - Syntax translation (Owl functional, RDF/XML, Turtle, …)
  - Debugging plugin (OntoDebug), store test cases, …
  - Short-cuts (make all individuals different, all classes disjoint, copy/import definitions)
+ **Easy to use** (graphic interface, many views)

THE DOMAIN

Domain

E-commerce website, selling technological products (cellphones, pc, accessories, videogames,...)

Subdomains

- **Product Domain**
  (hierarchical organization in categories, properties,...)
- **Customer activity Domain**
  (clicks on products, buying, reviews,...)

Goals

★ Improve products retrieval and **customer experience**

★ **Customer segmentation** (infer new customer classes based on their activities)

★ Query for **similar customers** and **similar products** (co-view, co-buy)

KNOWLEDGE BASE DESIGN

Design choices

- Which aspects of the domain?
- Class or Individual?
- Data or Object? (ex: City, Address, Country, product properties)
- Classes for N-ary relationships (ex: User actions)
- Import definitions of existing vocabularies => **interoperability** (FOAF, VCARD, GOOD RELATION)

THE DOMAIN

What we will see in the tool:

OWLViz OntoGraf

- The **Product**: *concepts*, hierarchy, *ObjectProperties* (ex:PlayStation4Accessory)
- **User activity**: UserAction (ex:ProductBuying), OnlineEcommerceAccount, Agent
- An example: an *instance* of Cellphone
- *DataProperties*

- Necessary condition axioms (=SubClassOf) (ex:Product)
- Necessary and sufficient condition axioms (=EquivalentClasses) (ex:PlayStation4Accessory)
- Covering axiom (ex:Product)
- Disjointness (Classes are assumed to overlap)
- Different Individuals (No unique name assumption)

REASONING TASKS

Reasoners

- We used **Pellet** (support for Owl DL and SWRL)
- ELK does not support DataProperty assertions (Owl 2 EL)
- HermiT does not support sqwrl built-in atoms
- … other possible reasoners …

Reasoning tasks

1. KB **consistency** (is there a model?)
2. KB **coherency** (are all concepts SAT?)
3. **Entailment** (aka Instance Checking)

Debugger

TESTING THE DEBUGGER

→ make the Abox contradicting Tbox

→ make a concept unsatisfiable

QUERYING AND REASONING

DL QUERIES

EXISTENCE OF PROPERTIES

gr:hasBrand **some** gr:Brand

LOW COST GAMING PRODUCT

Product_Gaming **and** hasPrice **some xsd:double**[< 20.0]

CUSTOMERS WHO LIKED PRODUCT X

performsAction **some** (ProductReview **and** reviewRating **value** 5 **and** ofProduct **value** B08BPTKHJH)

QUERYING AND REASONING

DL QUERIES

foaf:OnlineEcommerceAccount **and** performsAction **some** (ProductBuying **and** (ofProduct **some** (Product **and** gr:hasBrand **value** Apple)))

APPLE FAN

→ segmentation by Brand

ITALIAN CUSTOMER

→ segmentation by Country

foaf:OnlineEcommerceAccount **and** performsAction **some** (ProductBuying **and** deliveryLocation **some** (locatedInCity **some** (locatedInCountry **value** Italy)))

performsAction **some** (ofProduct **some** Videogame) **and** (performsAction **some** (ofProduct **some** VideogameConsole) **or** performsAction **some** (ofProduct **some** VideogameAccessory))

GAMER CUSTOMER

→ segmentation by Product Category

QUERYING AND REASONING

SPARQL QUERIES

We will always use these prefixes:

```
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 xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX gr: <http://purl.org/goodrelations/v1#>
PREFIX vcard: <http://www.w3.org/2006/vcard/ns#>
PREFIX : <http://www.semanticweb.org/user/ontologies/2021/1/e-commerce#>
```

PLUGINS:

SPARQL query:

≠

Snap SPARQL Query:

QUERYING AND REASONING

SPARQL QUERIES

GET ALL PRODUCTS

SPARQL query:

```
SELECT DISTINCT ?p ?category
WHERE {
?p a ?category.
?category (rdfs:subClassOf)+ :Product.
}
```

+ Fast
+ All SPARQL features supported

=> You need to explicitly specify each pattern (not a problem)

Snap SPARQL Query:

```
SELECT DISTINCT ?p
WHERE { ?p a :Product. }
```

+ Good editor (autocomplete, keywords and errors highlighted)
+ You don't need to specify inferred patterns
- Much slower (**inference at query time**)
- Some features are missing (LIMIT, path queries, NOT EXISTS)

QUERYING AND REASONING

SPARQL QUERIES

USER HISTORY

```
SELECT ?datetime ?action ?product
WHERE {
?a :performedByUser :Cristian_the_gamer;
        a ?action;
        :actionDatetime ?datetime;
        :ofProduct ?p.
?p gr:name ?product.
FILTER (?action != owl:NamedIndividual)
}
ORDER BY ?datetime
```

PRODUCT RETRIEVAL

```
SELECT DISTINCT ?property ?value
WHERE {
:B07HN8WJN7 ?property ?value
FILTER ( ?property != rdf:type )
}
```

QUERYING AND REASONING

SPARQL QUERIES

```
SELECT DISTINCT ?productID ?name ?price ?brand ?category
WHERE { #Products of the same category
{:B08L5PKKRJ a ?category.
 ?productID a ?category;
      gr:name ?name;
      gr:hasBrand ?brand;
      :hasPrice ?price.
FILTER (?productID != :B08L5PKKRJ) }
UNION #Products of the same brand
{?productID a ?category;
      gr:name ?name;
      gr:hasBrand ?brand;
      :hasPrice ?price.
:B08L5PKKRJ gr:hasBrand ?brand. }
FILTER (?category != owl:NamedIndividual) }
```

RELATED PRODUCTS

→ improved product retrieval
→ recommendations

QUERYING AND REASONING

SPARQL QUERIES

```
SELECT DISTINCT ?product1 ?product2 ?user ?datetime
WHERE {
?buy1 :ofProduct ?p1;
      :ofProduct ?p2;
      a :ProductBuying;
      :performedByUser ?user;
      :actionDatetime ?datetime.
?p1 gr:name ?product1.
?p2 gr:name ?product2.
#avoid duplicates
FILTER (?product1 < ?product2)
}
```

PRODUCTS BOUGHT TOGETHER

→ recommendations

QUERYING AND REASONING

SPARQL QUERIES

```
SELECT ?user1 ?user2 (COUNT (?product) AS ?n_common_products)
WHERE {
?buy a :ProductBuying;
      :ofProduct ?p;
      :performedByUser ?user1.
?buy2 a :ProductBuying;
      :ofProduct ?p;
      :performedByUser ?user2.
?user1 foaf:accountName ?name1.
?user2 foaf:accountName ?name2.
?p gr:name ?product.
#To avoid duplicates (<x,y> and <y,x>)
FILTER (?name1 < ?name2) }
GROUP BY ?user1 ?user2
```

SIMILAR CUSTOMERS: they bought the same products

→ recommendation

QUERYING AND REASONING

SPARQL QUERIES

```
SELECT DISTINCT ?user1 ?user2  ?brand (COUNT (?action) AS ?n_actions)
WHERE {
?action a ?actionClass;
      :ofProduct ?p1;
      :performedByUser ?user1.
?action2 a ?actionClass;
      :ofProduct ?p2;
      :performedByUser ?user2.
?user1 foaf:accountName ?name1.
?user2 foaf:accountName ?name2.
#They like the same brands
?p1 gr:name ?product;
      gr:hasBrand ?brand.
?p2 gr:name ?product;
      gr:hasBrand ?brand.
?actionClass rdfs:subClassOf :UserAction.
#To avoid duplicates (<x,y> and <y,x>)
FILTER (?name1 < ?name2) }
GROUP BY ?user1 ?user2 ?brand
```

SIMILAR CUSTOMERS: they like the same brands

→ customers segmentation and recommendation

Ex : They both bought a product of brand Apple and they both visualized products of brand Apple (=> 2 common actions on a brand)

QUERYING AND REASONING

SPARQL QUERIES

```
SELECT DISTINCT ?product_name ?price
WHERE {
?buy a :ProductBuying;
      :performedByUser :Cristian_the_gamer;
      :ofProduct ?product.
?product gr:name ?product_name;
      :hasPrice ?price;
FILTER ( !EXISTS {
?buy2 a :ProductBuying;
      :performedByUser :alessioNeri87;
      :ofProduct ?product. })
}
```

PRODUCTS TO BE RECOMMENDED based on similar customers

QUERYING AND REASONING

SPARQL QUERIES

```
SELECT DISTINCT ?user_account ?name ?birthday ?gender ?country
      (COUNT (?buy) AS ?n_buy) (SUM (?spent) AS ?total_spent)
WHERE {
#Gather info about people and accounts
?user_account a foaf:OnlineEcommerceAccount.
?user a foaf:Person;
      foaf:name ?name;
      foaf:birthday ?birthday;
      foaf:gender ?gender;
      foaf:account ?user_account.
#Buying activity of these users
?buy a :ProductBuying;
      :performedByUser ?user_account;
      :totalImport ?spent;
      :deliveryLocation ?delivery_location.
?delivery_location :locatedInCity ?city.
?city :locatedInCountry ?country. }
GROUP BY ?user_account ?name ?birthday ?gender ?country
ORDER BY DESC (?total_spent)
```

INFOS REGARDING “GOLD” CUSTOMERS

GOLD CUSTOMERS : Customers that spent a lot of money on this e-commerce website.

QUERYING AND REASONING

SPARQL QUERIES

```
SELECT DISTINCT ?username ?category
WHERE {
?buy a :ProductBuying;
      :performedByUser ?username;
      :ofProduct ?p.
?p a ?type.
?type rdfs:subClassOf ?category.
?category rdfs:subClassOf :Product.
FILTER (?category != owl:NamedIndividual)
FILTER ( NOT EXISTS {
?buy2 a :ProductBuying;
      :performedByUser ?username;
      :ofProduct ?p2.
?p2 a ?type2.
?type2 rdfs:subClassOf ?category2.
FILTER (?category2 != ?category) } )
}
```

ONE-CATEGORY BUYERS

Customers that bought products belonging to just one macro-category.

→ customers segmentation

QUERYING AND REASONING

SPARQL QUERIES

```
SELECT ?review ?user ?product
WHERE {
?review a :ProductReview;
      :ofProduct ?product;
      :performedByUser ?user.
?buy a :ProductBuying;
      :ofProduct ?product;
      :performedByUser ?user.
}
```

VERIFIED PURCHASE

Reviews such that the reviewed product was actually bought by the reviewer customer.

QUERYING AND REASONING

SPARQL QUERIES

BEST SELLERS

```
SELECT DISTINCT ?product_name
       (COUNT (?buy) AS ?n_buying)
WHERE {
?buy a :ProductBuying;
     :ofProduct ?p.
?p gr:name ?product_name
}
GROUP BY ?product_name
HAVING (?n_buying > 1)
ORDER BY DESC (?n_buying)
```

POPULAR BRANDS

```
SELECT DISTINCT ?brand
       (COUNT (?buy) AS ?n_buying)
WHERE {
?p gr:hasBrand ?brand.
?buy a :ProductBuying;
     :ofProduct ?p
}
GROUP BY ?brand
HAVING (?n_buying > 1)
ORDER BY DESC (?n_buying)
```

QUERYING AND REASONING

SPARQL QUERIES

MOST BOUGHT MACRO-CATEGORIES

```
SELECT DISTINCT ?macrocat
       (COUNT (?buy) AS ?n_buying)
WHERE {
?p a ?category.
?category (rdfs:subClassOf)+ ?macrocat.
?macrocat rdfs:subClassOf :Product.
FILTER (?category != owl:NamedIndividual)
?buy a :ProductBuying;
     :ofProduct ?p
}
GROUP BY ?macrocat
HAVING (?n_buying > 1)
ORDER BY DESC (?n_buying)
```

MOST BOUGHT CATEGORIES

```
SELECT DISTINCT ?category
       (COUNT (?buy) AS ?n_buying)
WHERE {
?p a ?category.
FILTER (?category != owl:NamedIndividual)
?buy a :ProductBuying;
     :ofProduct ?p
}
GROUP BY ?category
HAVING (?n_buying > 1)
ORDER BY DESC (?n_buying)
```

QUERYING AND REASONING

SPARQL QUERIES

```
SELECT ?product_name (COUNT (DISTINCT ?buy) AS ?n_buying) (COUNT (DISTINCT ?view) AS ?n_views)
(COUNT (DISTINCT ?review) AS ?n_reviews) (AVG (DISTINCT ?stars) AS ?avg_review_score)
WHERE {
?p gr:name ?product_name.
OPTIONAL {
?buy a :ProductBuying;
      :ofProduct ?p. }
OPTIONAL {
?view a :ProductVisualization;
      :ofProduct ?p. }
OPTIONAL {
?review a :ProductReview;
      :ofProduct ?p;
      :reviewRating ?stars. }
}
GROUP BY ?product_name
HAVING (?n_buying != 0 || ?n_views != 0 || ?n_reviews != 0)
ORDER BY ?product_name
```

PRODUCT ANALYSIS

QUERYING AND REASONING

SQWRL QUERIES

- **S**emantic **Q**uery-enhanced **W**eb **R**ule **L**anguage
- Native understanding of Owl (--> performs inference)
- **Rule-based** :
  <*antecedent: pattern specification*> **->** <*consequent: retrieval specification*>
- Built-in libraries (swrlb, abox, tbox, temporal)
- Supports negation as failure (with sets and set operators)
- Supports aggregation queries

QUERYING AND REASONING

FINAL NOTES

DL QUERIES

- Inference
- **Open World Assumption**
- Can return just (sub/super) classes or individuals
- Can not use variables => no comparisons between values
- Limited set of operators (no aggregation, no arithmetic,...

SPARQL QUERIES

- No automatic inference: everything must be explicit
- **Closed World Assumption**
- Can match any graph pattern
- Many operators and functions, aggregative queries, path queries
- Can return anything you want (properties, values, classes, individuals, results of operators or aggregations,..)

SQWRL QUERIES

- Inference (natively understands Owl)
- **Open World Assumption**
- **Operators for Closure** (sets, bags, set operators,...)
- Allows to query for individuals only
- Many functions provided by built-in libraries

04

KNOWLEDGE BASE VISUALIZATION

OwlViz

+ Good displayer of class hierarchies
- Properties and instances are not displayed

OntoGraph

+ Fast and easy (click to expand)
+ Displays also instances and object properties
+ Change views
+ Search bar, regexp
- Data properties are not displayed

THANK YOU FOR YOUR ATTENTION