PS 703301 – WS 2021/22
Current Topics in Computer Science

Final Report

Shaping Knowledge Graphs

Philipp Gritsch
Jamie Hochrainer
Kristina Magnussen
Danielle McKenney
Valerian Wintner

supervised by
M.Sc. Elwin Huaman

Contents

remove this part

I added a comment colour for everyone. ***¡Comment colour Jamie¿*** ***¡Comment colour Danielle¿*** ***¡Comment colour Philipp¿*** ***¡Comment colour Valerian¿*** ***¡Comment colour Kristina¿***

1. Introduction

add introduc tion to KGs

We used *CommonCrawl* ***¡maybe too specific as first sentence in introduction¿*** ***¡Agreed, I think the introduction to KGs should come before this¿*** datasets as the base for the *knowledge graph* which we wanted to assess. The data contained in those datasets is often inconsistent and might contain errors. In order to work with this data properly, it is necessary to shape the *knowledge graph* in which this data is contained. This shaping is done by inferring constraints over the data and validating it based on these constraints. Validating a graph against constraints gives important insight into the structure of the data. For instance, when all nodes of a type conform to constraints, then it may be useful to define these as required attributes for all future nodes to ensure uniformity in the data. Non conforming nodes may also deliver important insight into where information is missing. For example, if 99% of nodes of a given type conform to some constraints, it may be worthwhile to investigate the remaining 1% to see if they are missing necessary information or are otherwise corrupt.
¡Introduction should also contain what we cover in the report (not only motivation) with referring to Sections to give a short overview.¿ ***¡Agreed. I wrote something like this (see uncommented text on top of introduction), but I put it in approach. Should we put it back into introduction? Or should we write something new for introduction and leave the other text in approach?¿***

2. Related Work

Add thesis Werkmeister RDF2Graph. also add an- other work, maybe from sources in th sis, done by Philipp

3. Approach

Our framework ***¡do you mean web application? I thought framework is sth different¿*** ***¡I'm talking about the whole application here, not only the web application. I usually used framework when I was talking about our entire project, since this is also the term Elwin used, but we could also maybe change this to application or program?¿*** offers a way to evaluate a *knowledge graph* in an automated way. For this, we used *knowledge graphs* from the *CommonCrawl* datasets as a basis. The *knowledge graphs* are imported as a static file. After this, our framework infers constraints over this data set (see Section 3.2). These are validated automatically in the last step, see Section 3.3. The user can interact with this framework over the front-end, see Section 3.4. These different steps were implemented and tested separately. Once this was done, we consolidated them. The structure of our project can be seen in Fig. 1.

update figure

3.1. Technology Stack

refer to the readme here? Or should th happen some where else?

add reference to our github repo!

¡In general, it would be nice to have an introductory sentence at the beginning of each section¿ The framework was implemented in *Java*. We used *Maven* as a project management tool. We also used *Jena*, which offers an *RDF* API as well as support for *SPARQL* queries and the *ShEx* language. The front-end was implemented using *Vue3*[1] as a front-end framework and *PrimeVue* as a library for the different UI components. For the deployment of our application we used a single virtual machine. Access to the front-end is done via a single *Apache* server. The front-end accesses the back-end via an internal *REST-API*.

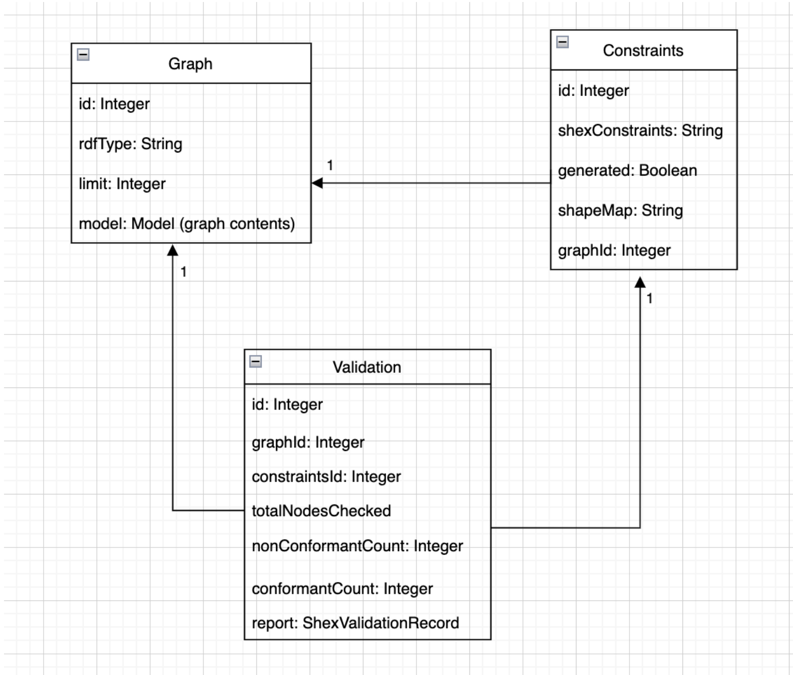


Figure 1: UML diagram of the framework structure

¡Missing: Subsection about generating subgraph (with limit), starting from a certain type of node.¿ ***¡Wouldn't this be part of Generating constraints? I feel like that doesn't really fit into technology stack¿***

3.2. Generating Constraints

For the generation of constraints, we used the adaption of the tool *RDF2Graph*[3] by Werkmeister[4][5]. ***¡(this is the old version, we use the fork)¿*** ***¡I adapted the sentence, do you think this works now, Valerian?¿*** and adapted it for our purposes. As input, *RDF2Graph* takes a *knowledge graph* from *CommonCrawl*. The properties of the graph are read out with several *SPARQL* queries. These properties are saved in a new *RDF* graph. As output, we receive a graph containing constraints for the initial input data. We use a tool offered by *RDF2Graph* to extract the constraints in *ShEx* syntax. ***¡RDF2Graph offers a tool to export the constraints to ShEx syntax.¿*** ***¡I adapted the sentence, do you think this is okay now, Valerian?¿***

Missing figure

add query to graph (chosen by Philipp), e.g. multiplicy of argument etc.

3.2.1. Integrating RDF2Graph with our framework

We implemented the following steps in order to integrate *RDF2Graph* into our project. We added *RDF2graph* to our framework so that they could be compiled together***¡, and in the process minimally updated it to be compatible with our version of Java and Jena¿***. ***¡I would leave out the minimally, no need to downplay the work you did on this here, I think. Apart from that, I like it, feel free to put it in¿*** In addition, we changed some of the initial parameters of the *RDF2Graph*, since it originally was intended as a stand-alone application. As we are handling *Models* in our software, we changed the input to *RDF2Graph* to a *Model*. In our application, *RDF2Graph* does not use any other storage apart from the *Model* data structure. Previously, such a Model needed to be created by *RDF2Graph*, now it is provided by our framework. We did this so we could have full control over the files handled by *RDF2Graph*. *RDF2Graph* allows for multithreaded execution, which requires a thread pool. This thread pool was initially created by *RDF2Graph*. In our framework, it is provided by our application. In addition, resources which are used by *RDF2Graph* had to be provided in a different way so that they are still available when running from a server environment. We also changed some of the queries. *RDF2Graph* supports multiple output graphs, however, this did not work . As we only work on one Model at a time, we only use one output graph.

Add expla-
nation for
Model? May
in glossary?

should we ex
plain this in
more detail?

Add explana
tion of limit
this section?

3.3. Validating Constraints

Given a *RDF* graph and a set of constraints, the validation consists of verifying that every node in the graph fulfils the requirements given in the constraints. A graph consists of several different types. Each of those types must conform to its definition outlined in the constraints. The results of the validation output a boolean flag for every single node in the graph, indicating whether or not it conforms to its type's constraints. In case of nonconformity, a reason will be given.

In our code, this is implemented in the following way. As input, we receive a *RDF* subgraph as well as a set of constraints. We use this to generate a *shape map*, which contains all of the types which need to be validated. For the actual validation, the *ShExValidator* provided by the *Jena* library was used. The validator requires a set of constraints defined in valid *ShEx* syntax and a *shape map*. The *shape map* describes which types of nodes need to be validated against which *ShEx* constraint definitions.

add reference
to Jena libra
here?

add picture/code snipped of shapeMap

The class *ShexValidationRecord* stores the result of the validation for every single node of the graph. Not only is the individual result of every node checked against its relevant constraints, but we also calculate the percentage of nodes that conform to their constraints.

3.4. Front-end

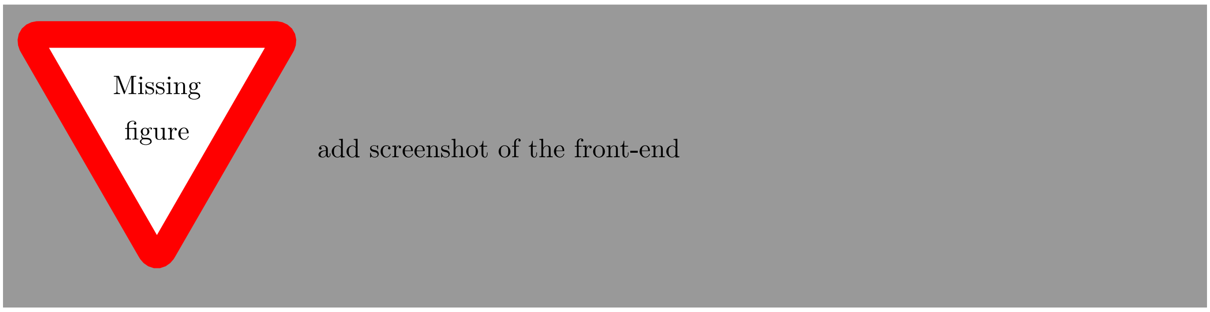
We implemented a front-end where the user can choose a *knowledge graph* as well as a type of knowledge graph and its type. In addition, the user can also set a limit. As output, *ShEx* constraints as well as a validation of those constraints are given. The constraints can be edited by the user and those edited constraints can be revalidated. If a node is deemed invalid, a reason is given, e.g. "Cardinality violation

check whethe
this is what
we are doing
in the finishe
version

explain this
in more de-
tail, maybe
also put the
explanation i
query

(min=1): 0". The user can download the subset of the graph which was validated. The interaction between user, front-end and server can also be seen in Fig. 2.

explain how different limi influences da output



update and scale sequenc diagram and refer to it

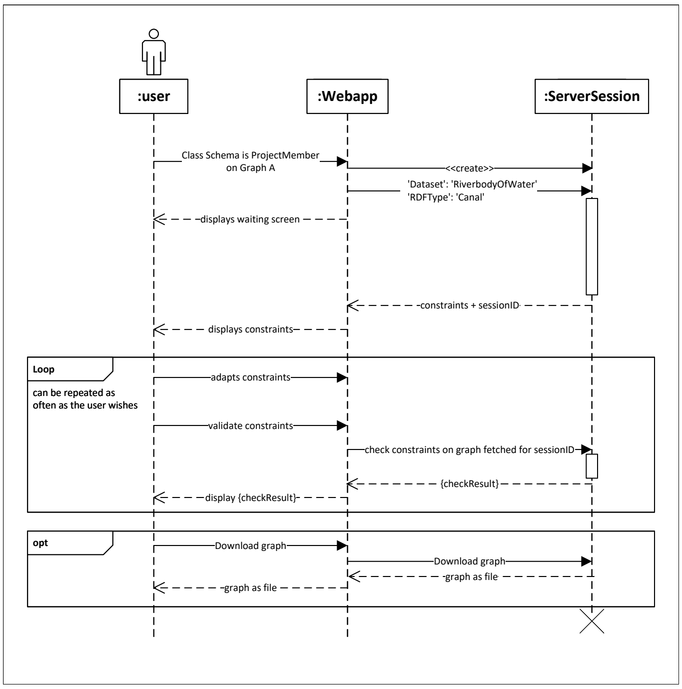


Figure 2: Sequence diagram showing the interaction between web application, user and server

4. Results

Our framework automatically infers constraints and validates the given data based on those constraints. This can be done on two different *CommonCrawl* datasets. The user can choose one of those datasets and a limit using the front-end. User can also edit constraints.

explain this limit in more depth, mayb in front-end?

Maybe add small figure that shows workflow of project here? Something similar like we did in presentation but more professional?

describe results of bench-mark tests here

5. Evaluation

add bench-marks here

5.1. Future work

check what E win said con-cerning Evalu ation on meet ing 20.01.202

Possible futu work could be: more dat sets, more po sibilities for user inputs

Our application currently only handles two different datasets. For future work, this could be expanded so that the framework could handle more and bigger datasets. Currently, the size of the datasets that can be handled is limited by the RAM on the virtual machine. One possible solution for this could be to only work on parts of the graph. One problem we encountered when handling datasets from *CommonCrawl* was the quality of these datasets. Many datasets include *non-unicode* characters, which are replaced by Jena with *unicode* characters. This takes a lot of computing time. In addition, many files include invalid *RDF* syntax or are otherwise damaged. This means that in order to handle additional datasets, some way of processing these datasets would have to be implemented. Processing could include filtering for broken files and invalid syntax and fixing this before handling the dataset in the framework. In addition, more possibilities for user interaction could be added. For instance, a feature could be added where a user can upload their own dataset and have it validated.

Should we add proper SPARQL end points here? Might not be possible?

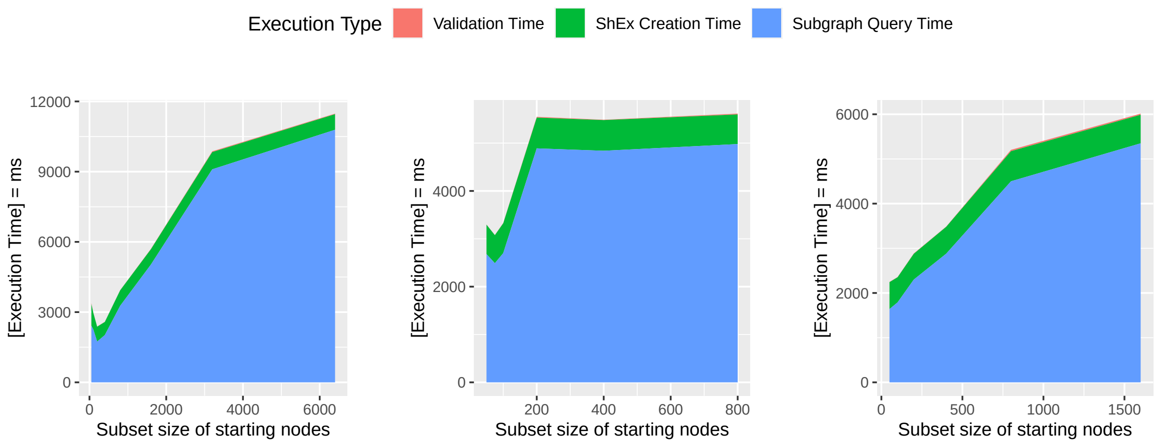
5.2. Methodology

¡i labeled all the included graphics with h!, when we have finished the report we might want to make it so, that one image is on the top and one on the bottom, if 2 pages are on the same page, for example¿ For taking the measurements, the application was started locally on our hardware,***¡I would put a full stop here and maybe start the next sentence like "This was done to minimise..¿*** to minimize side-effects of other applications running on the virtual machine where the live-instance is deployed. The JVM was additionally setup ***¡Additionally, the JVM was set up..?¿*** to use up to 16 GB of main memory for its heap to allow parallel queries without compromising the runtime of the executions, arising from extensive swap usage. ***¡This sentence is very long, maybe we can split it somehow?¿***

5.3. Runtime

Figures 3 and 4 show our ***¡the?¿*** measurements we obtained by changing the *LIMIT* input parameter. This parameter limits the size of the start-node subset, from which connected nodes are queried. All the measurements are shown in Tables 1 and 2.

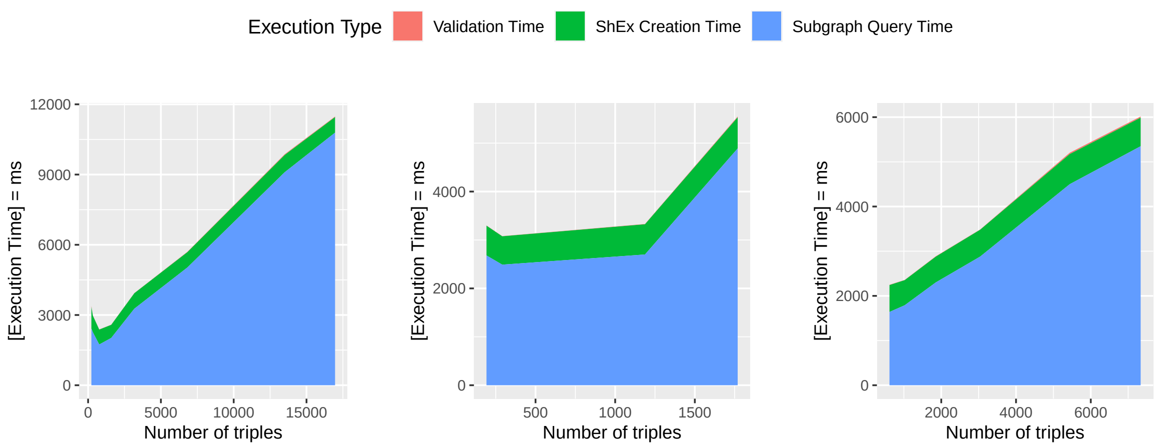
The results shown in Figure 3 were to be expected. First of all, the runtime of constructing the desired subset of the graph is considerably larger than the time needed to create the *ShEx* constraints, or to validate the constraints on the graph. Secondly, the runtime of constructing the subgraph scales with the *LIMIT*. This becomes especially evident in Figures 3a and 3c.



(a) RDFType = Canal (b) RDFType = RiverBodyOfWater (c) RDFType = Service

Figure 3: Execution times per RDFType, per size of start-node subset on RiverBodyOfWater dataset

To understand the behaviour shown in Figure 3b, we want to look at Figure 4, which shows the same runtimes, but grouped by the number of triples in the subgraph, on which the constraints are created. Unlike in Figures 4a and 4c the maximum number of triples (shown in the x-coordinate in Figure 4b), is 1769. This is also the amrefer to figures paperount ***¡What does this mean?¿*** of triples contained in the subgraph that we get without providing any limit. Therefore, providing a limit larger than 200 won't enrich the constructed graph, keeping the time almost constant in regards to the *LIMIT* parameter.



(a) RDFType = Canal (b) RDFType = RiverBodyOfWater (c) RDFType = Service

Figure 4: Execution times per RDFType, per number of triples on RiverBodyOfWater dataset

Figure 5 shows the runtime without limiting the construction of the subgraph. Note the much larger runtime needed for querying the graph, despite resulting in the same amount of triples when providing a large enough *LIMIT*.

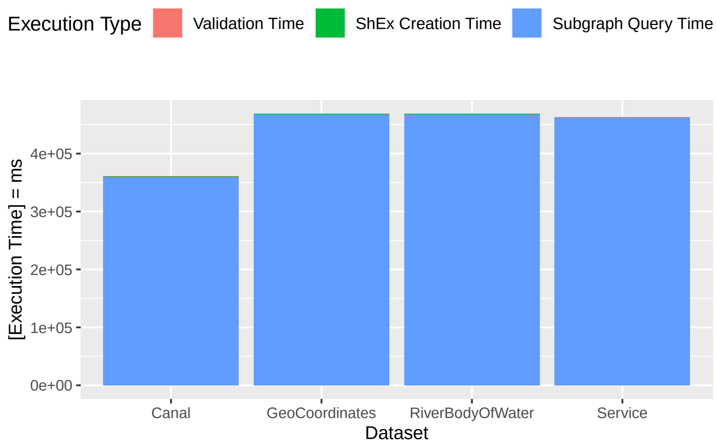


Figure 5: Execution times per RDFType of the RiverBodyOfWater dataset (containing 49915 triples)

5.4. Correctness

5.4.1. ShEx Generation

We thought *Shexer*, which was already mentioned in Section 2, was a good fit for cross validating our *ShEx*-generation. However, due to our limited knowledge of operating this tool, we did not manage to generate proper constraints for our RiverBodyOfWater-dataset. Our attempt at using this tool is shown in Figure 7, which generated the trivial, non-restrictive constraints shown in Figure 8.

Therefore, we checked the generated constraints manually for small subgraphs (see Figures 9, 10 and 11) and identified two issues with our tool.

Firstly, if the dataset consists of only stand-alone blank nodes, as seen in Figure 6, then *Rdf2Graph* does not infer any *ShEx*-constraints. This was the case for the generated subgraph using RiverBodyOfWater with a *LIMIT* of 50, and the resulting *ShEx* can be seen in Figure 10.

Secondly, optional properties are not always inferred and therefore missing from the generated *ShEx*-constraints. This also happens for unlimited subgraphs (see Figures 12, 13 and 14), with the exception of the RiverBodyOfWater-RDFtype, where it looks like the constraints are complete, ***¡Would put a full stop here and start new sentence "However,..."¿*** however due to the large graph manually checking for correctness is infeasible. We did not see a correlation between missing constraint-properties and the shape of the graph.

5.4.2. ShEx Validation

The generated *ShEx*-constraints for small subgraphs (Canal with *LIMIT* 50, Service with *LIMIT* 50) were cross validated using the online-tool *RDFShape*[2]. The validation result was the same as in our tool.

6. Conclusion

Which challenges did we face during the implementation? (Maybe depth of SPARQL query, outdated RDF2Graph?

Did we achieve what we wanted to do? How well and

```
[ a       <https://schema.org/Service> ;
  <https://schema.org/serviceType>
          "Marine Electrics"@en
] .

[ a                          <https://schema.org/Service> ;
  <https://schema.org/description>
          "A list of locations with dry dock facilities on the Main Canal of the Trent &
     Mersey Canal"@en ;
  <https://schema.org/serviceType>
          "Dry Dock"@en ;
  <https://schema.org/url>  <https://www.ukwaterwaysguide.co.uk/s/trent-mersey-canal/
    main-canal/dry-dock>
] .
```

Figure 6: Blank Nodes in Turtle File

References

[1] Vue.js, documentation. `https://v3.vuejs.org/`.

[2] J. E. L. Gayo. Rdfshape. `https://rdfshape.herokuapp.com/validate`, 2021. Accessed: 2022-01-23.

[3] J. van Dam. Rdf2graph. `https://github.com/jessevdam/RDF2Graph/`, 2022. Accessed: 2022-01-16.

[4] L. Werkmeister. Schema inference on wikidata. Master's thesis, Karlsruher Institut für Technologie, Fakultät für Informatik, 2018.

[5] L. WerkMeister. Rdf2graph. `https://github.com/lucaswerkmeister/RDF2Graph`, 2022. Accessed: 2022-01-16.

A. Contribution Statements

Please write down a short contribution statement for each member of your group. You may evaluate the contribution along the three common categories: i) conception (i. e., problem framing, ideation, validation, and method selection), ii) operational work (e. g., setting up your tech stack, algorithm implementation, data analysis, and interpretation), and iii) writing & reporting (i. e., report drafting, literature review, revision of comments, presentation preparations, etc.).

B. Appendix

You may use appendices to include any auxiliary results you would like to share, however cannot insert in the main text due to the page limit.

B.1. Code listings and additional data

```python
from shexer.shaper import Shaper
from shexer.consts import NT, SHEXC

namespaces_dict = {
    "http://www.w3.org/2001/XMLSchema#": "xsd",
    "http://www.w3.org/1999/02/22-rdf-syntax-ns#": "rdf",
    "http://www.w3.org/2000/01/rdf-schema#": "rdfs",
    "http://www.w3.org/2004/02/skos/core#": "skos",
    "http://schema.org/": "schema"
}

input_file = "rbow.nt"

shaper = Shaper(
    graph_file_input=input_file,
    all_classes_mode=True,
    input_format=NT,
    remove_empty_shapes=False,
    discard_useless_constraints_with_positive_closure=False,
    depth_for_building_subgraph=100,
    inverse_paths=True,
    shapes_namespace="http://schema.org/",
    all_instances_are_compliant_mode=False,
    namespaces_dict=namespaces_dict,
    instantiation_property="http://www.w3.org/1999/02/22-rdf-syntax-ns#type")  # Default
     rdf:type

output_file = "shexer_rbow.shex"

shaper.shex_graph(output_file=output_file,
                  output_format=SHEXC,
                  acceptance_threshold=.1)
```

Figure 7: Running shexer on the full graph

```
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
PREFIX : <http://schema.org/>

:PropertyValue
{
}


:RiverBodyOfWater
{
}


:Hotel
{
}

# ... many more empty terms
```

Figure 8: Shexer output

```
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
PREFIX schema: <http://schema.org/>

<https://schema.org/Canal> {
  <https://schema.org/description> rdf:langString?;
  <https://schema.org/name> rdf:langString?;
  #<https://schema.org/url> .? # <-- missing from generated ShEx.
}

<https://schema.org/RiverBodyOfWater> {
  <https://schema.org/description> rdf:langString;
  <https://schema.org/name> rdf:langString;
  <https://schema.org/url> .
}

<https://schema.org/Service> {
  <https://schema.org/description> rdf:langString?;
  <https://schema.org/serviceType> rdf:langString;
  <https://schema.org/url> .?
}
```

Figure 9: Generated *ShEx*-constraints of Canal with *LIMIT* 50

```
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
PREFIX schema: <http://schema.org/>
```

Figure 10: Generated *ShEx*-constraints of RiverBodyOfWater with *LIMIT* 50

```
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
PREFIX schema: <http://schema.org/>

<https://schema.org/Canal> {
  <https://schema.org/description> rdf:langString?;
  <https://schema.org/name> rdf:langString?;
  #<https://schema.org/url> .? # <-- missing from generated ShEx.
}

<https://schema.org/RiverBodyOfWater> {
  <https://schema.org/description> rdf:langString;
  <https://schema.org/name> rdf:langString;
  <https://schema.org/url> .
}

<https://schema.org/Service> {
  <https://schema.org/description> rdf:langString?;
  <https://schema.org/serviceType> rdf:langString;
  #<https://schema.org/url> .? # <-- missing from generated ShEx.
}
```

Figure 11: Generated *ShEx*-constraints of Service with *LIMIT* 50

```
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
PREFIX schema: <http://schema.org/>

<https://schema.org/Canal> {
  <https://schema.org/address> rdf:langString?; # <-- Only 1 standalone blank node with
    this property
  <https://schema.org/description> rdf:langString?;
  <https://schema.org/name> rdf:langString?;
  #<https://schema.org/url> .? # <-- missing from generated ShEx.
}

<https://schema.org/RiverBodyOfWater> {
  <https://schema.org/description> rdf:langString;
  <https://schema.org/name> rdf:langString;
  <https://schema.org/url> .
}

<https://schema.org/Service> {
  <https://schema.org/description> rdf:langString?;
  <https://schema.org/serviceType> rdf:langString;
  #<https://schema.org/url> .? # <-- missing from generated ShEx.
}
```

Figure 12: Generated *ShEx*-constraints of Canal without a *LIMIT*

Rdftype	Triples	$[t_{graph}] = \text{ms}$	$[t_{shex}] = \text{ms}$	$[t_{validation}] = \text{ms}$
Canal	16961	360000	737	45
GeoCoordinates	204	468000	585	4
RiverBodyOfWater	1769	468000	613	15
Service	7334	462000	618	19

Table 1: Execution times per RDF-Type, queried on full graph of the RiverBodyOfWater dataset (containing 49915 triples)

```
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
PREFIX schema: <http://schema.org/>

<https://schema.org/AdministrativeArea> {
  <https://schema.org/name> rdf:langString;
  <https://schema.org/url> .
}

<https://schema.org/Canal> {
  <https://schema.org/description> rdf:langString?;
  <https://schema.org/name> rdf:langString?;
  <https://schema.org/url> .?
}

<https://schema.org/EducationalOrganization> {
  <https://schema.org/name> rdf:langString;
  <https://schema.org/url> .
}

<https://schema.org/GeoCoordinates> {
  <https://schema.org/elevation> rdf:langString?;
  <https://schema.org/latitude> rdf:langString?;
  <https://schema.org/longitude> rdf:langString?
}

<https://schema.org/Map> {
  <https://schema.org/sameAs> .
}

<https://schema.org/Place> {
  (
    <https://schema.org/containsPlace> @<https://schema.org/AdministrativeArea>* |
    <https://schema.org/containsPlace> @<https://schema.org/EducationalOrganization>?
  );
  <https://schema.org/name> rdf:langString?;
  (
    <https://schema.org/url> .? |
    <https://schema.org/url> rdf:langString?
  )
}

<https://schema.org/RiverBodyOfWater> {
  <https://schema.org/address> rdf:langString?;
  <https://schema.org/alternateName> rdf:langString*;
  <https://schema.org/containedInPlace> @<https://schema.org/Place>*;
  <https://schema.org/description> rdf:langString*;
  <https://schema.org/geo> @<https://schema.org/GeoCoordinates>*;
  <https://schema.org/hasMap> @<https://schema.org/Map>*;
  <https://schema.org/image> rdf:langString*;
  <https://schema.org/name> rdf:langString+;
  <https://schema.org/sameAs> .*
}

<https://schema.org/Service> {
  <https://schema.org/description> rdf:langString?;
  <https://schema.org/serviceType> rdf:langString;
  <https://schema.org/url> .?
}
```

Figure 13: Generated *ShEx*-constraints of RiverBodyOfWater without a *LIMIT*

```
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
PREFIX schema: <http://schema.org/>

<https://schema.org/Canal> {
  <https://schema.org/description> rdf:langString?;
  <https://schema.org/name> rdf:langString?;
  #<https://schema.org/url> .? <-- Missing from generated ShEx
}

<https://schema.org/RiverBodyOfWater> {
  <https://schema.org/description> rdf:langString;
  <https://schema.org/name> rdf:langString;
  <https://schema.org/url> .
}

<https://schema.org/Service> {
  <https://schema.org/description> rdf:langString?;
  <https://schema.org/serviceType> rdf:langString;
  #<https://schema.org/url> .? # <-- Missing from generated ShEx
}
```

Figure 14: Generated *ShEx*-constraints of Service without a *LIMIT*

Rdftype	Limit	Triples	$[t_{graph}] = \text{ms}$	$[t_{shex}] = \text{ms}$	$[t_{validation}] = \text{ms}$
Canal	50	226	2420	923	44
Canal	100	328	2260	709	13
Canal	200	765	1740	637	8
Canal	400	1588	2020	559	9
Canal	800	3176	3270	661	8
Canal	1600	6817	5030	654	17
Canal	3200	13504	9100	736	33
Canal	6400	16961	10790	665	25
RiverBodyOfWater	50	192	2680	615	5
RiverBodyOfWater	75	291	2490	586	4
RiverBodyOfWater	100	1187	2700	624	7
RiverBodyOfWater	200	1769	4890	643	17
RiverBodyOfWater	400	1769	4840	641	8
RiverBodyOfWater	800	1769	4980	619	18
Service	50	615	1640	602	7
Service	100	1022	1790	562	6
Service	200	1852	2300	577	9
Service	400	3041	2880	601	6
Service	800	5437	4500	674	30
Service	1600	7334	5350	639	26

Table 2: Execution times per RDF-Type, limited size of start-node subset (using the RiverBodyOfWater dataset)