The QLever UI

+ its connection to other projects in our group
+ a crash course on some knowledge base basics

Talk @ AD Group Seminar
Freiburg, October 6, 2020

Hannah Bast
Algorithms & Data Structures Group
Department of Computer Science
University of Freiburg, Germany

UNI FREIBURG

Overview over this talk

Part 1	Knowledge Base Basics	12 slides
Part 2	Connection to other projects in our group	4 slides
Part 3	The Qlever UI	7 slides*

Lots of examples and demos

Please interrupt me at any time
if something is not clear

* One of these 7 slides has a list of the progress and features added to the Qlever UI over time, and I will spend quite a bit of time on that slide

Knowledge Base Basics 1/12

- Knowledge bases, simple version
  - A knowledge base can be represented as a collection of subject-predicate-object triples, for example:

```
<Meryl Streep>    <is-a>           <Person>              .
<Meryl Streep>    <Gender>         <Female>              .
<Meryl Streep>    <Date of birth>  "1949-06-22"          .
<Meryl Streep>    <Award won>      <Oscar Best Actress>  .
```

  - Modeling data as triples is the central element of the **Resource Description Framework** (RDF) specification; also:

    Each part of a triple is an International Resource Identifier (IRI)

    An IRI is just like a URI, but allowing more characters (ä, Ж, ず, ...)

    Objects can also be strings, so-called **literals**

    Literals can have languages and (basic) data types, see later slide

  - RDF is not a data format; what we see above is one of several data formats ("serializations") for RDF data, called **N-Triples** (NT)

Knowledge Base Basics 2/12

- The standard query language is SPARQL
  - A simple example query: all winners of an Oscar for Best Actress in the knowledge base, together with their birth date, youngest first

```
SELECT ?person ?date_of_birth WHERE {
  ?person    <Award won>      <Oscar Best Actress>      .
  ?person    <Date of birth>  ?date_of_birth            .
}
ORDER BY DESC(?date_of_birth)
```

  - Note how the query syntax is similar to the N-Triples input format

    Except that each component of a triple can also be a **variable**

  - The result is a **table** (one column per variable in the SELECT clause)

    Each row is simply an assignment to the variables such that all the triples in the body of the query exist in the knowledge base

    NOTE: If in the query above, a person has k birth dates, there would be k rows in the table for that person, one for each birth date

Knowledge Base Basics 3/12

- Important SPARQL features
  - List of features used in many typical queries (by example); understand that in SPARQL each intermediate result is a **table**

ORDER BY DESC(?year)	Order result rows by value of this column
OPTIONAL { ... }	enclosed triples can but don't have to match
FILTER (?height > 8000)	Reduce to rows with specified property
GROUP BY ?x	Conflate rows with same value for ?x
{ ... } UNION { ... }	union of two result tables
VALUES { ... }	create result table with explicit values
{ SELECT ... }	Subquery within a query (arbitrarily nestable)
BIND (?old AS ?new)	Copy result column (or modify with function)
<birth place>|<nationality>	Union of two predicates
<birth place>/<contained>	Predicate path of length 2
<contained>*	Transitive hull (+ = at least once)

Knowledge Base Basics 4/12

- Knowledge bases
  - Our first knowledge base example was oversimplified in two ways

    1. The IRIs were short and not universally unambiguous

    The same IRI might mean something different in a different knowledge base and then we have trouble when we combine them

    2. In real knowledge bases, IRIs do not have human-readable names, but those names are specified via a dedicated predicate

    Reason: we want IDs to be stable, but the name of an entity may change + an entity has a different name in different languages

  - In Wikidata, Meryl Streep has the (globally unique and stable) IRI

    <http://www.wikidata.org/entity/Q873>

    and the names in the various languages are specified as follows

```
<http://.../Q873>  <http://.../rdf-schema#label>  "Meryl Streep"@en  .
<http://.../Q873>  <http://.../rdf-schema#label>  "Merila Strīpa"@lv  .
<http://.../Q873>  <http://.../rdf-schema#label>  "Мерил Стрип"@ru  .
```

Knowledge Base Basics 5/12

- Knowledge bases
  - In Wikidata, the information about Meryl Streep looks as follows

    We now use the more compact **Turtle** (TTL) format, which has features like IRI prefixes and ; for stating multiple triples for the same subject

```
@prefix  wd:   <http://www.wikidata.org/entity>
@prefix  wdt:  <http://www.wikidata.org/prop/direct/>
@prefix  xsd:  <http://www.w3.org/2001/XMLSchema#>
@prefix  rdfs: <http://www.w3.org/2000/01/rdf-schema#>

wd:Q873          wdt:P31      wd:Q5                              ;
                 wdt:P21      wd:Q6581072                        ;
                 wdt:P569     "1949-06-22"^^xsd:dateTime         ;
                 wdt:P166     wd:Q103618                         .

wd:Q873          rdfs:label   "Meryl Streep"@en                  .
wd:Q5            rdfs:label   "human"@en                         .
wd:Q6581072      rdfs:label   "female"@en                        .
wd:Q103618       rdfs:label   "Academy Award for Best Actress"@en .
```

Knowledge Base Basics 6/12

- Reification, problem
  - The RDF idea of casting everything as **triples** is very elegant

    Compare this to a database where we usually have multiple tables and we need to understand the schema of each table (= what each column means) to understand what the data means
  - But consider this information about Meryl Streep's three Oscars:

1980	Oscar for Best Supporting Actress	"Kramer vs. Kramer"
1983	Oscar for Best Actress	"Sophie's Choice"
2912	Oscar for Best Actress	"The Iron Lady"
  - This seems to require an n-ary relation, where n > 2

    n = 4 in this case, because each element has four pieces of information: person, year, award, film
  - How can we cast this as triples? (which inherently capture only binary information, namely between a subject and an object)

Knowledge Base Basics 7/12

- Reification, first try
  - Consider the following triples (in the simpler N-Triples format again)

```
<Meryl Streep>  <won award>          <Oscar Best Supporting Actress> .
<Meryl Streep>  <won award>          <Oscar Best Actress> .
<Meryl Streep>  <won award in year>  "1980" .
<Meryl Streep>  <won award in year>  "1983" .
<Meryl Streep>  <won award in year>  "2012" .
<Meryl Streep>  <won award for film> <Kramer vs. Kramer> .
<Meryl Streep>  <won award for film> <Sophie's Choice> .
<Meryl Streep>  <won award for film> <The Iron Lady> .
```

  - Problem: we have now **lost information**

    The triples tell us that Meryl Streep has won awards in the years 1980, 1983, and 2021. They don't us tell which award in which year for what

Knowledge Base Basics 8/12

- Reification, Wikidata solution
  - For each element of the n-ary relation, introduce an intermediate entity and connect each component of the element to that node via a normal triple

    In Wikidata, these intermediate entities are called **statement nodes**
  - For example, the statement node for Meryl Streep's 2012 Oscar is called

    <http://.../statement/Q873-6ad4311a-47f9-8d9b-7c91-d387e71529ac>
  - The information about that Oscar is expressed via the following triples

```
@prefix  wd:   <http://www.wikidata.org/entity/>
@prefix  wds:  <http://www.wikidata.org/entity/statement/>
@prefix  p:    <http://www.wikidata.org/prop/>
@prefix  ps:   <http://www.wikidata.org/prop/statement/>
@prefix  pq:   <http://www.wikidata.org/prop/qualifier/>

wd:Q873                          p:166      wds:Q873-...-d387e71529ac .
wds:Q873-...-d387e71529ac        ps:P166    wd:Q103618    .   # award
wds:Q873-...-d387e71529ac        pq:P585    "2011-01-01"  .   # point in time
wds:Q873-...-d387e71529ac        pq:P1686   wd:Q269810    .   # for work
```

Knowledge Base Basics 9/12

- Reification, predicate variants and names
  - With reification, the "same" predicate can appear in different roles
  - For example, for the Wikidata property for "award received" we have:

wdt:P166	from an entity direct**t**ly to its main property
p:P166	from an entity to a statement node
ps:P166	from a statement node to the main property of that **s**tatement
pq:P166	from a statement node to an additional property ("**q**ualifier")
...	five more variants :o

  - Technically, these are completely different predicates; but their common suffix (P166) indicates that they have something in common; in particular, they share a common name via a connection to a "meta entity":

```
wd:P166   rdfs:label                  "award received"@en   .
wd:P166   wikibase:directClaim        wdt:P166              .
wd:P166   wikibase:claim              p:P166                .
wd:P166   wikibase:statementProperty  ps:P166               .
wd:P166   wikibase:qualifier          pq:P166               .
```

Knowledge Base Basics 10/12

- Interim conclusion
  - RDF, stable IDs, name predicates, reification are very elegant ideas

    Indeed, knowledge bases prior to Wikidata did not implement them so consistently, which really complicated working with them in depth
  - But one consequence is that even seemingly simple queries can becomes quite complex in SPARQL, for example

    All Oscars of Meryl Streep (and the films she won them for)

    Note that this is not a constructed query, but the kind of query which people ask on Google (expecting an informative answer)

    Let's look at the correct (and simplest) SPARQL query for this on the next slide

Knowledge Base Basics 11/12

- SPARQL query for Meryl Streep's Oscars

```
PREFIX  wd:   <http://www.wikidata.org/entity/>
PREFIX  wdt:  <http://www.wikidata.org/prop/direct/>
PREFIX  p:    <http://www.wikidata.org/prop/>
PREFIX  ps:   <http://www.wikidata.org/prop/statement/>
PREFIX  pq:   <http://www.wikidata.org/prop/qualifier/>
PREFIX  rdfs: <http://www.w3.org/2000/01/rdf-schema#>

SELECT ?film ?award WHERE {
  wd:Q873    p:P166     ?s           .    # to statement node
  ?s         ps:P166    ?award_id    .    # award
  ?s         pq:P1686   ?film_id     .    # for work
  ?award_id  wdt:P31    wd:Q19020    .    # instance of
  ?film_id   rdfs:label ?film        .    # name
  ?award_id  rdfs:label ?award       .    # name
  FILTER (LANG(?film) = "en")             # only English name
  FILTER (LANG(?award) = "en")            # only English name
}
```

Knowledge Base Basics 12/12

- Observations
  - Let us stop here with the "basics" for this talk, but understand:

    1. This is by far not everything: there are yet **more subtleties** and challenges when working with knowledge bases

    2. These are **not design flaws**; rather, representing large amounts of knowledge in a structured way has an inherent complexity

  - What about natural language, does it have the same problems?

    3. It is indeed the beauty of natural language that it allows to capture this complexity (and more) in a very **flexible** way, **without strict rules**

    4. There is a price: natural language is inherently **fuzzy**; one can make 100% precise statements in natural language, but that takes effort

    5. Also, there is no way that one can extract complex information from a text collection as **precisely** as from a knowledge base

    This is why I strongly believe that both text **and** knowledge bases are important, and no amount of AI will change that (in the next 50 years)

Other projects in our group 1/4

UNI FREIBURG

- Question Answering
  - To answer questions from a knowledge base, we need to translate a natural language query to its "logical form", e.g., a SPARQL query
    → work by Elmar, Niklas, Thomas (Aqqu) + hopefully **more of you**
  - Our work on the QLever UI is relevant for that in two respects

    1. To be able to translate a question to a SPARQL query, we need to understand the challenges and subtleties of SPARQL queries

    2. An intelligent and functioning SPARQL autocompletion is actually a big step towards question answering, for the following reason:

       - A good autocompletion allows the construction of a SPARQL query with minimal user input and little and simple user interaction
       - A question can be seen as minimal user input, so all that remains is to automate the user interaction part
  - More generally, understanding the intricacies of knowledge bases and SPARQL helps to deeper understand **natural language processing**

Other projects in our group 2/4

UNI FREIBURG

- Combination with text search
  - QLever is unique for its capabilities to search in text combined with a knowledge base; here are some sub-problems:

    Entity Linking: To combine text with a knowledge base, we need to recognize and disambiguate entity mentions (also co-references like "he", "she", "it", "the film") in the text → work by Matthias and Natalie

    Text extraction from PDF: Much text is available in the form of PDFs, but it is very hard to extract text (and semantic information like what is a header) from PDFs; this has nothing to do with the particular PDF standard, it's an inherent problem → talk by Claudius in November

    Error correction: Errors and wrong spacing are a big issue, for any text produced by humans and also for text extracted from PDFs; if this is not fixed, we get zero results → work by Markus, Matthias, Mostafa

    Note that error correction is not only relevant for text search, but also for SPARQL queries, question answering, and autocompletion!

Other projects in our group 3/4

- Efficiency
  - Interacting with a knowledge base is no fun (or infeasible) if it takes forever to build an index or ask queries

    QLever query engine: Fast index construction, basic architecture, query planning, efficient support of the many SPARQL features
    → work by Björn, Niklas, Florian (Kramer), Johannes

    Autocompletion: Suggestions are only fun when they are very fast
    → work by Niklas, Johannes, Theresa
- User Interfaces
  - Important to easily interact with the data and understand it, which is the foundation of much of our other work + as mentioned earlier, an important step towards question answering
    → work by Florian (Bäurle), Björn, Elmar, Johannes, Daniel (Kemen), Julian (Bürklin), Simon (Selg), Natalie, Theresa

    More about the QLever UI in the third part of this talk

Other projects in our group 4/4

- Domain-specific knowledge bases
  - Building knowledge bases for each particular domain has its own challenges, especially when the knowledge bases are huge

    Freebase and Clueweb: this was a huge challenge at the time (3.1B triples, 23.4B words) → work by Elmar and Björn

    Wikidata: it took us a year to enable QLever to index the complete Wikidata (13B triples) in less than a day without using huge amounts of RAM → work by Niklas and Johannes

    OpenStreetMap: ongoing project with its own challenges because of the huge number of objects and the geometrical data
    → work by Axel and Patrick (our OSM expert)

    Publication data / DBLP: our first search engine in 2006 still powers the DBLP publication search; in the meantime, there is also an RDF dump of the data → see end of the talk

The QLever UI 1/7

- Goal
  - Provide interactive suggestions in order to write a SPARQL query with the desired result as **effortlessly** and as **quickly** as possible
- Realization
  - When we started this work, we constructed all kinds of auxiliary data structures to make these suggestions sufficiently fast
  - Eventually, we realized that **all** suggestion queries could be formulated as SPARQL queries and instead of building special-purpose data structures, we improved QLever to process the Autocompletion SPARQL queries efficiently

    As a nice side effect, many useful features (that were missing so far) were added to QLever and many operations were sped up significantly

    Two birds with one stone: no need for a separate autocompletion engine and a better SPARQL engine

The QLever UI 2/7

- SPARQL autocompletion via SPARQL 1/3
  - For example, assume that we have typed the following incomplete query and are looking for suggestions for **objects** (cursor at _)

```
SELECT ... WHERE {
  wd:Q873   wdt:P166  A_      # Meryl Streep   award won
}
```

  - Then we can get these suggestions via the following SPARQL query

```
SELECT ?object ?name ?score WHERE {
  wd:Q873     wdt:P166                          ?object       .
  ?object     skos:altLabel|rdfs:label          ?name         .
  ?object     ^schema:about/wikibase:sitelinks  ?score        .
  FILTER REGEX(?name, "^A")
} ORDER BY DESC(?score)
```

Recall: the | operator takes the union of two predicates; the ^ operator reverses a predicate; the / operator expresses a "path" via an intermediate object which we don't care about

The QLever UI 3/7

- SPARQL autocompletion via SPARQL 2/3
  - Now assume that we have typed the following incomplete query and are looking for suggestions for **predicates** (cursor at _)

```
SELECT ... WHERE {
  wd:Q873   p:P166   ?s .        # Meryl Streep  award won
  ?s          _
}
```

  - Then we can get these suggestions via the following SPARQL query

```
SELECT ?predicate ?name ?score WHERE {
  { SELECT ?predicate (COUNT(?object_tmp) AS ?score) WHERE {
      wd:Q873      p:P166        ?s                .
      ?s           ?predicate    ?object_tmp       .
  } GROUP BY ?predicate }
  ?meta_tmp     ?connect_tmp     ?predicate     .
  ?meta_tmp     rdfs:label       ?name          .
} ORDER BY DESC(?score)
```

The QLever UI 4/7

- SPARQL autocompletion via SPARQL 3/3
  - Now assume that we have typed the following incomplete query and are looking for suggestions for **predicates** (cursor at _)

```
SELECT ... WHERE {
  ?x          _
}
```

  - Then we can get these suggestions via the following SPARQL query

```
SELECT ?predicate ?name ?score WHERE {
  { SELECT ?predicate (COUNT(?object_tmp) AS ?score) WHERE {
      ?x          ?predicate    ?object_tmp    .
  } GROUP BY ?predicate }
  ?meta_tmp    ?connect_tmp    ?predicate    .
  ?meta_tmp    rdfs:label          ?name         .
} ORDER BY DESC(?score)
```

The **?x ?predicate ?object_tmp** query looks like we have to scan the whole index, that looks infeasible

The QLever UI 5/7

- Challenges and progress over time
  - Provide all suggestions via SPARQL — example previous slide
  - Use names and aliases from the knowledge base — example previous slide
  - Ranking via information from the knowledge base — example previous slide
  - Context-sensitive suggestions
  - Essential SPARQL features added or made efficient — talk on its own
  - Efficient support for ql:has-predicate — talk on its own
  - Caching and memory efficiency and awareness — talk on its own
  - Sophisticated evaluation benchmarks and suite — talk on its own
  - Template mechanism for suggestion queries
  - Suggestion of reverse predicates
  - Timeout with fallback to context-agnostic suggestions — inside Qlever UI?
  - Automatic addition of name triples and select variables
  - Map UI ... with recursive clustering for large result sets — talk on its own

The QLever UI 6/7

- What is still missing?
  - Despite all the great progress, for some queries, it is still hard to guess how certain pieces of information are represented in the knowledge base:

    Transitivity: To get all entities of a certain type, we often need constructs such as wdt:P31/wdt:P279* ... e.g.: all cities

    It would be great if the UI could automatically realize that such a predicate path is appropriate or needed ... but how?

    One-hop connections: Sometimes, finding the predicate to get to the right statement node can be very unintuitive ... e.g.: members Bundestag

    It would be great if we could specify the kind of entity we want to **reach** and than the UI suggests how to get there ... but how?

    Clumsy SPARQL constructs: several intuitive notions are very clumsy to express in SPARQL; in particular, SPARQL queries that require a GROUP BY are always a pain ... e.g.: Bundestag members and their party

    Idea: context-sensitive suggestions of whole constructs (not just components of a triple) for extending a query

The QLever UI 7/7

- Let's do some live queries together
  - Think of some interesting list or table of information that can be extracted from Wikidata or OpenStreetMap

    With or without relation to a map, as you like
  - Then let us try to find the correct SPARQL query using the Qlever UI