

Semantic SPARQL Templates

Christina Davril

Structure

- What is KGQA?
- What are **semantic** SPARQL templates?
- **Key characteristics** of the templates
- *Wikipedia Lists* (benchmark)
- Results of frequency analyses

What is KGQA?

Knowledge Graphs

- Knowledge *Bases*
- **Examples:** Wikidata, PubChem, UniProt
- **RDF:** Resource **D**escription **F**ramework
- Triples:

<subject>

“The Hitchhiker's Guide to the Galaxy”

Q3107329

<predicate>

“author”

P50

<object>

“Douglas Adams”

Q42

- **SPARQL:** SPARQL **P**rotocol and **R**DF **Q**uery **L**anguage

Oskar Lafontaine (Q110719)

German politician

Statements

instance of



human

edit

► 1 reference

member of political party



The Left

edit

start time

29 December 2005

end time

March 2022

► 1 reference



Social Democratic Party of Germany

edit

start time

1966

end time

May 2005

▼ 0 references

+ add reference

Wikidata
example
page

Knowledge Graph Question Answering (KGQA)

QLever

"Give me German politicians and their current party!"



```
SELECT ?person_name ?party_name ?max_join_date ?sitelinks WHERE {  
  { SELECT ?person (MAX(?join_date) AS ?max_join_date) WHERE {  
    ?person wdt:P27 wd:Q183 .  
    ?person p:P102/pq:P580 ?join_date .  
  } GROUP BY ?person }  
  ?person wdt:P27 wd:Q183; p:P102 ?p102 .  
  ?p102 ps:P102 ?party .  
  ?p102 pq:P580 ?max_join_date .  
  ?person @en@rdfs:label ?person_name .  
  ?party @en@rdfs:label ?party_name .  
  ?person ^schema:about/wikibase:sitelinks ?sitelinks .  
}  
ORDER BY DESC(?sitelinks)
```



Angela Merkel
Rosa Luxemburg
Günter Grass
...

Christian Democratic Union
Communist Party
Social Democratic Party
...

What are semantic SPARQL templates?

Semantic SPARQL templates?

1. Politicians
2. Politicians are German
3. The most recent party for each politician



Semantic SPARQL templates?

Semantic Characteristic

1. Politicians
2. Politicians are German
3. The most recent party for each politician

Semantic Category

- CONSTRAINT
- CONSTRAINT
- ATTRIBUTE

Semantic SPARQL templates?

1. Politicians → CONSTRAINT

?politician

wdt:P106/wdt:P279*

wd:Q82955 .

<variable1>

<occupation>/<subclass-of>

<politician>

QLever Output:

?politician

 Q1000051

 Q1000053

...

Semantic Template

“path”: [varIRI] [pred] [obj] .

Semantic SPARQL templates?

2. Politicians are German → CONSTRAINT

?politician wdt:P106/wdt:P279* wd:Q82955 .

?politician wdt:P27 wd:Q183 .

<variable1> <country-of-citizenship> <Germany>

QLever Output:



“add_path”: [cont]
 [varIRI] [pred] [obj] .

Semantic SPARQL templates?

3. The most recent party for each politician → **ATTRIBUTE**

?politician wdt:P106/wdt:P279* wd:Q82955 .

?politician wdt:P27 wd:Q183 .

?politician ??? ?party .

<variable1> <current-political-party?> <variable2>

Oskar Lafontaine (Q110719)

German politician

Statements

instance of



human

edit

► 1 reference

member of political party



The Left

edit

start time

29 December 2005

end time

March 2022

► 1 reference



Social Democratic Party of Germany

edit

start time

1966

end time

May 2005

▼ 0 references

+ add reference

Oskar Lafontaine (Q110719)

German politician

Statements

instance of



human

edit

► 1 reference

member of political party



The Left

edit

start time

29 December 2005

end time

March 2022

► 1 reference



Social Democratic Party of Germany

edit

start time

1966

end time

May 2005

▼ 0 references

+ add reference

“the party with the maximum start time value (for each politician)”

3. The most recent party for each politician → ATTRIBUTE

?politician	wdt:P106/wdt:P279*	wd:Q82955 .	
?politician	wdt:P27	wd:Q183 .	
?politician	p:P102	?p102 .	ATTRIBUTE
?p102	ps:P102	?party .	ATTRIBUTE
?p102	pq:P580	?start_time .	ATTRIBUTE

P102
<member_of_
political_party>

P580
<start_time>

“add_path”: [cont]
[varIRI] [pred] [obj] .

Template identifier: `arg_agg`

Parameters:

- `[distinct]`: aggregation type modifier $\in \{ \text{DISTINCT}, \emptyset \}$
- `[agg]`: aggregation type $\in \{ \text{MAX}, \text{MIN}, \text{AVG} \}$
- `[var1]`: variable whose value(s) corresponding to the aggregated value should be output (*arg*)
- `[var2]`: variable whose values are aggregated
- `[var3]`: variable to store the aggregated value
- `([var4.1], ...)`: variables to group by
- `([var5.1], ...)`: additional variables to project
- `[cont]`: query graph to add to

Template:

```
SELECT [var1] [var3] ([var4.1], ...) ([var5.1], ...)
WHERE {
  {
    SELECT ( [agg] ( [distinct] [var2] ) AS [var3] ) ([var4.1], ...)
    WHERE { [cont] } GROUP BY ([var4.1], ...)
  }
  {
    SELECT ( [var2] AS [var3] ) [var1] ([var4.1], ...) ([var5.1], ...)
    WHERE { [cont] }
  }
}
```


Arguments:

- no **DISTINCT**
- **MAX**
- ?party (the “arg”)
- ?start_time
- ?max_start_time
- ?politician
- no other variables to project
- ?politician wdt:P106/wdt:P279* wd:Q82955 .
...
?p102 pq:P580 ?start_time .

```
SELECT [var1] [var3] ([var4.1], ...) ([var5.1], ...)
WHERE {
  {
    SELECT ( [agg] ( [distinct] [var2] ) AS [var3] ) ([var4.1], ...)
    WHERE { [cont] } GROUP BY ([var4.1], ...)
  }
  {
    SELECT ( [var2] AS [var3] ) [var1] ([var4.1], ...) ([var5.1], ...)
    WHERE { [cont] }
  }
}
```

```

6 SELECT ?party ?max_start_time ?politician
7 WHERE {
8   {
9     SELECT ( MAX(?start_time) AS ?max_start_time ) ?politician
10    WHERE {
11      ?politician wdt:P106/wdt:P279* wd:Q82955 .
12      ?politician wdt:P27 wd:Q183 .
13      ?politician p:P102 ?p102 .
14      ?p102 ps:P102 ?party .
15      ?p102 pq:P580 ?start_time .
16    } GROUP BY ?politician
17  }
18  {
19    SELECT ( ?start_time AS ?max_start_time ) ?party ?politician
20    WHERE {
21      ?politician wdt:P106/wdt:P279* wd:Q82955 .
22      ?politician wdt:P27 wd:Q183 .
23      ?politician p:P102 ?p102 .
24      ?p102 ps:P102 ?party .
25      ?p102 pq:P580 ?start_time .
26    }
27  }
28 }
29

```

- The most recent party for each politician → ATTRIBUTE

QLever Output:

?party	?max_start_time	?politician
Q166027	2004-01-01T00:00:00Z	Q1000800
Q49768	1930-01-01T00:00:00Z	Q1001323

Human-readable output, duplicate-free, showing well-known entities first

4. Names of politicians and political parties → ATTRIBUTE

“add_name”: [cont]
[var1.1] [pred] [var2.1] . FILTER (LANG[var2.1] = [lang1])
[var1.2] [pred] [var2.2] . FILTER (LANG[var2.2] = [lang2])

?politician rdfs:label ?politician_name . FILTER(LANG(?politician_name)="en")

5. Number of Wikipedia site links per politician → ATTRIBUTE

“add_path”: [cont]
[varIRI] [pred] [obj] .

?politician ^schema:about/wikibase:sitelinks ?sitelinks .

6. Output politician names and party names (*distinct* pairs) → OUTPUT

“select”: `SELECT [distinct] ([var1], ...) WHERE {
[cont]
}`

```
SELECT DISTINCT ?politician_name  
                ?party_name WHERE {  
                ...  
}
```

7. Order by descending number of Wikipedia site links → OUTPUT

“order”: `[cont]
ORDER BY [order1] ([var1]) [order2] ([var2])`

`DESC(?sitelinks)`

QLever Output:

	?politician_name	?party_name
1	Angela Merkel	Christian Democratic Union
2	Rosa Luxemburg	Communist Party of Germany
3	Konrad Adenauer	Christian Democratic Union

<https://qllever.cs.uni-freiburg.de/wikidata/nuT7zG>

Key characteristics of the templates

Why does “arg_agg” contain code duplication / two subqueries?

```
6 SELECT ?party ?max_start_time ?politician
7 WHERE {
8   {
9     SELECT ( MAX(?start_time) AS ?max_start_time ) ?politician
10    WHERE {
11      ?politician wdt:P106/wdt:P279* wd:Q82955 .
12      ?politician wdt:P27 wd:Q183 .
13      ?politician p:P102 ?p102 .
14      ?p102 ps:P102 ?party .
15      ?p102 pq:P580 ?start_time .
16    } GROUP BY ?politician
17  }
18  {
19    SELECT ( ?start_time AS ?max_start_time ) ?party ?politician
20    WHERE {
21      ?politician wdt:P106/wdt:P279* wd:Q82955 .
22      ?politician wdt:P27 wd:Q183 .
23      ?politician p:P102 ?p102 .
24      ?p102 ps:P102 ?party .
25      ?p102 pq:P580 ?start_time .
26    }
27  }
28 }
29
```

Why does “arg_agg” contain code duplication / two subqueries?

```
6 SELECT ?party ?max_start_time ?politician
7 WHERE {
8   {
9     SELECT ( MAX(?start_time) AS ?max_start_time ) ?politician
10    WHERE {
11      ?politician wdt:P106/wdt:P279* wd:Q82955 .
12      ?politician wdt:P27 wd:Q183 .
13      ?politician p:P102 ?p102 .
14      ?p102 ps:P102 ?party .
15      ?p102 pq:P580 ?start_time .
16    } GROUP BY ?politician
17  }
18  {
19    SELECT ( ?start_time AS ?max_start_time ) ?party ?politician
20    WHERE {
21      ?politician wdt:P106/wdt:P279* wd:Q82955 .
22      ?politician wdt:P27 wd:Q183 .
23      ?politician p:P102 ?p102 .
24      ?p102 ps:P102 ?party .
25      ?p102 pq:P580 ?start_time .
26    }
27  }
28 }
29
```

→ To account for ties.

Question Answering over Linked Data: QALD-9-plus, ID 351, April 2022

“Who is the Formula 1 race driver with the most races?”

Gold query:

```
SELECT DISTINCT ?uri
WHERE {
    ?uri wdt:P106 wd:Q10841764 .
    ?uri wdt:P1350 ?num .
}
ORDER BY DESC(?num) LIMIT 1
```

Q10841764	P1350
<Formula-One-driver>	<number of matches played/races/starts>

“Correct result”: Michael Schumacher (Q9671)

Question Answering over Linked Data: QALD-9-plus, ID 351, April 2022

“Who is the Formula 1 race driver with the most races?”

Gold query:

```
SELECT DISTINCT ?uri
WHERE {
    ?uri wdt:P106 wd:Q10841764 .
    ?uri wdt:P1350 ?num .
}
ORDER BY DESC(?num) LIMIT 1
```

Q10841764 P1350
<Formula-One-driver> <number of matches played/races/starts>

Problems:

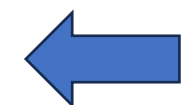
- Does not account for ties
→ Uses world knowledge
- Does not provide a source / verifiable ground truth
→ Why is the result not Kimi Räikkönen?
- Generic variables

“Correct result”: Michael Schumacher (Q9671)

This list is accurate as of the [2022 Australian Grand Prix](#). Drivers who only participated in Friday practice and who were not actually entered for the race are not included.

Formula One drivers by name

Driver Name ↕	Nationality ↕	Seasons Competed ↕	Drivers' Championships ↕	Race Entries ↕	Race Starts ↕	Pole Positions ↕	Race Wins ↕
Kimi Räikkönen^	 Finland	2001–2009 , 2012–2021	1 2007	353	349	18	21
Fernando Alonso~	 Spain	2001 , 2003–2018 , 2021–2022	2 2005–2006	339	336	22	32
Rubens Barrichello	 Brazil	1993–2011	0	326	322	14	11
Jenson Button^	 United Kingdom	2000–2017	1 2009	309	306	8	15
Michael Schumacher^	 Germany	1991–2006 , 2010–2012	7 1994–1995 , 2000–2004	308	306	68	91



Inclusion of zero counts using the “agg” template

Examples: “Film series and how many films they contain”, “Actors and their number of won awards”, ...

QALD-9-plus: 8 out of 412 examples

QALD-10: 3 out of 394 examples

Problems:

→ Indicates lack of variety regarding query structures

→ Counts of zero are only included in one example where an average count is required

QALD-10, ID 23: “How many spouses do heads of state have on average?”

- 1) “agg”: Count spouse statements for each head of state, incl. 0
- 2) “agg_all”: Take the mean

Inclusion of zero counts using the “agg” template

“agg” sub-type for COUNT:

```
SELECT (COUNT ( [distinct1] [var1.1] ) AS [var2.1] ) ([var3.1], ...)  
WHERE {  
    [cont2] _____> Items for which counts are computed  
    OPTIONAL { [cont1] } _____> Counted items  
}  
GROUP BY ([var3.1], ...)
```

Pro:

- Includes (real) counts of zero
- One version of “agg” for COUNT aggregation is enough.

Con:

- Includes missing values.
- More complicated to use.

Summary

- 18 templates
- Knowledge graph-independent
- For lower-level semantic purposes: e.g., “path”
- For higher-level semantic purposes: e.g., “arg_agg”
- Made to be generally applicable
- Use a basic set of SPARQL 1.1 constructs, e.g., no **HAVING** in addition to **FILTER**
- Based on Wikidata-based benchmark/dataset: *Wikipedia Lists*

Wikipedia Lists (benchmark)






Wikipedia Lists

- 60 examples
- examples are based on (information in) Wikipedia lists
 - re-creations of full Wikipedia lists → often tabular or table-like
 - questions about aspects of Wikipedia lists
- **Handwritten query version (sometimes created using templates as aid!)**
- **Generated, template-based query version**
- The Wikidata output (QLever) was compared to the info in the Wikipedia list and discrepancies resolved or documented
- The results of the handwritten and generated queries match

Wikipedia list vs. QLever Output

Zone 5: South and Central Americas

- 500 –  Falkland Islands
 - 500 –  South Georgia and the South Sandwich Islands
- 501 –  Belize
- 502 –  Guatemala
- 503 –  El Salvador

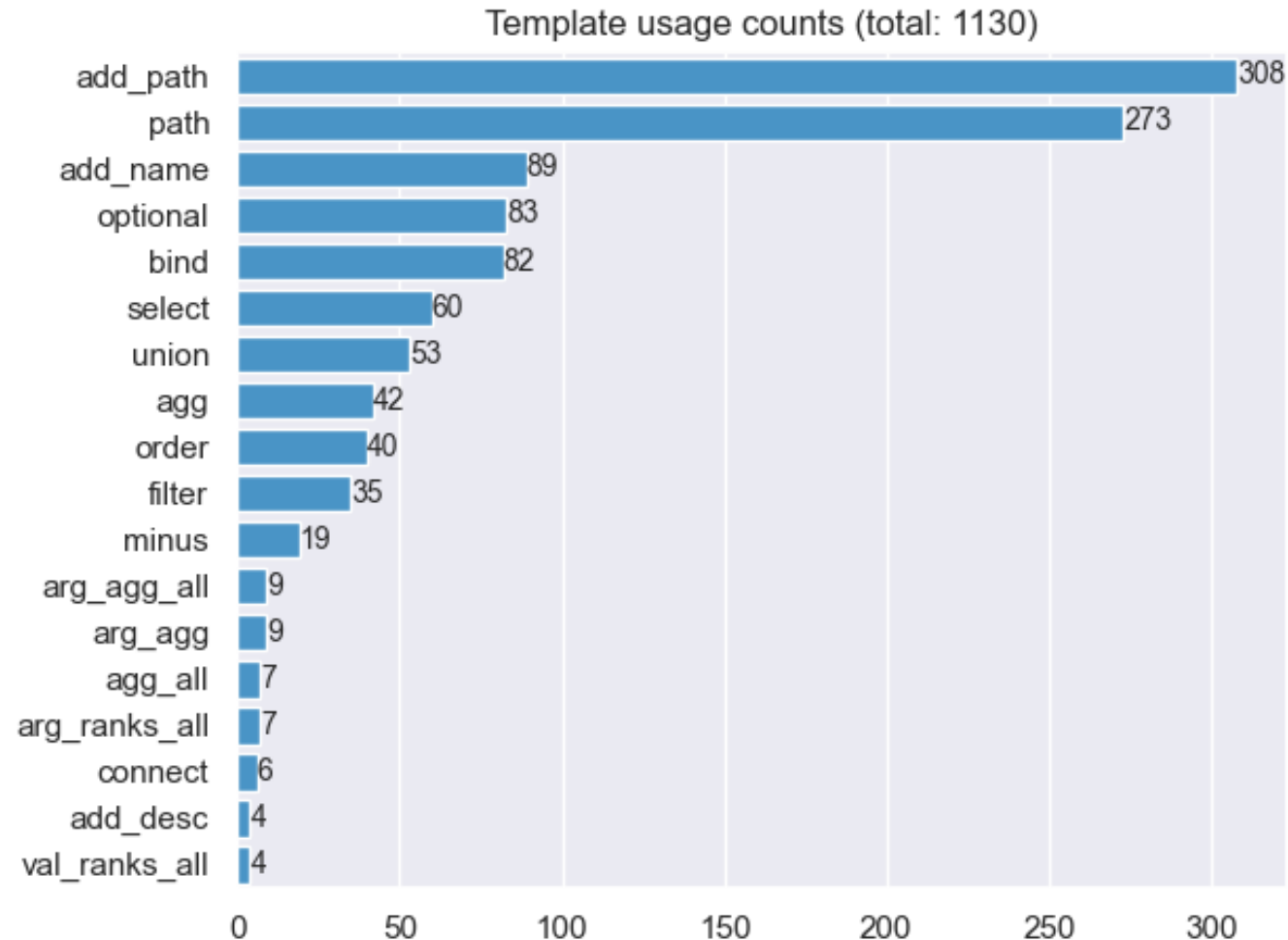
?country_calling_code	?sample_flag_image	?country_name
+500		Falkland Islands
+500		South Georgia and the South Sandwich Islands
+501		Belize
+502		Guatemala
+503		El Salvador

[en.wikipedia.org/wiki/List of country calling codes](https://en.wikipedia.org/wiki/List_of_country_calling_codes)

glever.cs.uni-freiburg.de/wikidata/fKDg2G

Results of frequency analyses

Template usage analysis



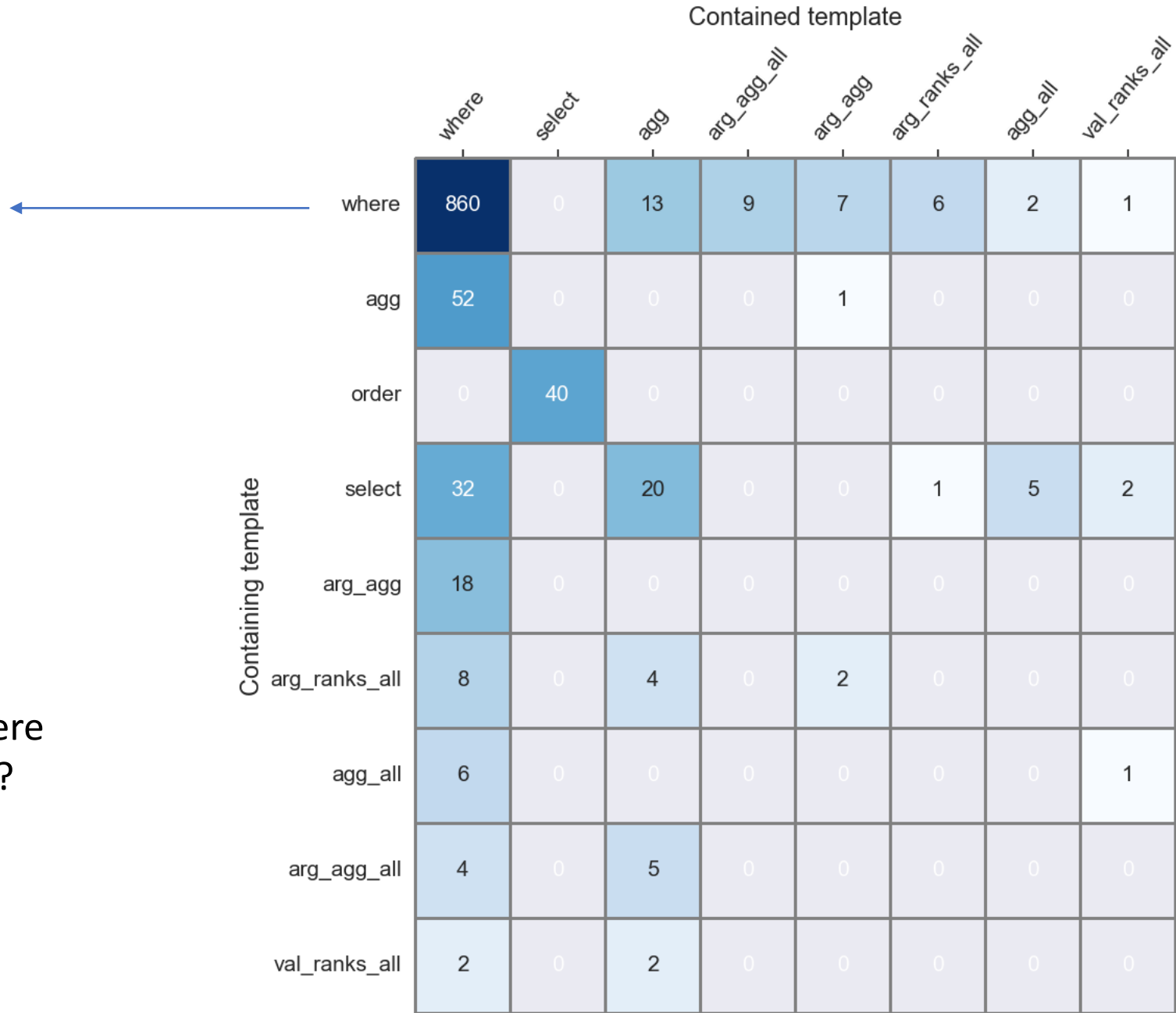
The 60 examples had 1,130 **instances of templates.**

path
 add_path
 union
 minus
 bind
 filter
 optional
 add_name
 add_desc

Which (other) templates were inserted into the templates?

“add_path”:

[cont]
 [varIRI] [pred] [obj] .



Example: “arg_agg_all” containing “agg”

“Which US president was played by the most actors in a movie? Also show the actors”

“Which Formula One driver won the most championships and in which years?”

“Which movie has won the most Oscars?”

“Who composed the music for the most Pixar films (excluding short films)?”

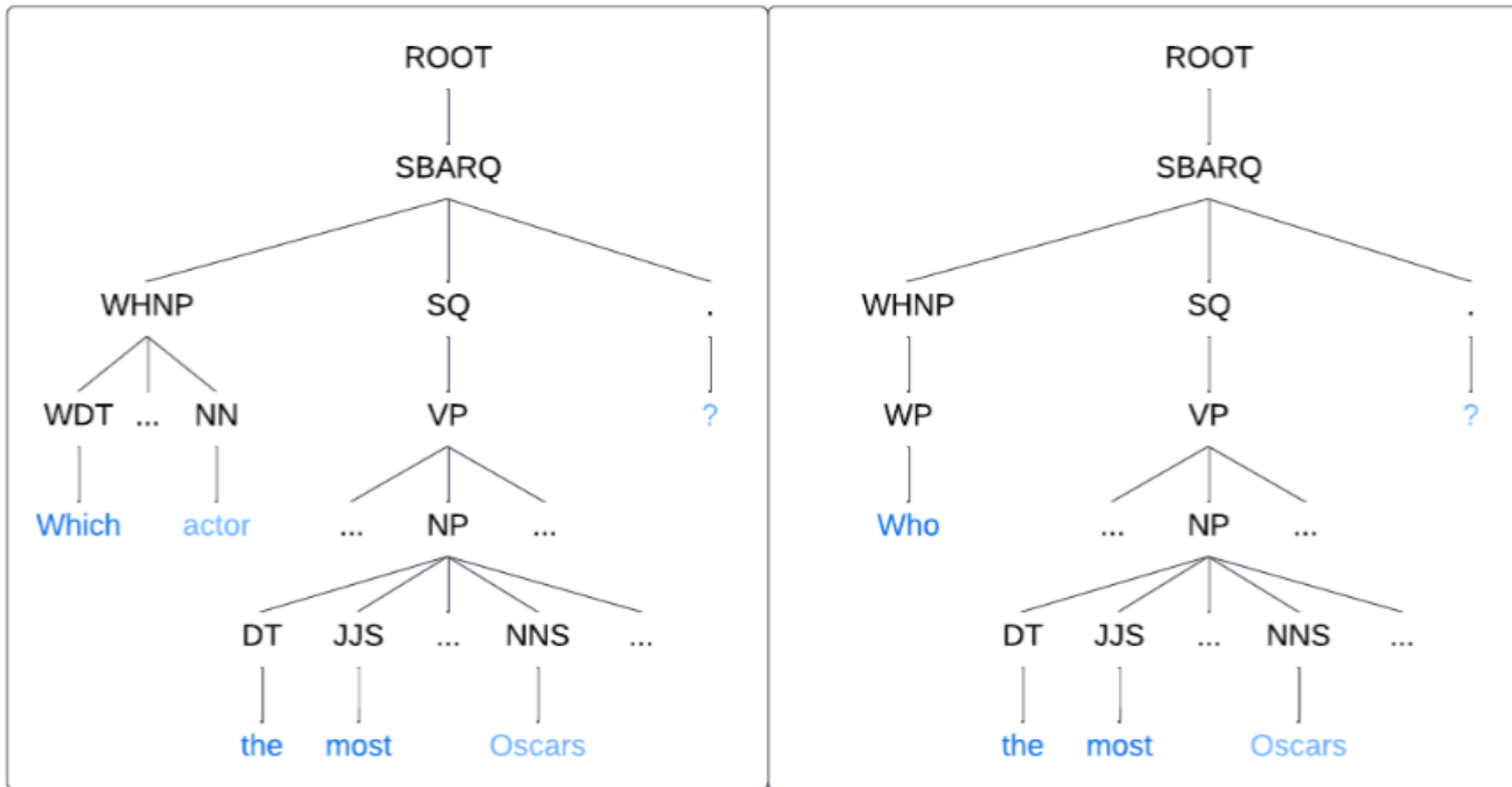
“Which country borders the most other countries?”

→ Clear pattern that may help recognize that this combination of templates is required

→ Same aggregators used every time:

COUNT in “agg”

MAX in “arg_agg_all”



Syntactic structure created with the help of the Stanford Lexicalized Parser v2.0.4.
Model: English PCFG; Tags: Penn Treebank syntactic and POS tagsets

Thank you for your attention and participation!

Questions?

Extra slides

Overview: Semantic Categories & Templates

CONSTRAINT: path, add_path, connect, filter, minus, arg_ranks_all, val_ranks_all

ATTRIBUTE: add_path, add_name, add_desc

AGGREGATE: agg, agg_all, arg_agg, arg_agg_all

COMBINE: union, bind

OUTPUT: select, order

wikipedia_lists.json – example entry

{

```
"id": 44,  
"aggregation": true,  
"question": "How many nuclear reactors operate in each country? Sort by descending number and only include  
countries that have operational reactors",  
"wikipedia_list": "https://en.wikipedia.org/wiki/Nuclear_power_by_country#Overview",  
"hm_qlever_link": "https://qlever.cs.uni-freiburg.de/wikidata/PeqCNw",  
"hm_query": "SELECT DISTINCT ?country_name (COUNT(DISTINCT ?nuclear_reactor) AS ?num_nuclear_reactors)  
WHERE { ?country wdt:P31/(wdt:P279)* wd:Q6256 OPTIONAL { ?nuclear_reactor wdt:P31/(wdt:P279)* wd:Q134447 .  
?nuclear_reactor wdt:P5817 wd:Q55654238 ; wdt:P17 ?country } ?country @en@rdfs:label ?country_name } GROUP  
BY ?country ?country_name HAVING ( ?num_nuclear_reactors > 0 ) ORDER BY DESC(?num_nuclear_reactors)",  
"results_comparison": "Even though the Wikidata output contains much fewer entities, four out of the top 5 of the  
Wikipedia list are within the top 5 of the Wikidata output, suggesting that the query is adequate. Countries were  
retrieved using a simple triple as the filtering by the number of operational reactors already eliminated many  
undesirable result entities.",  
"tb_query": "SELECT DISTINCT ?country_name ?num_nuclear_reactors WHERE { { { SELECT (COUNT(DISTINCT  
?nuclear_reactor) AS ?num_nuclear_reactors) ?country WHERE { ?country wdt:P31/wdt:P279* wd:Q6256 .  
OPTIONAL { ?country wdt:P31/wdt:P279* wd:Q6256 . ?nuclear_reactor wdt:P31/wdt:P279* wd:Q134447 .  
?nuclear_reactor wdt:P5817 wd:Q55654238 . ?nuclear_reactor wdt:P17 ?country . } } GROUP BY ?country } }  
FILTER(?num_nuclear_reactors > 0) ?country rdfs:label ?country_name . FILTER(LANG(?country_name)='en') }  
ORDER BY DESC(?num_nuclear_reactors)"
```

},

```

"44": {
  "[1]": {
    "desc": "countries",
    "template": "path",
    "arguments": [
      "?country",
      "wdt:P31/wdt:P279*",
      "wd:Q6256"
    ]
  },
  "[2]": {
    "desc": "nuclear reactors per country",
    "template": "add_path",
    "arguments": [
      "?nuclear_reactor",
      "wdt:P17",
      "?country",
      "[1]"
    ]
  },
  "[3]": {
    "desc": "nuclear reactors are nuclear reactors",
    "template": "add_path",
    ...

```

semantic_plans.json – example entry

Which (other) templates were inserted into the templates?

“add_path”:
[cont]
[varIRI] [pred] [obj] .

		Contained template																
		add_path	path	optional	select	add_name	bind	agg	union	filter	arg_agg_all	minus	agg_all	arg_agg	arg_ranks_all	connect	val_ranks_all	add_desc
Containing template	add_path	187	74	7		3	4		13	7		6	2	2	1	2		
	union	28	114	3			6		8	4		2						1
	optional	9	31	47		34	29		5	8					2			
	order				40													
	bind	29	8	7		3	22	6	3					3			1	
	add_name	14	28	14			4	1	4	8	8	2		1	3	1		1
	agg	4	7	5		23	1		4	1		6		1				1
	select					22	8	20		2			5		1		2	
	minus	8	12				1		16							1		
	arg_agg	14								2		2						
	filter	12		3		2	5	6		2	1	2		1				1
	arg_ranks_all	8						4						2				
	connect	3	6									1				2		
	arg_agg_all	1					1	5	1	1								
	agg_all	1	1			1	2			1							1	
	add_desc	1				2												
	val_ranks_all	2						2										
	path																	

Missing functionality in SPARQL

Among the templates, there are “arg_ranks_all” and “val_ranks_all” ...

arg_ranks_all e.g., needed for “Which are the top three electronegative chemical elements?”

→ ranks 1-3 overall

val_ranks_all e.g., needed for “How fast is the world’s second fastest man (100 m sprint)?”

→ rank 2 overall

... but there are no good templates for „arg_ranks “ and „val_ranks“ (with grouping)!

Example: “What is the second largest country on each continent?”

Attempts either require computationally expensive operations or pre-generation!