BioHackrXiv

BioHackJP 2023 Report R4: RDF Data integration using Shape Expressions

Jose Emilio Labra Gayo[1], Andra Waagmeester[2], Yasunori Yamamoto[3], Ángel Iglesias Préstamo[1], Toshiaki Katayama[3], Thomas Liener[4], Deepak R. Unni[6], Jerven Bolleman[6], Kiyoko F. Aoki-Kinoshita[7], Masashi Yokochi[8], Núria Queralt Rosinach[9], Hiroshi Mori[10], Daniel Fernández Álvarez[1], Alberto Labarga[11], Jose Alberto Benítez Andrades[1], Robert Hoehndorf[13], Eric Prud'hommeaux[14], Claude Nanjo[15], Nishad Thalhath[12], and Yoko Okabeppu[16]

1 WESO Lab, University of Oviedo, Spain **2** Micelio, Belgium **3** Database Center for Life Science, Japan **4** Unaffiliated **6** SIB Swiss Institute of Bioinformatics, Switzerland **7** Soka University, Hachioji, Tokyo, Japan **8** Osaka University, Suita, Osaka, Japan **9** Leiden University Medical Center, Netherlands **10** National Institute of Genetics, Mishima, Japan **11** Barcelona Supercomputing Center **12** RIKEN Center for Integrative Medical Sciences, Yokohama, JP **13** Janeiro Digital, USA **14** MedOntology, LLC, USA **15** OKBP inc. Japan

BioHackathon series:
DBCLS BioHackathon 2023
Kagawa, Japan, 2023
R4

Submitted: 06 Nov 2024

Published by BioHackrXiv.org

Background

In this report, we describe the activities that we have been carrying on during the Biohackathon 2023, held in Shodoshima, Japan. The main goal of the project has been to identify approaches and issues that can be used to integrate large RDF datasets by creating subsets described by Shape Expressions (Prud'hommeaux et al., 2014).

We have recently submitted a publication on creating subsets from Wikidata (Hosseini, 2023). Wikidata is a knowledge graph which is constantly in flux and getting to a size which makes it hard to locally replicate. By creating topical subsets we are able to dissect a managable subset that can be loaded in local RDF stores for further processing.

However, this subsetting app approach relies on Wikidata daily dumps, which are available in JSON format. For this hackathon we specifically choose to extend the subsetting mechanisms to work on RDF dumps or SPARQL endpoints.

RDF data provides a solution for data interoperability which in principle can enable different data sources to be smoothly integrated. Nevertheless, in practice, the growing adoption of RDF has also made that the consumption of RDF data is challenging given the size of RDF data collections makes them not feasible to be easily collected or manipulated. As an example, UniProt RDF data size is around 110 billions of triples or over 700 gigabytes of gzipped RDF/XML, and PubChem RDF is in the same order of magnitude in volume, both these resources describe 100's of different kinds of data. This situation requires some agents to provide intermediate resources to manipulate the RDF data which is consumed, and described. As an example, DBCLS has created the rdfconfig tool which provides descriptions of the RDF data collections they contain.

In order to facilitate the integration of those RDF data, this project has been exploring ways to create subsets of RDF data which could contain only the information of interest for some specific tasks. Making those subsets available for researchers in an easy way, could facilitate the research activities and give the RDF data more value.

Creating subsets of RDF data can also help when the information available in those large RDF data sources is continually evolving. So if a researcher wants to make reproducible research

based on those SPARQL endpoints, it may be possible that the results of the queries can differ along the time.

The possibility of having a tool that creates subsets can be considered as a way to create snapshots of the RDF data which could later be packaged and distributed along the research results, helping the creation of reproducible research based on RDF data.

Another reason for the creation of RDF subsets is to make a subset from multiple data sources where it is unfeasible to get a designated dataset using federated queries due to these data sizes and technical immaturity of SPARQL federated query handling.

The Shape Expressions language was designed as a concise and human-readable language to describe and validate RDF data. Although it was not initially designed to create subsets, the possibility of having a concise way to describe RDF makes it an ideal candidate for this task.

Indeed, ShEx has already been used to create subsets of Wikidata. The following paper contains a list of different subseting approaches which have been used in Wikidata: https://www.semantic-web-journal.net/content/wikidata-subsetting-approaches-tools-and-evaluation-0.

The main goals of this project have been to review the different approaches to create subsets of large RDF data in order to facilitate the integration of different RDF collections.

We identified the participation of 3 main agents:

- Data producers or providers who are interested in produce RDF data that can have more value and be used by third parties.
- Data consumers who are interested in an easy way to get access to the RDF data available in those sources and to create reproducible workflows which can contain manageable RDF subsets.
- Data integrators who can help in the intermediate process of bringing over the RDF data produced by the providers to the end consumers. One important aspect to take into account is that the RDF data may need to be transformed with actions like changing URI prefix declarations or manipulating the topology of the RDF data. Another aspect is the need to understand the structure of the RDF data collected and to agree in a common structure. For this, Shape expressions also offer the possibility to validate the RDF data that is produced and the RDF data that may be consumed. Avoiding the need of defensive programming techniques.

Outcomes

Creation of ShEx based subsets

One important aspect of data integration is the possibility of creating small subsets from large RDF data. During the biohackathon we explored several use cases taking data from UniProt, PubChem, TogoGenome, etc.

We created a github repository called subsetting-examples that contains some example SPARQL queries, ShEx schemas and scripts used during the biohackathon. From those, two of the example queries were rewritten in such a way that is accepted by the `pschema-rs` validator. More into this will be discussed later on.

The use cases that we explored are the following:

UniProt subset based on proteins

A subset of proteins in UniProt are glycosylated; these are known as glycoproteins. We wanted to extract all glycoproteins from UniProt by taking those entries that had Glycosylation Annotations.

We did a first experiment taking as input RDF dumps from UniProt. Kiyoko provided a simple SPARQL query that obtains proteins and annotations. Jose Labra converted the SPARQL query to ShEx. That ShEx schema was used as input in the PSChema tool that has been developed by Ángel Iglesias in Rust and we already created a subset that was later publishedinZenodo with its proper doi.

UniProt subset based on Subcellular locations

Yasunori provided an example SPARQL query about Subcellular locations that was later converted by Jose Labra to ShEx in order to create subsets from UniProt. This query is to obtain UniProt annotations based on an amino-acid position of a specific protein, especially to know whether an amino-acid residue is located in a cytoplasmic or a transmembrane domain. The subsets were generated by Ángel Iglesias using the PSChema Rust tool which was interesting as we discovered a bug that required an update of the tool. In the following section we will comment on the improvements that led to the results obtained.

The results obtained from the experiment

It is worth mentioning that we have used the `uniprotkb_reviewed_viruses_10239_0.rdf` dump (with a size of 1.13GB) from the Uniprot Knowledge graph to create the subsets from. As it is serialized in a RDF/XML format, a tiny pipeline for converting it into N-Triples, the serialization format accepted by the tool, is required. The script that automatically processes the compressed dump can be seen in the examples section of the tool's repository.

With that being said, `pschema-rs` has been tested against the already described Shapes tracking the time it takes for the tool to process the dataset and the memory consumed. The program is run three times for each Shape Expression, after which the average results are calculated. Note that a machine with an Intel(R) Xeon(R) Silver 4214 CPU @ 2.20GHz (12 cores and 24 threads), and 40GB of RAM memory installed is used to run the experiment. The results are presented in the table below:

Shape Expression	Number of triples Initially	Number of resulting triples	Time (s)	Memory consumption (GB)
`protein`	7,346,129	226,241	14.58	3.87
`subcellular_location`	7,346,129	1,084,151	37.76	3.75

In relation to the results obtained, it is important to note that despite being in the early stages of development, the tool demonstrates the capability to process large datasets in a reasonable amount of time, while maintaining an acceptable level of memory consumption. To achieve this, numerous optimizations were implemented, including the use of a system that converts textual representations of subject-predicate-object triple into numeric ones. This approach enables the tool to utilize less memory and improve its speed through the cache. Additionally, it is observed that the memory usage of the tool does not appear to be directly related to the complexity of the expression, but with the number of triples matched. However, further testing is being conducted to make this assertion more robust.

Analysed some use cases to extract subsets from PubChem, GlyCosmos and Reactome

Kiyoko suggested that we could use example 10 from PubChem documentation which works on PubChem RDF data. This query could be used to link the proteins in PubChem with those in UniProt. We analysed the RDF data dumps

Another use case suggested from GlyCosmos was this SPARQL query:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX glycan: <http://purl.jp/bio/12/glyco/glycan#>
PREFIX gco: <http://purl.jp/bio/12/glyco/conjugate#>
PREFIX faldo: <http://biohackathon.org/resource/faldo#>
PREFIX dcterms: <http://purl.org/dc/terms/>
PREFIX mass: <https://glycoinfo.gitlab.io/wurcsframework/org/glycoinfo/wurcsframew

SELECT
(SUBSTR(STR(?protein), 33) AS ?ac)
?beginP
    ?saccharide
WHERE {
      ?protein <http://purl.jp/bio/12/glyco/conjugate#glycosylated_at> ?glyco_site
    ?glyco_site faldo:location ?site  .
    ?site faldo:position ?beginP .
    ?glyco_site <http://purl.jp/bio/12/glyco/conjugate#has_saccharide> ?saccharide
  }
```

That SPARQL query should work in GlyCosmos SPARQL endpoint which follows this schema. The RDF dump can be obtained by downloading the results of this SPARQL query as RDF:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX glycan: <http://purl.jp/bio/12/glyco/glycan#>
PREFIX gco: <http://purl.jp/bio/12/glyco/conjugate#>
PREFIX faldo: <http://biohackathon.org/resource/faldo#>
PREFIX dcterms: <http://purl.org/dc/terms/>
PREFIX mass: <https://glycoinfo.gitlab.io/wurcsframework/org/glycoinfo/wurcsframew
CONSTRUCT {
  ?protein <http://purl.jp/bio/12/glyco/conjugate#glycosylated_at> ?glyco_site .
  ?glyco_site faldo:location ?site  .
  ?site faldo:position ?beginP .
  ?glyco_site <http://purl.jp/bio/12/glyco/conjugate#has_saccharide> ?saccharide .
}
WHERE {
    ?protein <http://purl.jp/bio/12/glyco/conjugate#glycosylated_at> ?glyco_site .
    ?glyco_site faldo:location ?site  .
    ?site faldo:position ?beginP .
    ?glyco_site <http://purl.jp/bio/12/glyco/conjugate#has_saccharide> ?saccharide
}
```

Reactome

Kiyoko also suggested a query from Reactome for extracting pathways involving glycoproteins. Reactome data are currently not accessible from any endpoint, so the data would first need to be downloaded using BioPAX level 3 format and loaded into a triplestore. Based on this, Kiyoko suggested using this SPARQL query:

```
PREFIX biopax3: <http://www.biopax.org/release/biopax-level3.owl#>
SELECT DISTINCT ?pathway_ID ?pathwayName ?organism
FROM<http://rdf.glycosmos.org/pathway_reactome_v83>
WHERE {
  ?pathway a biopax3:Pathway .
  ?pathway biopax3:displayName ?pathwayName .
  ?pathway biopax3:organism/biopax3:name ?organism .
```

```
  ?pathway biopax3:xref ?unixref .
  ?unixref biopax3:db ?db_g .
  FILTER(STR(?db_g)="GlyCosmos")
  ?pathway biopax3:xref ?unixref_r .
  ?unixref_r biopax3:db ?db_r ;
                    biopax3:id ?pathway_ID .
  FILTER(STR(?db_r)="Reactome")
}
```

which should be run in the following SPARQL endpoint: https://ts.glycosmos.org/sparql and the expected results should be the following.

TogoGenome

Hiroshi Mori and Yoko Okabepu created this example SPARQL query which is working against the TogoGenome SPARQL endpoint. TogoGenome is an RDF-based genome database developed by DBCLS. This SPARQL query retrieves the UniProt protein ID list for a single microbial strain (*Hydrogenobacter thermophilus* TK-6). Jose Labra converted the SPARQL query to ShEx.

Subsetting book

Andra Waagmeester created an executable book which will contain examples and descriptions about how to create RDF subsets.

The book was automatically published using Github actions created by Núria Queralt Rosinach.

Work on sheXer

sheXer is a tool that extracts ShEx descriptions from RDF content (in local dumps or remote files/endpoints) by mining instance-level data. sheXer is able to produce ShEx schemas and constraint-level stats w.r.t. compliance with the original RDF input. The stats are expressed as inline comments next to the shapes extracted. A problem of this tool is that it can raise out-of-memory errors or spend too much computation time when handling too large RDF dumps. A possibility to overcome this issue is to split the RDF dumps in portions, run sheXer for those portions and consolidate their results. During the biohackathon, Daniel Fernández created a tool that consolidates ShEx results.

The proposed sollution is able to handle shexer's output, so it is adequate for this use case. However, It assumes some structural features in the ShEx inputs, so it may raise issues when trying to use it with input ShEx files not generated by sheXer.

This tool is able to detect shape labels mentioned in more than one result file and merge their internal constraints in a new single shape. The integration of constraints ensure conformance between every instanceused to extract a shape in the input files and its correspondent shape. This is achieved by including a proper cardinality when a certain constraint is not observed among all the shapes to merge.

The tool is also capable of keeping sheXer's stats regarding number of instances and frequency of each observed constraint. However, the stats attached to merged shapes may be affected by cases in which a certain instance was used to extract a shape in N (more than one) partial dumps. In such cases, the instance will have N times more weight than it should in the merged stats.

Use case about Multi-omics data integration

We were discussing a possible use case about data integration on multi-omics.

Med2RDF

- UniProt
- A possible example from natural language: "Patient with headache and some symptoms"
- Tasks to do: Obtain relationships, extract triplets, obtain treatments/possible diagnostics/test
- HPO
- https://athena.ohdsi.org/ (OMOP vocabularies)
- Generate RDF from relationships

Possible tools to use: ShExML.

Analysis about RDF data description mechanisms

As with any data source, it is necessary to describe its structure. And if we want to obtain subsets of RDF data then the RDF data has to be described in sufficient detail that can facilitate the subsetting process. One way is to write ShEx to describe what the subset ought to be. Another approach would be to see if other modes of representation can be used to describe the structure of RDF data and then generate shapes from this description.

As part of this effort, we explored the following:

- LinkML: Linked Data Modeling Language (LinkML), a flexible modeling language that allows you to define schemas in YAML that describe the structure of your data. LinkML also provides a framework for translating this YAML representation into other forms like ShEx.
- RDF-config: RDF-config is a tool that generates SPARQL queries, schema diagrams, and most importantly ShEx from a simple YAML-based configuration that conforms to a specification.

Both LinkML and RDF-config describe the structure of RDF data in YAML and have sufficient tooling to automatically generate ShEx shapes. As a result, we decided to compare the ShEx output of both tools by using PubChem as the data source of interest.

PubChem provides an RDF endpoint and has captured the structure of their RDF data as a LinkML YAML. This is the file that serves as an input to LinkML to generate ShEx shapes. Deepak provided the ShEx shapes derived from the YAML.

Alternatively, Toshiaki had captured the structure of RDF data in PubChem in the YAML-based configuration of RDF-config. After which, this YAML file was used to generate ShEx shapes using RDF-config tools. Toshiaki provided the ShEx shapes derived from the YAML.

It is clear that the philosophy adopted by LinkML and RDF-config is similar:

- define the structure of RDF data in YAML
- automatically generate ShEx shapes based on the YAML

Next, we analyzed the shapes and a thorough side by side comparison revealed that the shapes generated by RDF-config are cleaner and simpler. This can be attributed to the fact that the RDF-config YAML was prepared after investigating the PubChem RDF endpoint and carefully cataloguing the classes, properties, and predicates. On the other hand, the shapes generated by LinkML contained more properties and uses specific types that are native to LinkML but provides more expressivity.

Following is an example of the `Taxonomy` shape generated by LinkML:

```
<Taxonomy> CLOSED {
    (  $<Taxonomy_tes> (  rdf:type @linkml:Uri * ;
          skos:prefLabel @linkml:String ? ;
          skos:altLabel @linkml:String ? ;
          owl:sameAs @linkml:Uri ? ;
          skos:closeMatch @linkml:Uri *
```

```
        ) ;
        rdf:type [ <Taxonomy> ] ?
    )
}
```

In the case of rdfconfig. We found that the initialTaxonomy shape was the following:

```
<PubChemTaxonomyShape> {
  rdf:type [biopax:taxonomy] ;
  dcterms:title xsd:string  ;
  skos:closeMatch IRI *
}
```

Which was later improved during the biohackathon. The new updated version obtained from RDF-config was:

```
<PubChemTaxonomyShape> {
  rdf:type [sio:SIO_010000] ;
  skos:prefLabel xsd:string  ;
  skos:altLabel xsd:string * ;
  skos:closeMatch IRI * ;
  cito:isDiscussedBy IRI * ;
  owl:sameAs IRI
}
```

One aspect of the shapes generated by LinkML is that the constraint applied on `rdf:type` property needs to be improved. In the example above, it is clear that the constraint on `rdf:type` is self-referential, i.e. the constraint states that all instances of `Taxonomy` must have `rdf:type` as `Taxonomy`. This is the default expression of ShEx from LinkML and definitely needs improvement. Another way to express the `rdf:type` constraint would be to explicitly state the values for `rdf:type` as enumeration in the LinkML YAML such that this is parsed appropriately. When we tried expressing the enumeration and then translating the YAML to ShEx, we realized that the enumerations were not being parsed properly. There is a ticket in the LinkML repository on GitHub and a corresponding pull request that fixes this issue.

Future work

Add FROM <...> to ShEx

- Problem to solve: how to validate/extract RDF data when it is behind different RDF graphs?
- Currently ShEx processors assume a single RDF graph but it may be interesting to add some kind of annotation to ShEx which allows ShEx processors to take into accout the `FROM` declarations when they are working against SPARQl endpoints.

Machine-readable way that RDF data providers describe their RDF dumps

One problem for creating RDF data subsets from RDF data dumps is that not all RDF providers follow the same structure to publish the RDF dumps and that the descriptions of the contents of those RDF dumps are not published.

In order to solve that problem it would be interesting to follow some guidelines or common patterns such as providing files in the `./well-known/` namespace. These files should contain or direct to machine-readable descriptions in either ShEx or SHACL.

During the biohackathon we were reviewing the way that different data providers follow to describe their RDF dumps. Some of the providers we took into account were: - Uniprot -

PubChem - TogoGenome

In order to add `.well-known/void` declarations, Jerven Bolleman already had the void-generator. VoID is a framework to describe the statistical distribution of data elements and their links. During the biohackathon, Jerven Bolleman and Jose Labra started a new project called void2shapes to generate ShEx and SHACL declarations from void descriptions in UniProt. We succeeded in generating a minimal ShACL representation from the VoID files of UniProt and SwissLipids.

Improve the ShEx tooling

We are currently transitioning some of the ShEx implementations that were initially implemented in Scala to Rust. We have been looking to some tools in the Rust ecosystem like the Severless SPARQL endpoint that has been developed by Nishad in Javascript for Deno based on the Oxigraph database that is implemented in Rust that has a binding that works in WebAssembly. We consider that this solution could enable further performance and dynamic scalability and enable edge-computing use cases. One possible use case would be to adapt the next Rust implementation of ShEx to work on WebAssembly and could be integrated with OxiGraph and do ShEx validation on-the-fly.

Improvements to sheXer

- Automatic splitting of large RDF data files
- Parallel/distributed extraction of ShEx portions
- Automatic consolidation of ShEx portions
- Generation of rdfconfig yaml files

SPARQL to ShEx converter

- We noticed that in several use cases, we start with an example SPARQL query which we want later to convert to ShEx. An interesting project would be to generate ShEx from those SPARQL queries.

Explore the idea about ShEx transitions

- Being able to declare transitions between ShEx schemas so they can be checked as pre- and post- conditions, example before/after SPARQL updates. This idea has been provided by Thomas Liener in this document.

ShEx template injection

- Being able to have parameterizable ShEx schemas whose specific values could be injected in a way that ShEx schemas could be dynamically generated

RDF representation of ShapeMaps

- We resumed the work on this issue and we expect to update the ShapeMap specification including a description of a representation of shape maps that shows how to represent them in RDF.

Generating ShEx files from void descriptions

- RDF representation of ShEx uses RDF lists which are a bit challenging to generate from SPARQL Construct queries. Jerven Bolleman solved it generating IRIs for the intermediate nodes but it may be interesting to generate the nodes in the list as blank nodes. Eric Prod'hommeaux raised this issue in Apache Jena and it would be interesting work to see if we could solve the issue or find some way to solve the problem.

- This work will be folllowed on in the void2shapes project.

Conversion from ShEx to LinkML

Another topic of discussion was the possibility of converting ShEx shapes to LinkML. This would allow for the conversion of existing ShEx shapes to a flexible format such as the LinkML YAML and enable the subsequent use of the tooling provided by LinkML to generate other outputs that are supported by the LinkML framework.

For example, if we have a ShEx shape `GeneShape`,

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX SO: <http://purl.obolibrary.org/obo/SO_>
PREFIX dcterms: <http://purl.org/dc/terms/>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

<GeneShape> {
  rdf:type [ SO:0000704 ] ;
  dcterms:title xsd:string  ;
  dcterms:alternative xsd:string * ;
}
```

Then we should be able to convert and represent this shape in LinkML YAML that looks like so,

```
id: http://example.org/Example
name: Example

prefixes:
  ex: http://example.org/
  linkml: https://w3id.org/linkml/
  rdf: http://www.w3.org/1999/02/22-rdf-syntax-ns#
  SO: http://purl.obolibrary.org/obo/SO_
  dcterms: http://purl.org/dc/terms/
  xsd: http://www.w3.org/2001/XMLSchema#

imports:
  - linkml:types

default_prefix: ex
default_range: string

classes:
  Gene:
    slots:
      - type
      - title
      - alternative
    slot_usage:
      type:
        slot_uri: rdf:type
        range: TypeEnum
      title:
        slot_uri: dcterms:title
      alternative:
        slot_uri: dcterms:alternative
```

```
slots:
  type:
  title:
  alternative:

enums:
  TypeEnum:
    permissible_values:
      gene:
        meaning: SO:0000704
```

The above LinkML YAML is a working example and can be parsed using LinkML framework.

Preliminary analysis seem to indicate that the conversion is possible but with some limitations which can be addressed if the use-cases are well defined and sufficent examples are available for rapid prototyping. This is something that Jose Labra and Deepak would like to work on in the future.

Conversion from ShEx to RDF-config YAML files

Another possibility is to explore the generation of RDF-config YAML files that follow this specification.

Use case about integrating RDF data and clinical records

The accelerating pace of discoveries in biotechnology offers great hope in medicine. Yet, access to clinical information necessary for research and innovation has been challenging. Clinical data often resides in proprietary stores and proprietary formats that are not amenable to integration with other data sources. The increasing adoption of the Fast Healthcare Interoperability Resources (FHIR) standard by Electronic Health Records (EHR) vendors and FHIR's support for RDF is offering hope that clinical data may soon be made more accessible to a broader community. Moreover, current investigations on how to support known semantic web standards such as SPARQL, ShEx and ShExMap within FHIR all stand to make FHIR and thereby clinical data more accessible to the broader research community.

Use case about drug repurposing

Computational drug repurposing is a well known strategy to speed up drug development with potentially lower overall development costs and shorter development timelines with de-risked 'old' compounds. However, current knowledge graphs used in AI-based drug repurposing lack metabolite-related information, which is data especially interesting to include for metabolic diseases drug discovery. In this BioHackathon, we investigated ways to retrieve metabolite-related information crossing several knowledge bases such as UniProt or PubChem to feed drug repurposing knowledge graphs. Even though there is a lot of curated metabolite data stored, this data is not neither easily nor systematically retrievable from databases. In the future, we aim at producing efficient ways for metabolite-related information retrieval for knowledge graph construction based on subsetting strategies such as using RDFconfig to generate SPARQL queries or using ShEx-based approaches.

Acknowledgements

We would like to thank the fellow participants at BioHackathon 2023 for their collaboration and constructive advice, which greatly influenced our project. We are grateful to the organizers for providing this platform and the developers of open source language models. Special thanks to our mentors, advisors, and colleagues for their guidance and support. Without their

contributions, our project in linked data standardization with LLMs in bioinformatics would not have been possible.

References

Hosseini, J. E. A. W., Seyed AND Labra-Gayo. (2023). *Wikidata subsetting: Approaches, tools, and evaluation*. https://www.semantic-web-journal.net/content/wikidata-subsetting-approaches-tools-and-evaluation-0

Prud'hommeaux, E., Labra Gayo, J. E., & Solbrig, H. (2014). Shape expressions: An RDF validation and transformation language. *Proceedings of the 10th International Conference on Semantic Systems, SEMANTICS 2014*, 32–40.