Universidad de Oviedo

Trabajo de investigación

Plaza F036-570-DFA0340
Área de Lenguajes y Sistemas Informáticos

Departamento de Informática

Jose Emilio Labra Gayo

Universidad de Oviedo

Índice general

Universidad de Oviedo

1. Introducción

Este documento contiene el trabajo de investigación realizado por el candidato Jose Emilio Labra Gayo para la plaza de Catedrático de Universidad número 16, convocada por la Universidad de Oviedo en la Resolución de 25 de Junio de 2021, (BOE de 9 de Julio de 2021), con el código F036-570-DFA0340 del Cuerpo de Catedrático de Universidad/, para el área de conocimiento Lenguajes y Sistemas Informáticos.

Oviedo,
Septiembre de 2021

Jose Emilio Labra Gayo

2. Contexto del trabajo de investigación

El presente trabajo de investigación se enmarca dentro de la línea de investigación llevada a cabo por el candidato en los últimos años dentro del grupo de investigación WESO.

2.1 Justificación de la temática

Se ha escogido como temática para el trabajo de investigación la creación de subconjuntos de grafos de conocimiento mediante Shape Expressions por varios motivos:

- El creciente interés en la utilización de grafos de conocimiento, especialmente Wikidata.
- La implicación del candidato en la creación del lenguaje Shape Expressions y sus diferentes aplicaciones
- La participación del candidato como coordinador en varios *hackathones* con la temática de la creación de subconjuntos de grafos de conocimiento.
- La subcontratación durante el verano de 2021 del equipo WESO por parte de la Universidad de Virginia para la realización de un prototipo que mejorase el proyecto Scholia, que incluía como tarea la utilización de subconjuntos de Wikidata en el ámbito de Scholia
- La aprobación del proyecto de investigación presentado en el primer ejercicio, que también incluía como una tarea la creación de subconjuntos de grafos de conocimiento.

Los motivos anteriores han servido de aliciente al candidato para involucrarse en los últimos meses en la temática del trabajo de investigación.

2.2 Formato del trabajo de investigación

En el Reglamento para los concursos de acceso a los cuerpos de funcionarios docentes universitarios de la Universidad de Oviedo, aprobado por el Consejo de Gobierno de 18 de diciembre de 2008 (Boletín Oficial del Principado de Asturias de 14 de enero de 2009) se indica que para la segunda prueba del concurso, el candidato debe presentar un resumen de su trabajo de investigación, sin especificar el formato ni el idioma del mismo.

Se ha considerado conveniente presentar el trabajo de investigación con la estructura de un artículo científico extendido y se ha publicado como un *preprint* en el repositorio Arxiv [1].

Una vez finalizado el concurso, será enviado para su posible publicación a un congreso o revista de investigación que todavía no ha sido seleccionado.

Por ese motivo, el artículo se presenta en idioma inglés dado que en la convocatoria no se hace mención explícita a requisitos de idioma.

2.3 Nuevas líneas de investigación

El trabajo de investigación aquí presentado ha permitido al candidato abrir nuevas líneas de investigación en las cuales todavía no se había trabajado y que se considera que pueden ser prometedoras:

- La validación mediante Shape Expressions de grandes grafos de conocimiento como Wikidata mediante tecnologías de procesamiento escalables como Apache Spark y Pregel. Esta línea se considera muy prometedora y con grandes aplicaciones, puesto que se desconoce la existencia de ningún validador que permita afrontar grafos de conocimiento del tamaño de Wikidata y el algoritmo presentado en el presente trabajo sí lo permite.
- La creación de una extensión del lenguaje Shape Expressions para describir y validar Wikidata puede facilitar la adopción del lenguaje por los expertos de dominio, al tratar las referencias y cualificadores como elementos de primera clase, mejorando la legibilidad de las Shape Expressions sobre los esquemas de entidades actuales, que describen la serialización RDF.
- La definición formal de grafos Wikidata, en un mismo ámbito que los *property graphs* y los grafos RDF, permite la comparación en un mismo marco de dichas tecnologías, facilitando la identificación de las características comunes y diferenciadoras de las mismas.
- La creación de una extensión del lenguaje Shape Expressions para la validación de *property graphs* puede ser útil para la adopción del lenguaje entre la comunidad de usuarios de dichos grafos.

[1] `https://arxiv.org/`

3. Creating Knowledge Graphs Subsets using Shap

3.1 Abstract

The initial adoption of knowledge graphs by Google and later by big companies has increased their popularity. In this paper we present a formal model for three different types of knowledge graphs which we call RDF-based graphs, property graphs and wikibase graphs. Although Shape Expressions were initially created to describe and validate RDF-based graphs, we present an extension of the language that can also be used to describe property graphs and wikibase graphs. An important problem of knowledge graphs is the large amount of data which jeopardizes their practical application. In order to palliate the problem, one approach is to create subsets of those knowledge graphs for some domains. We review some approaches that can be used to generate those subsets employing descriptions of the subsets using the Shape Expressions language.

3.2 Introduction

Review all the introduction. Empezar con alguna frase chula sobre grafos de conocimiento...

Since Google announced in 2012 the use of knowledge graphs to improve their search results in 2012 [61], Knowlege Graphs have been increasingly adopted by the industry. Several companies like Airbnb [9], Amazon [33], eBay [54], Facebook [50], IBM [15], LinkedIn [24], Microsoft [60], Uber [23], etc. have already announced their use of some kind of Knowledge Graphs.

Hablar de los 3 tipos de grafos de conocimiento tratados en el paper brevemente

There have been several approaches to represent knowledge using graphs. In this paper we will review and formalize three of them: RDF-based graphs, property graphs and wikibase graphs.

RDF-based graphs is one of the most well-known approaches given that RDF was proposed as a W3C recommendation already in 1999 [51] and a large ecosystem of tools have been created around it. An important aspect of RDF is the use of URIs, which facilitates interoperability.

re a bit about data? SPARQL?

On the other hand, graph databases like Neo4J have also been employed to represent knowledge graphs. They employ a data model which allows to annotate both vertices and edges with pairs of property-values and has become to be known as property graphs.

Wikidata started in 2012 as a support project for Wikipedia but has been evolving and acquiring more and more importance as a hub of public knwowledge. The data model emplyed by Wikidata combined several aspects from RDF following linked data principles and from property graphs, allowing the annotation of statements by property-values using qualifiers and references. The software suite that implements Wikidata is known as Wikibase and can be used to represent other knowledge graphs with the same data model, we will refer to these kind of knowledge graphs as wikibase graphs.

Given that knowledge graphs usually capture information about some real world entities, it is common that their size increases with their usage. A good example is the evolution of Wikidata...

algunas estadís-

Motivation:
- Talk about why the success of knowledge graphs can also be a problem for performance...
- Talk about the sucess of ShEx as an intuitive language

once the paper ed

The main contributions of this paper are:

- We created a formal model for Wikibase graphs
- We extended the ShEx language to describe Wikibase graphs
- We propose a common framework that facilitates the comparison between 3 types of knowlege graphs: RDF-based knowledge graphs, property graphs and wikibase graphs.
- We formalize several approaches to generate subsets of Wikibase graphs:
  - Item and property generated subgraphs
  - Matching-generation subgraphs
  - ShEx + Slurping
  - ShEx based traversal
  - ...

The structure of the paper is as follows: Section 3.3 presents some preliminary definitions about sets and graphs. Section 3.4 introduces knowledge graphs and presents 3 main types of knowledge graphs: RDF-based, Property graphs and Wikibase graphs. Section 3.5 presents techniques to describe knowledge graphs. We describe the Shape Expressions language that has been developed to describe RDF-based graphs, and we extend it to describe property graphs and wikibase graphs. Section 3.6 presents the problem of creating subsets of knowledge graphs and describes several approaches to create subsets of wikibase graphs. Section 3.7 describes some real-world applications where it is important to generate subsets of knowledge graphs. Section 3.8 presents some results of those applications. Finally, section 3.9 reviews the related work and section 3.10 presents some conclusions.

3.3 Preliminaries

Definitions

In this section, we provide some basic definitions that we will use in the rest of the paper.

Sets

. The finite set with elements $a_1, \ldots, a_n$ is written $\{a_1, \ldots, a_n\}$, $\emptyset$ represents the empty set, $S_1 \cup S_2$ is the union of sets S_1 and S_2, $S_1 \cap S_2$ the intersection and $S_1 \times S_2$ the Cartesian product. $\mathtt{FinSet}(S)$ represents the set of all finite subsets of S. A tuple $\langle A_1, \ldots A_n \rangle$ is the cartesian product $A_1 \times \cdots \times A_n$.

Given a set S, its set of partitions is defined as $\mathtt{part}(s) = \{(s_1, s_2) \mid s_1 \cup s_2 = s \wedge s_1 \cap s_2 = \emptyset\}$.

Definition 3.3.1 — Graph. A *graph* is a tuple $G = \langle \mathcal{V}, \mathcal{E} \rangle$, where $\mathcal{V}$ is a set of nodes, and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{E} \times \mathcal{V}$ is a set of edges.

A multigraph is a graph where it is possible to have more than one edge between the same two nodes.

Definition 3.3.2 — Directed edge-labelled graph. A *directed edge-labelled graph* is a tuple $\mathcal{G} = \langle \mathcal{V}, \mathcal{E}, \mathcal{P} \rangle$, where $\mathcal{V}$ is a set of nodes, $\mathcal{P}$ is a set of labels also called predicates or properties, and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{L} \times \mathcal{V}$ is a set of edges. Each element $(x, p, y) \in \mathcal{E}$ is called a triple, where x is the subject, p is the predicate or property and y is the object.

Definition 3.3.3 — Triple-based graphs. A *triple-based graphs* is a directed edge-labelled graph $\mathcal{G} = \langle \mathcal{S}, \mathcal{P}, \mathcal{O}, \rho \rangle$ where $\mathcal{S}$ is a set of subjects, $\mathcal{P}$ is a set of predicates or properties and $\mathcal{O}$ is a set of objects or values, and $\rho \subseteq \mathcal{S} \times \mathcal{P} \times \mathcal{O}$. Those sets don't need to be disjoint, and usually $\mathcal{P} \subseteq \mathcal{S} \subseteq \mathcal{O}$.

Definition 3.3.4 — Hypergraph. A *hypergraph* is a tuple $G = \langle \mathcal{V}, \mathcal{E} \rangle$ where $\mathcal{V}$ is a set of nodes and $\mathcal{E} \subseteq \mathtt{FinSet}(\mathcal{V})$ is a set of edges. Notice that $\mathcal{E}$ is a family of subsets of vertices.

Bag

Regular bag exp

Regular bag exp matching

Definition 3.3.5 — Shape assignment. Given a graph $\mathcal{G}$ with vertex set $\mathcal{V}$ and a finite set of labels $\mathcal{L}$, a *shape assignment* over $\mathcal{G}$ and $\mathcal{L}$ is a subset of $\mathcal{V} \times \mathcal{L}$. We use τ to denote shape assignments, and we write $n@l$ instead of (n, l) for elements of shape assignments. Note that shape assignments correspond to shape maps in [56] and typings in [5, 62].

3.4 Knowledge graphs models

Although the term *knowledge graphs* was already in use in the 1970s [57], the current notion of knowledge graphs was popularized by Google in 2012 [61].

We adopt an informal definition of knowledge graphs which has been inspired by Hogan et al [27]:

Definition 3.4.1 — Knowledge graph. A *Knowledge graph* is graph of data intended to represent knowledge of some real world domain, whose nodes represent entities of interest and whose edges represent relations between these entities.

review

The previous definition is deliberately open. The main feature of a knowledge graph is that it is intended to represent information about entities of some real world domain using a graph-based data structure.

Knowledge graphs are usually classified as:

- Licence/proprietor: There are public and open knowledge graphs like Yago [63], DBpedia [36] or Wikidata [67] as well as enterprise-based and proprietary knowledge graphs [50] like Google, Amazon, etc.
- Scope: there are general-purpose knowledge graphs which contain information about almost all domains like Wikidata as well as domain specific knowledge graphs which contain information from some specific domains like healthcare, education, chemistry, biology, cybersecurity, etc. [1]

Knowledge graphs can be represented using multiple technologies and in fact, the information about how Google's knowledge graph is implemented is not public. Nevertheless, in this paper, we will focus on three main technologies:

- **RDF based knowledge graphs** represent information using directed graphs whose edges are labels.
- **Property graphs** allow property–value pairs and a label to be associated with nodes and edges. Property graphs have been implemented by several popular graph databases like Neo4j [3].
- **Attributed graphs** allow property-value pairs associated with edges to add meta-data about the relationship represented by the edge and the values of those properties can themselves be nodes in the graph. The main example in this category is Wikidata where property-value pairs encode qualifiers and references.

3.4.1 RDF based knowledge graphs

Resource Description Framework (RDF) [13], is a W3C recommendation which is based on directed edge-labelled graphs.

The RDF data model defines different types of nodes, including *Internationalized Resource Identifiers* (*IRIs*) [16] which can be used to globally identify entities on the Web; literals, which allow for representing strings (with or without language tags) and values from other datatypes (integers, decimals, dates, etc.); and *blank nodes*. Blank nodes can be considered as existential variables that denote the existence of some resource for which an IRI or literal is not known or provided. They are locally scoped to the file or RDF store, and are not persistent or have portable identifiers [26].

Definition 3.4.2 — RDF Graph. Given a set of IRIs $\mathcal{I}$, a set of blank nodes $\mathcal{B}$ and a set of literals *Lit*, an *RDF graph* is a triple based graph $\mathcal{G} = \langle \mathcal{S}, \mathcal{P}, \mathcal{O}, \rho \rangle$ where $\mathcal{S} = \mathcal{I} \cup \mathcal{B}$, $\mathcal{P} = \mathcal{I}$, $\mathcal{O} = \mathcal{I} \cup \mathcal{B} \cup Lit$ and $\rho \subseteq \mathcal{S} \times \mathcal{P} \times \mathcal{O}$

There are several syntaxes for RDF graphs like Turtle, N3, RDF/XML, etc. In this document, we will use Turtle.

Example 3.1 As a running example, we will represent information about Tim Berners-Lee declaring that he was born in London, on 1955, and was employed by CERN and London's country is UK. That information can be encoded in Turtle as:

```
prefix :      <http://example.org/>
prefix xsd:   <http://www.w3.org/2001/XMLSchema#>

:timbl   :birthPlace     :London ;
         :birthDate      "1955-08-06"^^xsd:date ;
         :employer       :CERN .
:London  :country        :UK .
```

Review the visualization after removing foundingMember information and added awards

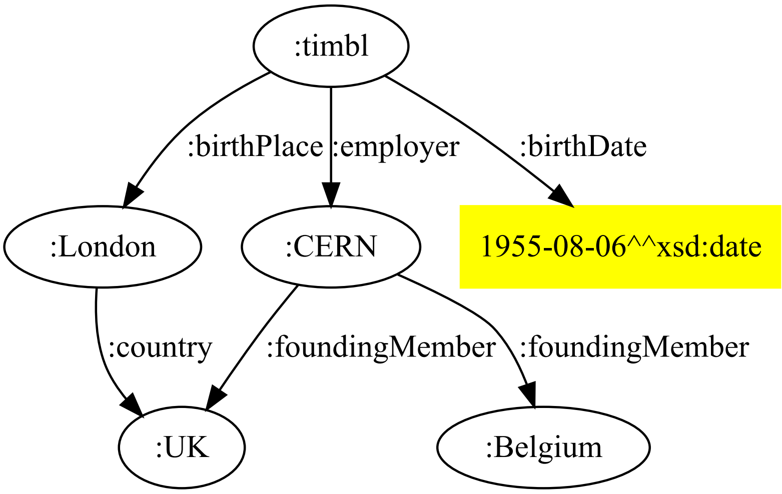

Figure 3.1: Example graph representation of RDF data

Figure 3.1 shows a possible visualization of that RDF graph using RDFShape, a tool developed by the authors of this paper which allows to play with RDF graphs[1] [34]:

■

The neighbors of a node $n \in \mathcal{V}$ in an RDF graph $\mathcal{G}$ are defined as $neighs(n, \mathcal{G}) = \{(n, p, y) \mid (n, p, y) \in \mathcal{G}\}$.

RDF can be considered the basic element of the semantic web technology stack, forming a simple knowledge representation language on top of which several technologies have been developed like SPARQL for querying RDF data as well as RDFS and OWL to describe vocabularies and ontologies.

RDF reification and RDF-*

An important aspect of RDF as a knowledge representation formalism is to be able to represent information about RDF triples themselves, which is called reification. In this section we will present some of the possible approaches for reification using a simple example to help understand the approach used by Wikibase to serialize its data model to RDF. We also present the RDF-* approach which has become popular in the RDF-ecosystem with its support by several RDF stores like GraphDB [2].

■ **Example 3.2** As an example, we may want to qualify the statement that Tim Berners-Lee was employed by CERN, declaring that he was employed at two different points in time: in 1980 and between 1984 and 1994.

Several approaches have already been proposed RDF reification [25]:

- Standard RDF reification was introduced in RDF 1.0 [41]. It consists of using the predicates `rdf:subject`, `rdf:predicate` and `rdf:object` as well as the class `rdf:Statement` to explicitly declare statements.

```
_:s1 rdf:type          rdf:Statement ;
     rdf:subject       :timbl ;
     rdf:predicate     :employer ;
     rdf:object        :CERN ;
     :start            "1980"^^xsd:gYear ;
     :end              "1980"^^xsd:gYear .
```

[1] It is possible to interactively play with the example following this permalink: `https://rdfshape.weso.es/link/16275427216`

[2] `https://www.ontotext.com/knowledgehub/fundamentals/what-is-rdf-star/`

```
_:s2 rdf:type          rdf:Statement ;
     rdf:subject       :timbl ;
     rdf:predicate     :employer ;
     rdf:object        :CERN ;
     :start            "1984"^^xsd:gYear ;
     :end              "1994"^^xsd:gYear .
```

- Create a statement that models the n-ary relation [17]. For example, we can create two nodes `:s1` and `:s2` to represent the the 2 employments of Tim-Berners-Lee at CERN.

```
:timbl :employer :s1, :s2.
:s1 :employerV :CERN;
    :start      "1980"^^xsd:gYear ;
    :end        "1980"^^xsd:gYear .
:s2 :employerV :CERN;
    :start      "1984"^^xsd:gYear ;
    :end        "1994"^^xsd:gYear .
```

- Create *singleton properties* for each statement and link those properties with a specific predicate to the real property [48].

```
:timbl :employer1 :CERN ;
       :employer2 :CERN .

:employer1 :singletonPropertyOf :employer ;
 :start "1980"^^xsd:date ;
 :end   "1980"^^xsd:date .

:employer2 :singletonPropertyOf :employer ;
 :start "1984"^^xsd:date ;
 :end   "1994"^^xsd:date .
```

- RDF1.1 [13] included the concept of named graphs, which can be used to associate each triple with a different graph.

```
:g1 :timbl    :employer :CERN .
:g1 :employed :start    "1980"^^xsd:date   .
:g1 :employed :end      "1980"^^xsd:date   .
:g2 :timbl    :employer :CERN .
:g2 :employed :start    "1984"^^xsd:date   .
:g2 :employed :end      "1994"^^xsd:date   .
```

- RDF-* [14] has been recently introduced as an extension of RDF that includes RDF graphs as either the subjects or objects of a statement.

```
<<:timbl :employer :CERN>> :start "1980"^^xsd:gYear ;
                           :end   "1980"^^xsd:gYear .
<<:timbl :employer :CERN>> :start "1984"^^xsd:gYear ;
                           :end   "1994"^^xsd:gYear .
```

- Wikidata's RDF serialization follows a hybrid approach using a direct link to capture the preferred value and singleton nodes that represent the statements capturing the n-ary relationship [17]. It also follows a convention that employs the same local name of the property preceded by different namespaces: `wdt:` for the direct link, `p` for the link between the node and the singleton statements, `ps:` for the link between the singleton statements and the values, and `pq:` for the link between the singleton statements and the qualified values. The previous example using Wikidata RDF serialization could be [3]:

```
:timbl wdt:employer :CERN .
:timbl p:employer :s1 .
:timbl p:employer :s2 .
:s1    ps:employer :CERN ;
       pq:start "1980"^^xsd:gYear ;
       pq:end   "1980"^^xsd:gYear .
:s2    ps:employer :CERN ;
       pq:start "1984"^^xsd:gYear ;
       pq:end   "1994"^^xsd:gYear .
```

■

3.4.2 Property graphs

Property graphs have become popular thanks to several commercial graph databases like Neo4j [4], JanusGraph [5] or Sparksee [6]. A property graph has unique identifiers for each node/edge and allows to add property-value annotations to each node/edge in the arc as well as type annotations.

The following definition of a property graph follows [59].

Definition 3.4.3 — Property graph. Given a set of types $\mathcal{T}$, a set of properties $\mathcal{P}$, and a set of values $\mathcal{V}$, a *property graph* $\mathcal{G}$ is a tuple $\langle \mathcal{N}, \mathcal{E}, \rho, \lambda_n, \lambda_e, \sigma \rangle$ where $\mathcal{N} \cap \mathcal{E} = \emptyset$, $\rho : \mathcal{E} \mapsto \mathcal{N} \times \mathcal{N}$ is a total function, $\lambda_n : \mathcal{N} \mapsto \mathtt{FinSet}(\mathcal{T})$, $\lambda_e : \mathcal{E} \mapsto \mathcal{T}$, and $\sigma : \mathcal{N} \cup \mathcal{E} \times \mathcal{P} \mapsto \mathtt{FinSet}(\mathcal{V})$.

A property graph is formed by a set of node identifiers $\mathcal{N}$ and a set of edges $\mathcal{E}$ where ρ associates a pair of nodes (n_1, n_2) to every $e \in \mathcal{E}$ where n_1 is the subject and n_2 is the object, λ_n associates a set of types for node identifiers (notice that property graphs allow nodes to have more than one type), λ_e associates a types for each edge identifier, and σ associates a set of values to pairs (i, p) such that $i \in \mathcal{N} \cup \mathcal{E}$ is a node or edge and $p \in \mathcal{P}$ is a property.

■ **Example 3.3** As an example, we will represent information about Tim Berners-Lee in a property graph encoding his birth place with a relation to a node that represents London, and his birth date with a value for that property in the same node. We can also represent that its employer has been CERN in two times, one in 1980, and another between 1984 and 1994.

$$
\begin{aligned}
\mathcal{T} &= \{\text{Human}, \text{City}, \text{Metropolis}, \text{Country}, \text{Organization}, \text{birthPlace}, \text{country}, \text{employer}\} \\
\mathcal{P} &= \{\text{label}, \text{birthDate}, \text{start}, \text{end}\} \\
\mathcal{V} &= \{\text{Tim Berners-Lee}, 1955, 1980, 1984, 1994, \text{London}, \text{UK}\} \\
\mathcal{N} &= \{n_1, n_2, n_3, n_4\} \qquad \mathcal{E} = \{r_1, r_2, r_3, r_4\} \\
\rho &= r_1 \mapsto (n_1, n_2), r_2 \mapsto (n_2, n_3), r_3 \mapsto (n_1, n_4), r_4 \mapsto (n_1, n_4) \\
\lambda_n &= n_1 \mapsto \{\text{Human}\}, n_2 \mapsto \{\text{City,Metropolis}\}, n_3 \mapsto \{\text{Country}\}, n_4 \mapsto \{\text{Organization}\} \\
\lambda_e &= r_1 \mapsto \text{birthPlace}, r_2 \mapsto \text{country}, r_3 \mapsto \text{employer}, r_4 \mapsto \text{employer} \\
\sigma &= (n_1, \mathtt{label}) \mapsto \text{Tim Berners-Lee}, \ (n_1, \mathtt{birthDate}) \mapsto 1955 \\
&\quad (n_2, \text{label}) \mapsto \text{London}\}, \ (n_3, \text{label}) \mapsto \mathrm{UK}, \ (n_4, \text{label}) \mapsto \text{CERN} \\
&\quad (r_3, \text{start}) \mapsto 1980, \ (r_3, \text{end}) \mapsto 1980, \ (r_4, \text{start}) \mapsto 1984, \ (r_4, \text{end}) \mapsto 1994
\end{aligned}
$$

■

[3] We omit the representation of values and use English names instead of numbers for clarity

[4] `https://neo4j.com/`

[5] `https://janusgraph.org/`

[6] `https://www.sparsity-technologies.com/#sparksee`

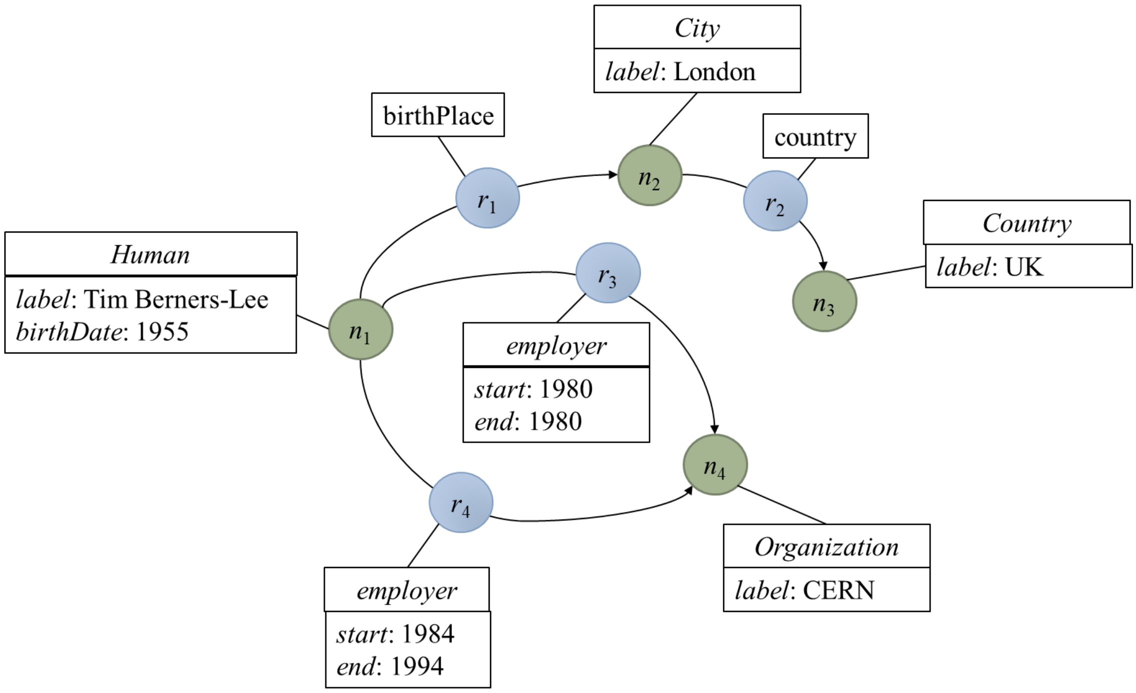

Figure 3.2: Example graph visualization of a property graph

Figure 3.2 presents a possible visualization of a property graph.

Review the figure to remove long dates and add metropolis

Cypher is a property graph query language that was initially developed for Neo4j [19]. The following Cypher script can generate the property graph represented in figure 3.2:

```
CREATE (n1:Human {label:'Tim Berners-Lee', birthDate:1955})
CREATE (n2:City:Metropolis {label:'London'})
CREATE (n3:Country {label:'UK'})
CREATE (n4:Organization {label:'CERN'})
CREATE
  (n1)-[:birthPlace]->(n2),
  (n2)-[:country]->(n3),
  (n1)-[:employer {start:[1980], end: [1980]}]->(n4),
  (n1)-[:employer {start:[1984], end:[1994]}]->(n4),
```

As can be seen in previous example, property graphs are multigraphs because it is possible to have more than one edges between nodes.

3.4.3 Wikibase graphs

Wikidata[7] started in 2012 to support Wikipedia [67]. It has become one of the biggest human knowledge bases, maintained both by humans collaboratively as by bots, which update the contents from external services or databases. Several organizations are donating their data to Wikidata and collaborate in its maintenance providing resources. A remarkable case is Google, which migrated its previous knowledge graph Freebase to Wikidata in 2017 [64].

Apart of Wikipedia, Wikidata has been reported to be used by external applications like Apple's Siri [8] and it has been adopted as the central hub for knowledge in several domains like life

[7] http://wikidata.org/

[8] https://lists.wikimedia.org/pipermail/wikidata/2017-July/010919.html

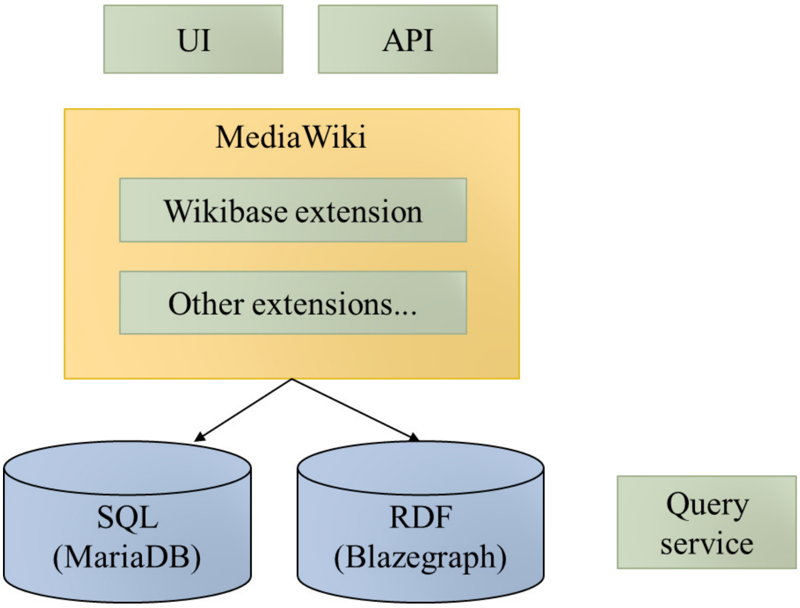

Figure 3.3: Simplified architecture of Wikibase

sciences [8], libraries [58] or social science [12]. As of August, 2021, it contains information about more than 94 millions of entities [9] and since its launch there have been more than 1,400 millions of edits.

Wikibase [10] is a set of open source tools which run Wikidata. With Wikibase it is possible to create Knowledge graphs that follow the same data model as Wikidata but that represent information from other domains. The projects that are using Wikibase are called Wikibase instances, some examples of wikibase instances are Rhizome [11] or Enslaved [12]. Given that Wikidata was the first and most common Wikibase instance the terms are sometimes used indistinctly.

Wikibase was initially created from MediaWiki software which ensured adoption by the Wikimedia community. Internally, Wikidata content is managed by a relational database (MariaDB) which consists of strings stored and versioned as character blobs [40]. but was not suitable for advanced data analysis and querying. With the goal of facilitating those tasks and integrate Wikibase within the semantic web ecosystem, the Wikimedia Foundation adopted BlazeGraph [13] as a complementary triplestore and graph database. In this way, there are 2 main data models that coexist in Wikibase: a document-centric model based on MediaWiki and an RDF-based model based on RDF which can be used to do SPARQL queries through the Query Service.

A simplified view of Wikibase architecture is depicted in figure 3.3 [14].

Wikibase data model: informal introduction

The Wikibase data model [15] is defined as an abstract data model that can have different serializations like JSON and RDF. It is defined using UML data structures and a notation called Wikidata Object Notation.

[9] https://www.wikidata.org/wiki/Wikidata:Statistics
[10] https://wikiba.se/
[11] https://rhizome.org/about/
[12] https://enslaved.org/
[13] https://blazegraph.com/
[14] A more in-depth view of Wikibase architecture can be found at https://addshore.com/2018/12/wikidata-architecture-overview-diagrams/
[15] https://www.mediawiki.org/wiki/Wikibase/DataModel

Informally, the Wikibase data model is formed from entities and statements about those entities. An entity can either be an item or a property. An item is usually represented using a `Q` followed by a number and can represent any thing like an abstract of concrete concept. For example, Q80 represents Tim Berners-Lee in Wikidata.

A property is usually represented by a `P` followed by a number and represents a relationship between an item and a value. For example, P19 represents the property *place of birth* in Wikidata.

The values that can be associated to a property are constrained to belong to some specific datatype. There can be compound datatypes like geographical coordinates.

The current list of datatypes are: quantities, dates and times, geographic locations, geographic shapes, web resources, items, properties, media, non-translated strings, monolingual and multilingual texts/multitexts.

A statement consists of:

- A property which is usually denoted using a `P` followed by a number.
- A declaration about the possible value (in wikibase terms, it is called a *snak*) which can be a specific value, `no value` declaration or a `some value` declaration.
- A rank declaration which can be either `preferred`, `normal` or `deprecated`.
- Zero or more qualifiers which consist of a list of property-value pairs
- Zero or more references which consist of a list of property-value pairs.

Wikibase data model: formal definition

We define a formal model for Wikibase which is based on Multi-Attributed Relational Structures (MARS) [47]. For brevity, we model both qualifiers and references as attributes and don't handle the no-value and some-value snaks.

Definition 3.4.4 — Wikibase graphs. Given a mutually disjoint set of items $\mathcal{Q}$, a set of properties $\mathcal{P}$ and a set of data values $\mathcal{D}$, a *Wikibase graph* is a tuple $\langle \mathcal{Q}, \mathcal{P}, \mathcal{D}, \rho \rangle$ such that $\rho \subseteq \mathcal{E} \times \mathcal{P} \times \mathcal{V} \times \mathtt{FinSet}(\mathcal{P} \times \mathcal{V})$ where $\mathcal{E} = \mathcal{Q} \cup \mathcal{P}$ is the set of entities which can be subjects of a statement and $\mathcal{V} = \mathcal{E} \cup \mathcal{D}$ is the set of possible values of a property.

In practice, Wikibase graphs also add the constraint that every item $q \in \mathcal{Q}$ (or property $p \in \mathcal{P}$) has a unique integer identifier $q^i \in \mathbb{N}$ ($p^i \in \mathbb{N}$).

In the Wikibase data model, statements contain a list of properties-values and the values can themselves be nodes from the graph. This is different from property graphs, where the set of vertices and the set of values are disjoint.

■ **Example 3.4 — Running example as a Wikibase graph.** We continue with the running example about Tim Berners-lee, but extend it with more information about awards. More concretely, we add the information that Tim Berners-Lee was awarded with the *Princess of Asturias* ($\mathtt{PA}$) award together with Vinton Cerf ($\mathtt{vintCerf}$) [16], and that the country of that award is Spain:

[16] The award was really obtained by Tim Berners-Lee, Vinton Cerf, Robert Kahn and Lawrence Roberts, we included here only the first two for simplicity

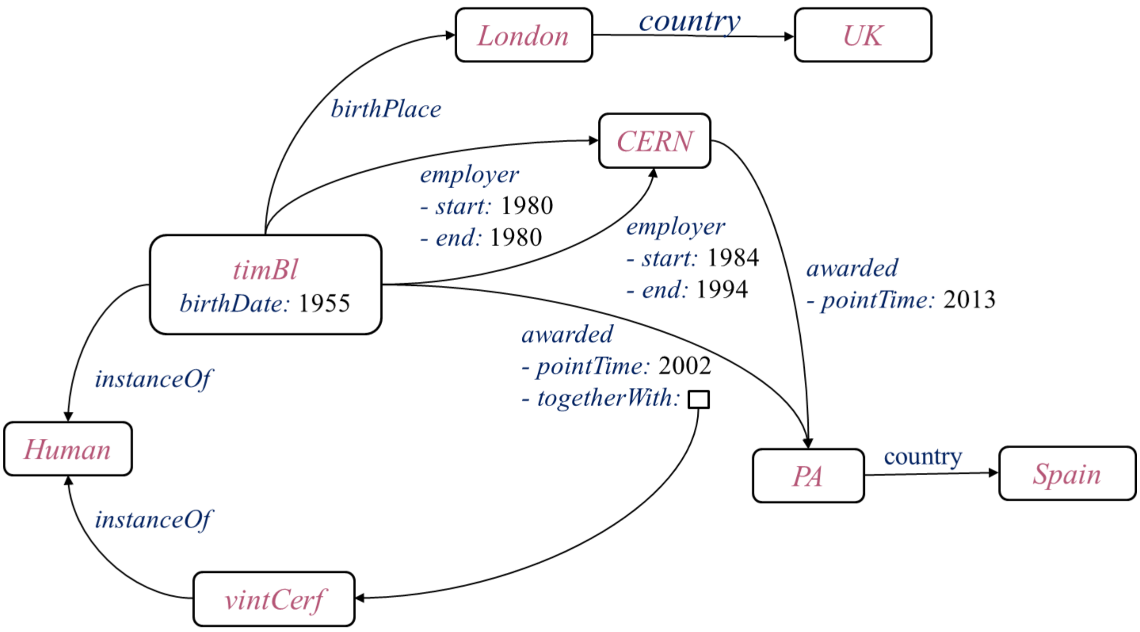

Figure 3.4: Visualization of example wikibase graph

$\mathcal{Q}$ = { timBl, vintCerf, London, CERN, UK, Spain, PA, Human}
$\mathcal{P}$ = { birthDate, birthPlace, country, employer, awarded,
start, end, pointTime, togetherWith, instanceOf}
$\mathcal{D}$ = { 1984,1994,1980,1955}
ρ = { (timBl, instanceOf, Human, { }),
(timBl, birthDate, 1955, { }),
(timBl, birthPlace, London, { }),
(timBl, employer, CERN, { start:1980, end:1980 }),
(timBl, employer, CERN, { start:1984, end:1994 }),
(timBl, awarded, PA, {pointTime: 2002, togetherWith:vintCerf}),
(London, country, UK, { }),
(vintCerf, instanceOf, Human, { })
(CERN, awarded, PA, { pointTime: 2013 })
(PA, country, Spain, { }) }

Figure 3.4 presents a possible visualization of a wikibase graph.

■

The Wikibase data model supports 2 main export formats: JSON and RDF. The JSON one directly follows the structure of the Wikibase data model and is employed by the JSON Dumps while the RDF serialization follows semantic web and linked data principles.

Wikibase JSON serialization

The JSON serialization follows the Wikibase data model. It basically consists of an array of entities where each entity is a JSON object that captures all the local information about the entity: the labels, descriptions, aliases, sitelinks and statements that have the entity as subject. Each JSON object is represented in a single line. A remarkable feature of this encoding is that it captures the output neighbourhood of every entity in a single line making it amenable to processing models that focus on local neighbourhoods because the whole graph can be processed in a single pass.

```
[
 { "type": "item", "id": "Q42", "claims": { "P31": [...
 { "type": "item", "id": "Q80", "claims": { "P108": [...
 { "type": "property", "id": "P108", "claims": { ...
 ...
]
```

Wikibase RDF serialization

The RDF serialization[17] of Wikidata was designed with the goal of being able to represent all the structures of the Wikibase data model in RDF, maintaining compatibility with semantic web vocabularies like RDFS and OWL and avoiding the use of blank nodes [17].

■ **Example 3.5 — RDF serialization of a node.** As an example, the information about Tim Berners-Lee (Q80) declaring that he was as employer (P108) of CERN (Q42944) between 1984 and 1994 is represented as [18]:

```
wd:Q80 rdf:type wikibase:Item ;
 wdt:P108 wd:Q42944     ;
 p:P108  :Q80-4fe7940f   .

:Q80-4fe7940f rdf:type wikibase:Statement ;
 wikibase:rank wikibase:NormalRank ;
 ps:P108       wd:Q42944 ;
 pq:P580       "1984-01-01T00:00:00Z"^^xsd:dateTime ;
 pq:P582       "1994-01-01T00:00:00Z"^^xsd:dateTime .
```

The RDF serialization uses a direct arc to represent the preferred statement represented by prefix alias `wdt:` leaving the rest of the values of a property accessible through the namespaces `p:`, `ps:` and `pq:`.

The reification model employed by Wikidata creates auxiliary nodes that represent each statement. In the previous example, the node `:Q80-4fe7940f` represents the statement which can be qualified with the start and end time.

■

Apart of the the dumps, RDF serialization is also employed by the Wikidata Query Service [4, 39] and users of Wikidata are required to use and understand the singleton statement approach and namespace conventions employed.

Add an example SPARQL query to obtain Timberners-Lee employers and start-end dates

We could define how we can convert Wikibase graphs to RDF graphs

[17] `https://www.mediawiki.org/wiki/Wikibase/Indexing/RDF_Dump_Format`
[18] The full Turtle serialization can be obtained at: `https://www.wikidata.org/wiki/Special:EntityData/Q80.ttl`

3.5 Describing Knowledge Graphs

3.5.1 Describing and validating RDF

At the end of 2013, an *RDF Validation Workshop* [19] was organized by W3C/MIT to discuss use cases and requirements related with the quality of RDF data. One of the conclusions of the workshop was that there was a need for a high-level language that could describe and validate RDF data.

Shape Expressions (ShEx) were proposed as such a language in 2014 [55]. It was designed as a high-level and concise domain-specific language to describe RDF. The syntax of ShEx is inspired by Turtle and SPARQL, while the semantics is inspired by RelaxNG and XML Schema.

In this section we describe a simplified abstract syntax of ShEx following [6][20].

Definition 3.5.1 — ShEx schema. A *ShEx Schema* is defined as a tuple $\langle \mathcal{L}, \delta \rangle$ where $\mathcal{L}$ set of shape labels, and $\delta : \mathcal{L} \rightarrow \mathcal{S}$ is a total function from labels to shape expressions.

The set of shape expressions $se \in \mathcal{S}$ is defined using the following abstract syntax:

se	::=	cond	Basic boolean condition on nodes (node constraint)
	|	s	Shape
	|	se_1 AND se_2	Conjunction
	|	se_1 OR se_2	Disjunction
	|	NOT se	Negation
	|	$@l$	Shape label reference for $l \in \mathcal{L}$
s	::=	CLOSED $\{te\}$	Closed shape
	|	$\{te\}$	Open shape
te	::=	$te_1;te_2$	Each of te_1 and te_2
	|	$te_1 \mid te_2$	Some of te_1 or te_2
	|	$te*$	Zero or more te
	|	ϵ	Empty triple expression
	|	$\llcorner \xrightarrow{p} @l$	Triple constraint with predicate p

Intuitively, shape expressions define constraints on nodes while triple expressions define constraints on the neighbourhood of nodes, and shapes qualify those neighbourhoods by disallowing triples with other predicates in the case of closed shapes or allowing them in the case of open shapes.

The restrictions imposed on shape expressions schemas in [56] also apply here. Namely, in a schema $(\mathcal{L}, \delta, \mathcal{S})$

- the shape label references used by the definition function δ are themselves defined, i.e. if $@l$ appears in some shape definition, then l belongs to $\mathcal{L}$;
- no defintion $\delta(l)$ uses a reference $@l$ to itself, neither directly nor transitively, except while traversing a shape. For instance, $\delta(l) = @l$ AND se is forbidden, but $\delta(l) = \{\llcorner \xrightarrow{p} @l\}$ is allowed.

■ **Example 3.6 — Example of ShEx schema.** A ShEx schema that describes the RDF graph presented in example 3.1 can be defined as:

[19] https://www.w3.org/2012/12/rdf-val/

[20] The full specification of ShEx is available at https://shex.io/shex-semantics/

$$
\begin{array}{rcl}
\mathcal{L} & = & \{\ \mathtt{Person}, \mathtt{Place}, \mathtt{Country}, \mathtt{Organization}, \mathtt{Date}\} \\
\delta(\mathtt{Person}) & = & \{\ \ \llcorner \xrightarrow{\mathtt{birthDate}} @\mathtt{Date};\ \llcorner \xrightarrow{\mathtt{birthPlace}} @\mathtt{Place}; \\
 & & \ \ \llcorner \xrightarrow{\mathtt{employer}} @\mathtt{Organization} * \} \\
\delta(\mathtt{Place}) & = & \{\ \ \llcorner \xrightarrow{\mathtt{country}} @\mathtt{Country}\} \\
\delta(\mathtt{Country}) & = & \{\ \ \} \\
\delta(\mathtt{Organization}) & = & \{\ \ \} \\
\delta(\mathtt{Date}) & = & \text{xsd:Date}
\end{array}
$$

■

ShEx has several concrete syntaxes like a compact syntax (ShExC) and an RDF syntax defined based on JSON-LD (ShExJ) [21].

■ **Example 3.7 — Example of ShEx in ShExC compact syntax.** The previous ShEx schema can be defined using the compact syntax as:

```
:Person {
  :birthPlace  @:Place ;
  :birthDate   @:Date ;
  :employer    @:Organization ;
}
:Place {
  :country     @:Country
}
:Country       {}
:Organization  {}
:Date          {}
```

when rdfshape again...

In general, it is possible to visualize ShEx schemas using UML-like class diagrams. Figure 3.5 presents a visualization of the previous schema using RDFShape [22]

■

Apart of describing RDF data, Shape Expressions have been designed to enable validation and checking if an RDF node conforms to some shape.

The semantics of Shape Expression validation can be defined with a relation between an RDF node, an RDF graph, a ShEx schema and a shape assignment.

As an example of validation, we have implemented the ShEx-s library which is used by RDFShape [23].

The semantics of ShEx schemas is based on a conformance relation parameterized by a shape assignment: we say that node n in graph $\mathcal{G}$ conforms to shape expression se with shape assignment τ, and we write $\mathcal{G}, n, \tau \vDash se$.

The following rules are defined similar [5], where it is shown that there exists a unique maximal shape assignment τ_{max} that allows to define conformance independently on the shape assignment.

The conformance relation is defined recursively on the structure of se by the set of inference rules presented below.

the change of open shape

the order of ...e rules...

[21] See [56] for details.

[22] This visualization can be interactively generated following: `https://rdfshape.weso.es/link/16275431752`

[23] It is possible to see the results of validating the previous example in RDFShape following this link: https://rdfshape.weso.es/link/16275436158

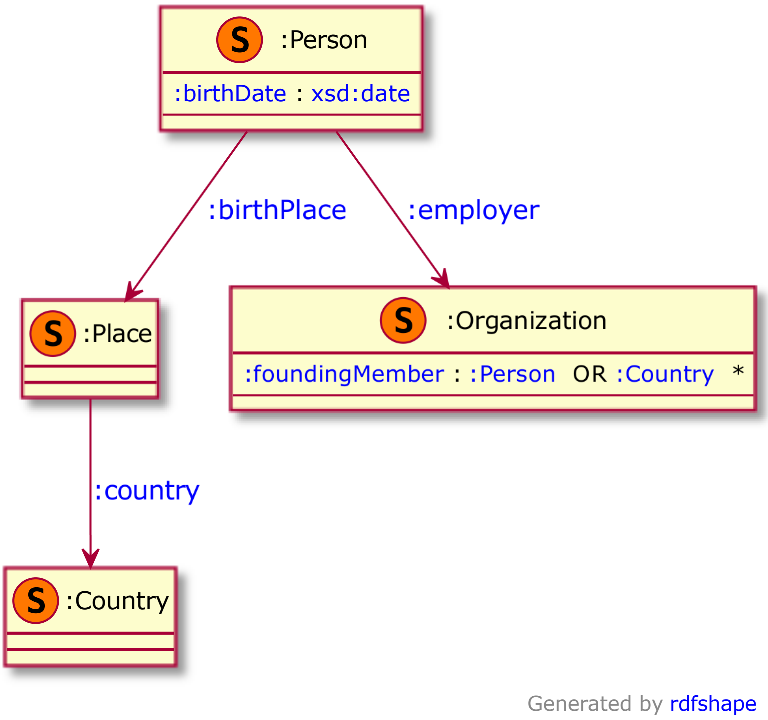

Figure 3.5: ShEx schema visualization as UML-like diagrams

$$\text{Cond}\frac{\mathtt{cond}(n) = \mathtt{true}}{\mathcal{G}, n, \tau \vDash \mathtt{cond}} \qquad \text{AND}\frac{\mathcal{G}, n, \tau \vDash se_1 \quad \mathcal{G}, n, \tau \vDash se_2}{\mathcal{G}, n, \tau \vDash se_1 \ \mathtt{AND}\ se_2}$$

$$\text{OR}_1\frac{\mathcal{G}, n, \tau \vDash se_1}{\mathcal{G}, n, \tau \vDash se_1 \ \mathtt{OR}\ se_2} \qquad \text{OR}_2\frac{\mathcal{G}, n, \tau \vDash se_2}{\mathcal{G}, n, \tau \vDash se_1 \ \mathtt{OR}\ se_2}$$

$$\text{NOT}\frac{\mathcal{G}, n, \tau \nvDash se}{\mathcal{G}, n, \tau \vDash \ \mathtt{NOT}\ se} \qquad \text{ClosedShape}\frac{\mathtt{neighs}(n, \mathcal{G}) = ts \quad \mathcal{G}, ts, \tau \Vdash te}{\mathcal{G}, n, \tau \vDash \mathtt{CLOSED}\ \{te\}}$$

$$\text{OpenShape}\frac{ts = \{\langle x, p, y\rangle \in \mathtt{neighs}(n, \mathcal{G}) \mid p \in \mathtt{preds}(te)\} \quad \mathcal{G}, ts, \tau \Vdash te}{\mathcal{G}, n, \tau \vDash \{te\}}$$

where $\mathtt{preds}(te)$ is the set of predicates that appear in a triple expression te and can be defined as:

$$\begin{array}{lcl}
\mathtt{preds}(te_1;te_2) & = & \mathtt{preds}(te_1) \cup \mathtt{preds}(te_2) \\
\mathtt{preds}(te_1 \mid te_2) & = & \mathtt{preds}(te_1) \cup \mathtt{preds}(te_2) \\
\mathtt{preds}(_ \xrightarrow{p} te) & = & \{p\} \\
\mathtt{preds}(te*) & = & \mathtt{preds}(te) \\
\mathtt{preds}(\epsilon) & = & \emptyset
\end{array}$$

The rules for node constraint (Cond), conjunction, disjunction, and negation, are as expected. A node n conforms to an open shape with triple expression te if its neighbourhood restricted to the triples with predicates from te conform, meaning that triples whose predicates are not mentioned in te are not constrained by the shape (rule OpenShape). Conformance to a closed shape requires to consider the whole neighbourhood of the node (rule ClosedShape).

Conformance to a shape uses a second conformance relation defined on sets on neighbourhood triples ts instead of nodes n. The set of neighbourhood nodes ts of a graph $\mathcal{G}$ conforms to a triple expression te with shape assignment τ, written as $\mathcal{G}, ts, \tau \Vdash te$, as defined by the following inference rules recursively on the structure of $\mathtt{te}$.

$$\mathtt{EachOf}\,\frac{(\mathtt{ts}_1,\mathtt{ts}_2) \in \mathtt{part}(\mathtt{ts}) \qquad \mathcal{G},\mathtt{ts}_1,\tau \Vdash \mathtt{te}_1 \qquad \mathcal{G},\mathtt{ts}_2,\tau \Vdash \mathtt{te}_2}{\mathcal{G},\mathtt{ts},\tau \Vdash \mathtt{te}_1;\mathtt{te}_2}$$

$$\mathtt{OneOf}_1\,\frac{\mathcal{G},\mathtt{ts},\tau \Vdash \mathtt{te}_1}{\mathcal{G},\mathtt{ts},\tau \Vdash \mathtt{te}_1 \mid \mathtt{te}_2} \qquad \mathtt{OneOf}_2\,\frac{\mathcal{G},\mathtt{ts},\tau \Vdash \mathtt{te}_2}{\mathcal{G},\mathtt{ts},\tau \Vdash \mathtt{te}_1 \mid \mathtt{te}_2}$$

$$\mathtt{TripleConstraint}\,\frac{\mathtt{ts} = \{\langle x,p,y\rangle\} \qquad \mathcal{G},y,\tau \vDash @l}{\mathcal{G},\mathtt{ts},\tau \Vdash \text{\textvisiblespace} \xrightarrow{p} @l} \qquad \mathtt{Star}_1\,\frac{}{\mathcal{G},\emptyset,\tau \Vdash \mathtt{te}*}$$

$$\mathtt{Star}_2\,\frac{(\mathtt{ts}_1,\mathtt{ts}_2) \in \mathtt{part}(\mathtt{ts}) \qquad \mathcal{G},\mathtt{ts}_1,\tau \Vdash \mathtt{te} \qquad \mathcal{G},\mathtt{ts}_2,\tau \Vdash \mathtt{te}*}{\mathcal{G},\mathtt{ts},\tau \Vdash \mathtt{te}*}$$

We are now ready to define the semantics of ShEx schemas independently on shape assignments. A shape assignment τ for graph $\mathcal{G}$ and $\mathcal{S}$ is called *valid* if for every node n in $\mathcal{G}$ and every shape expression label l defined in $\mathcal{S}$, if $n@l \in \tau$, then $\mathcal{G},n,\tau \vDash @l$.

Lemma 3.5.1 — Boneva et al (6). For every graph $\mathcal{G}$, there exists a unique maximal valid shape shape assignment τ_{max} such that if τ is a valid shape assignment for $\mathcal{G}$ and $\mathcal{S}$, then $\tau \subseteq \tau_{max}$.

out ShEx-*?

3.5.2 Describing and validating Property graphs

In this section we define a ShEx extension called PShEx that can be used to describe and validate Property graphs. According to the definition of property graphs, nodes and edges can have associated labels as well as a set of property/values. In this way, it is necessary to adapt the definition of ShEx to describe pairs or property/values that we will call qualifiers.

The language PShEx is composed of three main categories: shape expressions ($\mathtt{se}$) that describe the shape of nodes, triple expressions ($\mathtt{te}$) that describe the shape of edge relationships and qualifier expressions ($\mathtt{qs}$) that describe qualifiers sets of property/values associated with node/edge identifiers.

Definition 3.5.2 — PShEx schema. A *PShEx Schema* is a tuple $\langle \mathcal{L}, \delta \rangle$ where $\mathcal{L}$ set of shape labels, and $\delta : \mathcal{L} \rightarrow \mathcal{S}$ is a total function from labels to shape expressions $\mathtt{se} \in \mathcal{S}$ defined using the abstract syntax:

$$
\begin{array}{lcll}
se & ::= & cond_{t_s} & \text{Basic boolean condition on set of types } t_s \subseteq \mathcal{T} \\
 & \mid & s & \text{Shape} \\
 & \mid & se_1 \texttt{ AND } se_2 & \text{Conjunction} \\
 & \mid & se_1 \texttt{ OR } se_2 & \text{Disjunction} \\
 & \mid & \texttt{NOT } se & \text{Negation} \\
 & \mid & @l & \text{Shape label reference for } l \in \mathcal{L} \\
 & \mid & qs & \text{Qualifiers of that node} \\
s & ::= & \texttt{CLOSED } \{te\} & \text{Closed shape} \\
 & \mid & \{te\} & \text{Open shape} \\
te & ::= & te_1 ; te_2 & \text{Each of } te_1 \text{ and } te_2 \\
 & \mid & te_1 \mid te_2 & \text{Some of } te_1 \text{ or } te_2 \\
 & \mid & te* & \text{Zero or more } te \\
 & \mid & \llcorner \xrightarrow{p} @l\ qs & \text{Triple constraint with property type } p \\
 & & & \text{whose nodes satisfy the shape } l \text{ and qualifiers } qs \\
qs & ::= & \lfloor ps \rfloor & \text{Open qualifier specifiers } ps \\
 & \mid & \lceil ps \rceil & \text{Closed qualifier specifiers } ps \\
ps & ::= & ps_1 , ps_2 & \text{Each of } ps_1 \text{ and } ps_2 \\
 & \mid & ps_1 \mid ps_2 & \text{OneOf of } ps_1 \text{ or } ps_2 \\
 & \mid & ps* & \text{zero of more } ps \\
 & \mid & p : cond_v & \text{Property } p \text{ with value conforming to } cond_v \\
 & & & cond_{v_s} \text{ is a boolean condition on sets of values } v_s \subseteq \mathcal{V}
\end{array}
$$

We will omit the list of qualifiers when it is empty.

■ **Example 3.8** As an example, we can define a PShEx schema that describes the property graph from example 3.3 where $hasType_t$ is a condition that is satisfied when the set of types of a node contains the type t, i.e. $hasType_t(vs) = \texttt{true}$ if $t \in vs$ and $String, Date$ are conditions on the values that are satisfied when the values have the corresponding type.

$$
\begin{array}{lcll}
\mathcal{L} & = & \{ & Person, Place, Country, Org\} \\
\delta(\text{Person}) & = & & hasType_{Human} \texttt{ AND } \lfloor label : String, birthDate : Date \rfloor \texttt{ AND } \{ \\
 & & & \llcorner \xrightarrow{birthPlace} @Place; \\
 & & & \llcorner \xrightarrow{employer} @Org\ \lfloor start : Date, end : Date \rfloor * \\
 & & \} & \\
\delta(\text{Place}) & = & & \lfloor label : String \rfloor \texttt{ AND } \{ \\
 & & & \llcorner \xrightarrow{country} @Country \\
 & & \} & \\
\delta(\text{Country}) & = & & hasType_{Country} \texttt{ AND } \lfloor label : String \rfloor \{\} \\
\delta(\text{Org}) & = & & hasType_{Organization} \texttt{ AND } \lfloor label : String \rfloor \{\}
\end{array}
$$

■

In order to define the semantic specification of PShEx we will need to define the neighbourhood of a node in a property graph.

Definition 3.5.3 — Neighbourhood of node in property graph. The neighbors of a node $n \in \mathcal{N}$ in a property graph $\mathcal{G} = \langle \mathcal{N}, \mathcal{E}, \rho, \lambda_n, \lambda_e, \sigma \rangle$ are defined as $neighs(n) = \{(n, p, y, vs) \mid \exists v \in \mathcal{E}$ such that $\rho(v) = (n, y) \wedge \lambda_e(v) = p \wedge vs = \{(k, v) \mid \sigma(k, v) = ws \wedge v \in ws\}\}$

■ **Example 3.9** The neighbors of node n_1 in property graph 3.3 are:

$$
\begin{array}{lll}
neighs(n_1) & = \{ & (n_1, birthPlace, n_2, \{\}), \\
 & & (n_1, employer, n_4, \{(start, 1980), (end, 1980)\}), \\
 & & (n_1, employer, n_4, \{(start, 1984), (end, 1994)\})\}
\end{array}
$$

■

The semantic specification of PShEx can defined in a similar way to the ShEx one. Given a property graph $\mathcal{G}$, and a shape assignment τ, a node identifier $n \in \mathcal{N}$ conforms with a shape expression se, which is represented as $\mathcal{G}, n, \tau \vDash se$ according to the following rules:

$$\text{Cond}_{ts}\frac{\lambda_n(n) = vs \qquad cond_{ts}(vs) = true}{\mathcal{G}, n, \tau \vDash cond_{ts}} \qquad \text{AND}\frac{\mathcal{G}, n, \tau \vDash se_1 \qquad \mathcal{G}, n, \tau \vDash se_2}{\mathcal{G}, n, \tau \vDash se_1 \texttt{ AND } se_2}$$

$$\text{OR}_1\frac{\mathcal{G}, n, \tau \vDash se_1}{\mathcal{G}, n, \tau \vDash se_1 \texttt{ OR } se_2} \qquad \text{OR}_2\frac{\mathcal{G}, n, \tau \vDash se_2}{\mathcal{G}, n, \tau \vDash se_1 \texttt{ OR } se_2}$$

$$\text{NOT}\frac{\mathcal{G}, n, \tau \nvDash se}{\mathcal{G}, n, \tau \vDash \texttt{ NOT } se} \qquad \text{ClosedShape}\frac{neighs(n, \mathcal{G}) = ts \qquad \mathcal{G}, ts, \tau \Vdash s'}{\mathcal{G}, n, \tau \vDash \texttt{CLOSED}\,\{te\}}$$

$$\text{OpenShape}\frac{ts = \{\langle x, p, y\rangle \in neighs(n, \mathcal{G}) \mid p \in preds(te)\} \qquad \mathcal{G}, ts, \tau \Vdash te}{\mathcal{G}, n, \tau \vDash \{te\}}$$

where $preds(te)$ is the set of edge labels (or predicates) that appear in a triple expression te and can be defined as:

$$\begin{array}{lcl} preds(te_1;te_2) & = & preds(te_1) \cup preds(te_2) \\ preds(te_1 \mid te_2) & = & preds(te_1) \cup preds(te_2) \\ preds(\textvisiblespace \xrightarrow{p} te) & = & \{p\} \\ preds(te*) & = & preds(te) \\ preds(\epsilon) & = & \emptyset \end{array}$$

As in the case of ShEx, the previous definition uses a second conformance relation defined on sets on triples ts instead of nodes n. The set of neighbourhood nodes ts from a property graph $\mathcal{G}$ conforms to a triple expression te with shape assignment τ, written $\mathcal{G}, ts, \tau \Vdash s$, as defined by the following inference rules.

$$\text{EachOf}\frac{(ts_1, ts_2) \in part(ts) \qquad \mathcal{G}, ts_1, \tau \Vdash te_1 \qquad \mathcal{G}, ts_2, \tau \Vdash te_2}{\mathcal{G}, ts, \tau \Vdash te_1;te_2}$$

$$\text{OneOf}_1\frac{\mathcal{G}, ts, \tau \Vdash te_1}{\mathcal{G}, ts, \tau \Vdash te_1 \mid te_2} \qquad \text{OneOf}_2\frac{\mathcal{G}, ts, \tau \Vdash te_2}{\mathcal{G}, ts, \tau \Vdash te_1 \mid te_2}$$

$$\text{TripleConstraint}\frac{ts = \{\langle x, p, y, s\rangle\} \qquad \mathcal{G}, y, \tau \vDash @l \qquad \mathcal{G}, s, \tau \vdash qs}{\mathcal{G}, ts, \tau \Vdash \textvisiblespace \xrightarrow{p} @l\ qs}$$

$$\text{Star}_1\frac{}{\mathcal{G}, \emptyset, \tau \Vdash te*}$$

$$\text{Star}_2\frac{(ts_1, ts_2) \in part(ts) \qquad \mathcal{G}, ts_1, \tau \Vdash te \qquad \mathcal{G}, ts_2, \tau \Vdash te*}{\mathcal{G}, ts, \tau \Vdash te*}$$

We declare a conformance relationship $\mathcal{G}, s, \tau \vdash qs$ between a graph $\mathcal{G}$ a set $s \in P \times V$ of property-value elements, a shape assignment τ and a qualifier specifier qs.

$$\text{OpenQs}\ \frac{s' = \{(p,v) \in s | p \in \mathtt{props}(ps)\} \quad \mathcal{G}, s', \tau \vdash ps}{\mathcal{G}, s, \tau \vdash \lfloor ps \rfloor} \qquad \text{CloseQs}\ \frac{\mathcal{G}, s, \tau \vdash ps}{\mathcal{G}, s, \tau \vdash \lceil ps \rceil}$$

$$\text{EachOfQs}\ \frac{\mathcal{G}, s, \tau \vdash ps_1 \quad \mathcal{G}, s, \tau \vdash ps_2}{\mathcal{G}, s, \tau \vdash ps_1\, ,\, ps_2}$$

$$\text{OneOfQs}_1\ \frac{\mathcal{G}, s, \tau \vdash ps_1}{\mathcal{G}, s, \tau \vdash ps_1 \mid ps_2} \qquad \text{OneOfQs}_2\ \frac{\mathcal{G}, s, \tau \vdash ps_2}{\mathcal{G}, s, \tau \vdash ps_1 \mid ps_2}$$

$$\text{StarQs}_1\ \frac{}{\mathcal{G}, \emptyset, \tau \vdash ps*} \qquad \text{StarQs}_2\ \frac{(s_1, s_2) \in \mathtt{part}(s) \quad \mathcal{G}, s_1, \tau \vdash ps \quad \mathcal{G}, s_2, \tau \vdash ps*}{\mathcal{G}, s, \tau \vdash ps*}$$

$$\text{PropertyQs}\ \frac{s = \{(p,w)\} \quad conv_v(w) = \mathtt{true}}{\mathcal{G}, s, \tau \vdash p : cond_v}$$

where propsps is the set of properties that appear in a property specifier ps and can be defined as:

$$\begin{array}{lcl} \mathtt{props}(ps_1\, ,\, ps_2) & = & \mathtt{props}(ps_1) \cup \mathtt{props}(ps_2) \\ \mathtt{props}(ps_1 \mid ps_2) & = & \mathtt{props}(ps_1) \cup \mathtt{props}(ps_2) \\ \mathtt{props}(ps*) & = & \mathtt{preds}(ps) \\ \mathtt{props}(p : cond_v) & = & \{p\} \end{array}$$

As in the case of ShEx, the semantics of ShEx schemas can be defined independently on shape assignments. A shape assignment τ for graph $\mathcal{G}$ and $\mathcal{S}$ is called *valid* if for every node n in $\mathcal{G}$ and every shape expression label l defined in $\mathcal{S}$, if $n@l \in \tau$, then $\mathcal{G}, n, \tau \models @l$.

3.5.3 Describing and validating Wikibase graphs

In this section we present an extension of the ShEx language presented in section 3.5.1 adapted to the wikibase graphs definitions 3.4.4 that we call WShEx.

Definition 3.5.4 — WShEx schema. A *WShEx Schema* is defined as a tuple $\langle \mathcal{L}, \delta \rangle$ where $\mathcal{L}$ set of shape labels, and $\delta : \mathcal{L} \rightarrow \mathcal{S}$ is a total function from labels to w-shape expressions.

The set of w-shape expressions $se \in \mathcal{S}$ is defined using the following abstract syntax. Notice that it is an extension of the abstract syntax for ShEx modifying the rule for triple constraint adding a new element for qualifier specifiers and adding the corresponding rules for qualifier specifiers. The new rules are [24]:

[24] We present the whole abstract syntax in Annex **??**

se	$::=$	$cond$	Basic boolean condition on nodes (node constraint)
	$\mid$	s	Shape
	$\mid$	se_1 `AND` se_2	Conjunction
	$\mid$	se_1 `OR` se_2	Disjunction
	$\mid$	`NOT` se	Negation
	$\mid$	$@l$	Shape label reference for $l \in \mathcal{L}$
s	$::=$	`CLOSED` s'	Closed shape
	$\mid$	s'	Open shape
s'	$::=$	$\{\ te\ \}$	Shape definition
te	$::=$	$te_1 ; te_2$	Each of te_1 and te_2
	$\mid$	$te_1 \mid te_2$	Some of te_1 or te_2
	$\mid$	$te*$	Zero or more te
	$\mid$	$\llcorner \xrightarrow{p} @l\ qs$	Triple constraint with predicate p value conforming to l and qualifier specifier qs
	$\mid$	ϵ	Empty triple expression
qs	$::=$	$\lfloor ps \rfloor$	Open property specifier
	$\mid$	$\lceil ps \rceil$	Closed property specifier
ps	$::=$	$ps\ ,ps$	*EachOf* property specifiers
	$\mid$	$ps \mid ps$	*OneOf* property specifiers
	$\mid$	$ps*$	zero of more property specifiers
	$\mid$	ϵ	Empty property specifier
	$\mid$	$p{:}@l$	Property p with value conforming to shape l

■ **Example 3.10 — Example of WShEx schema.** A ShEx schema that describes the RDF graph presented in example 3.1 can be defined as:

$$
\begin{aligned}
\mathcal{L} &= \{\ \mathsf{Person}, \mathsf{Place}, \mathsf{Country}, \mathsf{Organization}, \mathsf{Date}, \mathsf{Award}\} \\
\delta(\mathsf{Person}) &= \{\ \llcorner \xrightarrow{\mathsf{birthDate}} @\mathsf{Date};\ \llcorner \xrightarrow{\mathsf{birthPlace}} @\mathsf{Place}; \\
&\qquad \llcorner \xrightarrow{\mathsf{employer}} @\mathsf{Organization}\ \lfloor \mathsf{start} : @\mathsf{Date}, \mathsf{end} : @\mathsf{Date} \rfloor * \\
&\qquad \llcorner \xrightarrow{\mathsf{awarded}} @\mathsf{Award}\ \lfloor \mathsf{pointTime} : @\mathsf{Date}, \mathsf{togetherWith} : @\mathsf{Person} \rfloor * \\
&\quad \} \\
\delta(\mathsf{Place}) &= \{\ \llcorner \xrightarrow{\mathsf{country}} @\mathsf{Country}\} \\
\delta(\mathsf{Country}) &= \{\ \} \\
\delta(\mathsf{Award}) &= \{\ \llcorner \xrightarrow{\mathsf{country}} @\mathsf{Country}\} \\
\delta(\mathsf{Organization}) &= \{\ \} \\
\delta(\mathrm{Date}) &= \ \in \mathsf{xsd : date}
\end{aligned}
$$

■

The compact syntax for WShEx is similar to ShExC adding the symbols `{{...}}` to declare open qualifier specifiers and `[[...]]` for closed ones.

■ **Example 3.11 — Example of WShEx schema using the compact Syntax.**

```
:Researcher {
 birthPlace      @<Place>           ;
 birthDate       @<Time>            ;
 employer        @<Organization> *
             {{ :start    @:Date ,
                :end      @:Date
             }} ;
 awarded        @<Award> *
             {{ :pointTime     @:Date ,
                :togetherWith  @:Person
             }}
```

```
}
:Place          { country @<Country> }
:Organization   {}
:Award          { country @<Country> }
:Country        {}
:Date           xsd:date
```

■

The semantics of WShEx is based on the semantics of ShEx (see 3.5.1). We present here the changes to the semantics which affect only to the rule that declares conformance to triple constraints. It takes now into account qualifier specifiers [25]. We declare a new conformance relationship $\mathcal{G}, s, \tau \vdash qs$ between a graph $\mathcal{G}$ a set $s \in P \times V$ of property-value elements, a shape assignment τ and a qualifier specifier qs.

$$\text{TripleConstraint}\frac{ts = \{\langle x, p, y, s\rangle\} \quad \mathcal{G}, y, \tau \vDash @l \quad \mathcal{G}, s, \tau \vdash qs}{\mathcal{G}, ts, \tau \Vdash \_ \xrightarrow{p} @l\ qs}$$

$$\text{OpenQs}\frac{s' = \{(p, v) \in s | p \in preds(ps)\} \quad \mathcal{G}, s', \tau \vdash ps}{\mathcal{G}, s, \tau \vdash \lfloor ps \rfloor} \qquad \text{CloseQs}\frac{\mathcal{G}, s, \tau \vdash ps}{\mathcal{G}, s, \tau \vdash \lceil ps \rceil}$$

$$\text{EachOfQs}\frac{\mathcal{G}, s, \tau \vdash ps_1 \quad \mathcal{G}, s, \tau \vdash ps_2}{\mathcal{G}, s, \tau \vdash ps_1\,,\, ps_2}$$

$$\text{OneOfQs}_1\frac{\mathcal{G}, s, \tau \vdash ps_1}{\mathcal{G}, s, \tau \vdash ps_1 \mid ps_2} \qquad \text{OneOfQs}_2\frac{\mathcal{G}, s, \tau \vdash ps_2}{\mathcal{G}, s, \tau \vdash ps_1 \mid ps_2}$$

$$\text{StarQs}_1\frac{}{\mathcal{G}, \emptyset, \tau \vdash ps*} \qquad \text{StarQs}_2\frac{(s_1, s_2) \in part(s) \quad \mathcal{G}, s_1, \tau \vdash ps \quad \mathcal{G}, s_2, \tau \vdash ps*}{\mathcal{G}, s, \tau \vdash ps*}$$

$$\text{EmptyQs}\frac{}{\mathcal{G}, \emptyset, \tau \vdash \epsilon} \qquad \text{PropertyQs}\frac{s = \{(p, v)\} \quad \mathcal{G}, v, \tau \vDash @l}{\mathcal{G}, s, \tau \vdash p : @l}$$

3.6 Knowledge Graphs Subsets

Esta sección es un borrador de ideas todavía...

In this section we will review different approaches to create subsets of knowledge graphs.

We will focus mainly on Wikidata subsets although the approaches could also be applied to RDF-based graphs and property graphs.

3.6.1 Wikibase Subsets: Formal definition

The following definition of Wikibase subset is based on the definition of wikibase graphs given at section 3.4.3.

Definition 3.6.1 — Wikibase subset. Given a wikibase graph $\mathcal{G} = \langle \mathcal{Q}, \mathcal{P}, \mathcal{D}, \mathcal{S}\rangle$, a wikibase subgraph is defined as $\mathcal{G}' = \langle \mathcal{Q}', \mathcal{P}', \mathcal{D}', \mathcal{S}'\rangle$ such that: $\mathcal{Q}' \subseteq \mathcal{Q}$, $\mathcal{P}' \subseteq \mathcal{P}$, $\mathcal{D}' \subseteq \mathcal{D}$ and $\mathcal{S}' \subseteq \mathcal{S}$

■ **Example 3.12 — Example of wikibase subgraph.** Given the wikibase graph from example 3.4

[25] The full inference rules of WShEx semantics are presented in Annex **??**

$\mathcal{G}' = \langle \mathcal{Q}', \mathcal{P}', \mathcal{D}', \mathcal{S}' \rangle$ where

$$\begin{aligned}
\mathcal{Q}' &= \{\mathtt{timBl}, \mathtt{London}, \mathtt{CERN}\}\\
\mathcal{P}' &= \{\mathtt{birthPlace}, \mathtt{employer}, \mathtt{start}\}\\
\mathcal{D} &= \{1980, 1984\}\\
\mathcal{S} &= \{(\mathtt{timBl}, \mathtt{birthPlace}, \mathtt{London}, \{\}),\\
&\quad (\mathtt{timBl}, \mathtt{employer}, \mathtt{CERN}, \{\mathtt{start} : 1980\}),\\
&\quad (\mathtt{timBl}, \mathtt{employer}, \mathtt{CERN}, \{\mathtt{start} : 1984\})\}
\end{aligned}$$

is a wikibase subgraph of $\mathcal{G}$. ■

3.6.2 Generating subgraphs from subsets of items or properties

Wikibase subgraphs can be generated from a set of items or properties, where we collect the subgraph associated with those entities.

Definition 3.6.2 — Item-generated subgraph. Given a wikibase graph $\mathcal{G} = \langle \mathcal{Q}, \mathcal{P}, \mathcal{D}, \mathcal{S} \rangle$ and a subset of items $\mathcal{Q}_s \subset \mathcal{Q}$ generates an *item-generated subgraph* $\langle \mathcal{Q}', \mathcal{P}', \mathcal{D}', \mathcal{S}' \rangle$ such that:

$$\begin{aligned}
\mathcal{Q}' &= \{q \in \mathcal{Q} \mid (q,_,_,_) \vee (_,_,q,_) \in \mathcal{S}'\}\\
&\quad \cup \{q \in \mathcal{Q} \mid (_,_,_,q_s) \in \mathcal{S}' \wedge (_,q) \in q_s\}\\
\mathcal{P}' &= \{p \in \mathcal{P} \mid (_,p,_,_) \in \mathcal{S}'\}\\
&\quad \cup \{p \in \mathcal{P} \mid (_,_,_,q_s) \in \mathcal{S}' \wedge (p,_) \in q_s\}\\
\mathcal{D}' &= \{d \in \mathcal{D} \mid (_,_,d,_) \in \mathcal{S}'\}\\
&\quad \cup \{d \in \mathcal{D} \mid (_,_,_,q_s) \in \mathcal{S}' \wedge (_,d) \in q_s\}\\
\mathcal{S}' &= \{(q,_,_,_) \in \mathcal{S} \mid q \in \mathcal{Q}_s\}\\
&\quad \cup \{(_,_,q,_) \in \mathcal{S} \mid q \in \mathcal{Q}_s\}\\
&\quad \cup \{(_,_,_,q_s) \in \mathcal{S} \wedge \exists q \in \mathcal{Q}_s \mid (_,q) \in q_s\}
\end{aligned}$$

Notice that the item-generated subgraph usually contains more items than the items provided by $\mathcal{Q}_s$.

■ **Example 3.13 — Example of item-generated subgraph.** Given the wikibase graph from example 3.4 and $\mathcal{Q}_s = \{\mathtt{timBl}\}$ the item generated subgraph is:

$$\begin{aligned}
\mathcal{Q}' &= \{\mathtt{timBl}, \mathtt{CERN}, \mathtt{vintCerf}, \mathtt{PA}\}\\
\mathcal{P}' &= \{\mathtt{birthDate}, \mathtt{birthPlace}, \mathtt{employer}, \mathtt{awarded},\\
&\quad \mathtt{start}, \mathtt{end}, \mathtt{togetherWith}\}\\
\mathcal{D}' &= \{1984, 1994, 1980, 1955\}\\
\mathcal{S}' &= \{(\mathtt{timBl}, \mathtt{birthDate}, 1955, \{\}),\\
&\quad (\mathtt{timBl}, \mathtt{birthPlace}, \mathtt{London}, \{\}),\\
&\quad (\mathtt{timBl}, \mathtt{employer}, \mathtt{CERN}, \{\mathtt{start} : 1980, \mathtt{end} : 1980\}),\\
&\quad (\mathtt{timBl}, \mathtt{employer}, \mathtt{CERN}, \{\mathtt{start} : 1984, \mathtt{end} : 1994\}),\\
&\quad (\mathtt{timBl}, \mathtt{awarded}, \mathtt{PA}, \{\mathtt{togetherWith} : \mathtt{vintCerf}\}),\\
&\quad (\mathtt{vintCerf}, \mathtt{awarded}, \mathtt{PA}, \{\mathtt{togetherWith} : \mathtt{timBl}\})\}
\end{aligned}$$

■

Definition 3.6.3 — Property-generated subgraph. Given a wikibase graph $\mathcal{G} = \langle \mathcal{Q}, \mathcal{P}, \mathcal{D}, \mathcal{S} \rangle$, with the set of entities $\mathcal{E} = \mathcal{Q} \cup \mathcal{P}$ a subset of properties $\mathcal{P}_s \subset \mathcal{P}$ generates a *property generated subgraph* $\langle \mathcal{Q}', \mathcal{P}', \mathcal{D}', \mathcal{S}' \rangle$ such that:

$$
\begin{aligned}
\mathcal{Q}' &= \{q \in \mathcal{Q} \mid \exists p \in \mathcal{P}_s \mid (q, p, \_, \_) \in \mathcal{S}\} \\
&\cup \{q \in \mathcal{Q} \mid \exists p \in \mathcal{P}_s \mid (\_, p, q, \_) \in \mathcal{S}\} \\
&\cup \{q \in \mathcal{Q} \mid (\_, \_, \_, q_s) \in \mathcal{S} \wedge \exists p \in \mathcal{P}_s \mid (p, \_) \in q_s\} \\
\mathcal{P}' &= \{p \in \mathcal{P}_s \mid (\_, p, \_, \_) \in \mathcal{S}\} \\
&\cup \{p \in \mathcal{P}_s \mid \exists q_s \mid (\_, \_, \_, qs) \in \mathcal{S} \wedge (p, \_) \in q_s\} \\
\mathcal{D}' &= \{d \in \mathcal{D} \mid \exists p \in \mathcal{P}_s \mid (\_, p, d, \_) \in \mathcal{S}\} \\
&\cup \{d \in \mathcal{D} \mid (\_, \_, \_, q_s) \in \mathcal{S} \wedge \exists p \in \mathcal{P}_s \mid (p, d) \in q_s\} \\
\mathcal{S}' &= \{(\_, p, \_, \_) \in \mathcal{S} \mid p \in \mathcal{P}_s\} \\
&\cup \{(\_, \_, \_, q_s) \in \mathcal{S} \mid \exists p \in \mathcal{P}_s \mid (p, \_) \in q_s\}
\end{aligned}
$$

The property generated subgraph usually contains more properties than the properties provided by $\mathcal{P}_s$.

■ **Example 3.14 — Example of property-generated subgraph.** Given the wikibase graph from example 3.4 and $\mathcal{P}_s = \{\mathtt{birthDate}, \mathtt{togetherWith}\}$ the property generated subgraph is:

$$
\begin{aligned}
\mathcal{Q}' &= \{\mathtt{timBl}, \mathtt{vintCerf}, \mathtt{PA}\} \\
\mathcal{P}' &= \{\mathtt{birthDate}, \mathtt{awarded}, \mathtt{togetherWith}\} \\
\mathcal{D}' &= \{1955\} \\
\mathcal{S}' &= \{(\mathtt{timBl}, \mathtt{birthDate}, 1955, \{\}), \\
&(\mathtt{timBl}, \mathtt{awarded}, \mathtt{PA}, \{\mathtt{togetherWith} : \mathtt{vintCerf}\}), \\
&(\mathtt{vintCerf}, \mathtt{awarded}, \mathtt{PA}, \{\mathtt{togetherWith} : \mathtt{timBl}\})\}
\end{aligned}
$$

■

Notice that it is possible to define a *Datatype-generated subgraph* in a similar way than the previous definitions.

Definition 3.6.4 — Entity-generated subgraph. Given a subset of entities $\mathcal{E}_s \subset \mathcal{Q} \cup \mathcal{P}$, the entity-generated subgraph is defined as the union of the item-generated subgraph with all the items in $\mathcal{E}_s$ and the property-generated subgraph with all the properties in $\mathcal{E}_s$.

3.6.3 Graph matching

Definition 3.6.5 — Matching expression. Given a wikibase graph $\mathcal{G} = \langle \mathcal{Q}, \mathcal{P}, \mathcal{D}, \mathcal{S} \rangle$ where $\mathcal{E} = \mathcal{Q} \cup \mathcal{P}$ and $\mathcal{V} = \mathcal{E} \cup \mathcal{D}$, a matching expression M_s is a set of matchers where each matcher m follows the grammar:

m	$::=$	subject(e)	Subject $e \in \mathcal{E}$
	$\mid$	property(p)	Property $p \in \mathcal{P}$
	$\mid$	value(v)	Value $v \in \mathcal{V}$
	$\mid$	qualifier(p,v)	Qualifier with property $p \in \mathcal{P}$ and value $v \in \mathcal{V}$
	$\mid$	qualifiedProp(p)	Qualifier with property $p \in \mathcal{P}$
	$\mid$	qualifiedValue(v)	Qualifier with value $v \in \mathcal{V}$

■ **Example 3.15 — Example of a matcher expression.** An example of a matcher expression is $M_s = \{\text{property(country)}, \text{qualifiedProp(togetherWith)}\}$ ■

Definition 3.6.6 — Matching-generated subgraph. Given a matching expression M_s over a wikibase graph $\mathcal{G} = \langle \mathcal{Q}, \mathcal{P}, \mathcal{D}, \mathcal{S} \rangle$ we can define the matching-generated subgraph as a wikibase graph $\mathcal{G}' = \langle \mathcal{Q}'\mathcal{P}'\mathcal{D}'\mathcal{S}' \rangle$ such that:

$$
\begin{aligned}
\mathcal{Q}' &= \{q \in \mathcal{Q} \mid (q,\_,\_,\_) \in \mathcal{S}' \cup \{q \in \mathcal{Q} \mid (\_,\_,q,\_) \in \mathcal{S}'\} \\
&\quad \cup \{q \in \mathcal{Q} \mid (\_,\_,\_,q_s) \in \mathcal{S}' \wedge (\_,q) \in q_s\} \\
\mathcal{P}' &= \{p \in \mathcal{P} \mid (\_,p,\_,\_) \in \mathcal{S}' \cup \{p \in \mathcal{P} \mid (\_,\_,\_,q_s) \in \mathcal{S}' \wedge (p,\_) \in q_s\} \\
\mathcal{D}' &= \{d \in \mathcal{D} \mid (\_,\_,d,\_) \in \mathcal{S}'\} \cup \{d \in \mathcal{D} \mid (\_,\_,\_,q_s) \in \mathcal{S}' \wedge (\_,d) \in q_s\} \\
\mathcal{S}' &= \{(q,\_,\_,\_) \in \mathcal{S} \mid \text{subject}(q) \in M_s\} \\
&\quad \cup \{(\_,p,\_,\_) \in \mathcal{S} \mid \text{property}(p) \in M_s\} \\
&\quad \cup \{(\_,\_,v,\_) \in \mathcal{S} \mid \text{value}(v) \in M_s\} \\
&\quad \cup \{(\_,\_,\_,q_s) \in \mathcal{S} \mid \text{qualifier}(p,v) \in M_s \wedge \exists (p,v) \in q_s\} \qquad \cup \{(\_,\_,\_,q_s) \in \mathcal{S} \mid \text{qualifiedProp}(p) \in M
\end{aligned}
$$

■ **Example 3.16 — Example of matching-generated subgraph.** Given the wikibase graph $\mathcal{G}$ of example 3.4 and the matching-expression M_s in example **??**, the matching-generated subgraph of $\mathcal{G}$ from M_s is the wikibase graph $\mathcal{G}' = \langle \mathcal{Q}', \mathcal{P}', \mathcal{D}', \mathcal{S}' \rangle$ such that:

$$
\begin{aligned}
\mathcal{Q}' &= \{\mathtt{PA}, \mathtt{Spain}, \mathtt{London}, \mathtt{UK}, \mathtt{timBl}, \mathtt{vintCerf}\} \\
\mathcal{P}' &= \{\mathtt{country}, \mathtt{awarded}, \mathtt{togetherWith}\} \\
\mathcal{D}' &= \{\} \\
\mathcal{S}' &= \{(\mathtt{timBl}, \mathtt{awarded}, \mathtt{PA}, \{\mathtt{togetherWith} : \mathtt{vintCerf}\}), \\
&\quad (\mathtt{vintCerf}, \mathtt{awarded}, \mathtt{PA}, \{\mathtt{togetherWith} : \mathtt{timBl}\}) \\
&\quad (\mathtt{PA}, \mathtt{country}, \mathtt{Spain}, \{\}) \\
&\quad (\mathtt{London}, \mathtt{country}, \mathtt{UK}, \{\})\}
\end{aligned}
$$

■

This approach is followed by WDumper [26].

It consists of matching each triple with a target specification of the expected contents.

the approach
e running exam-
3.6.4

ShEx Slurping

Keep track of the matched triples while validating.

Create a new semantics of Shape Expressions that generates slurps while validating...

[26] https://github.com/bennofs/wdumper

This approach has been implemented by PyShEx [27] and ShEx.js [28] implementations.

This approach may be difficult to scale as it is validating at the same time that it is doing the slurping process. Validating complexity is high because it may require to check the different partitions of a node neighbourhood.

3.6.5 Flatten ShEx + Slurping

It is possible to identify ShEx subsets that enable faster validation. One such subset that we name flatten ShEx defines triple expressions as follows:

3.6.6 ShEx based traversal using Pregel

Pregel [38] has been proposed as an scalable computational model created by Google to handle large graphs. It is based on Bulk Synchronous Parallel (BSP) model which simplifies parallel programming having different computation and communication phases. Pregel is an iterative algorithm where each phase is called a superstep. Following the lemma *think like a vertex*, it is a vertex-centric abstraction where at each superstep, a vertex executes a user defined function (called vertex program) which can update its status and later sends messages to neighbors along graph edges. Supersteps end with a synchronization barrier that guarantees that messages sent at one superstep are received at the beginning of the next superstep. Vertices may change status between active and inactive and the algorithm terminates when all vertices are inactive and no more messages are sent.

GraphX was proposed in 2014 as a graph processing framework embedded in Apache Spark. Its API includes a variant of Pregel which is used to implement several graph algorithms like PageRank, connected components, triangle counting, etc.

GraphX defines an API for graphs based on RDDs (resilient distributed datasets). An $\mathsf{RDD}[\mathcal{V}]$ is an abstraction of a collection of values of type $\mathcal{V}$ which are immutable and can be partitioned to run data-parallel operations like *map* and *reduce*.

A graph $\mathsf{Graph}[\mathcal{V},\mathcal{E}]$ represents and abstraction of vertices with values of type $\mathcal{V}$ and edges of type $\mathcal{E}$ where internally the vertices are represented as $\mathsf{RDD}[(\mathsf{Id},\mathcal{V})]$, i.e. a collection of a tuple with an Id (a Long value) and a $\mathcal{V}$, and edges are represented as $\mathsf{RDD}[(\mathsf{Id},\mathsf{Id},\mathcal{E})]$, i.e. a triple where the first and second components are the Id of the source and destiny respectively, and the third component is the edge property $p \in \mathcal{E}$. A graph $\mathsf{Graph}[\mathcal{V},\mathcal{E}]$ also provides what is called a *triplets* view which represents edges as collections of triplets of the form $\mathsf{RDD}[(\mathcal{V},\mathcal{E},\mathcal{V})]$. A triplet t will be denoted by the type `Triplet` and provides access to the source vertex (using `t.srcAttr`), the destiny (`t.dstAttr`) and the edge property (`t.attr`).

GraphX provides several built-in operators for graphs[29]. We will use the following in the rest of the paper:

- `mapVertices`(`g`: `Graph`[$\mathcal{V}$,$\mathcal{E}$], `f`: (`Id`,$\mathcal{V}$)$\rightarrow$ $\mathcal{V}$): `Graph`[$\mathcal{V}$,$\mathcal{E}$] maps every pair `(id,v)` in the vertices of `g` to `(id, f(v))`.
- `mapReduceTriples`(`g`:`Graph`[$\mathcal{V}$,$\mathcal{E}$], `m`: ($\mathcal{V}$,$\mathcal{E}$,$\mathcal{V}$)$\rightarrow$[(`Id`,$\mathcal{M}$)], `r`:($\mathcal{M}$,$\mathcal{M}$)$\rightarrow$$\mathcal{M}$):`RDD`[(Id,$\mathcal{M}$)], encodes the two-stage parallel computation process commonly known as mapReduce using the triplets view. It takes as parameters, a grapg `g`, a map function `m` and a reduce function `r`.
  In the first stage it applies the `m` to each triplet in the graph to generate a list of messages that will be sent to the vertices identified a given `id`.
  In the second stage, it groups all the messages sent to a given vertex applying the reduce function `r` to each pair of messages.

[27] `https://github.com/hsolbrig/PyShEx`
[28] `https://github.com/shexjs/shex.js`
[29] **\url {https://spark.apache.org/docs/latest/graphx-programming-guide.html}**.

- `joinVertices`(`g`:`Graph`[$\mathcal{V}$,$\mathcal{E}$], `msgs`:`RDD`[(`Id`, $\mathcal{M}$)], `f`:(`Id`, $\mathcal{V}$,$\mathcal{M}$)$\rightarrow\mathcal{V}$): `Graph`[$\mathcal{V}$,$\mathcal{E}$], joins the collection of messages sent to a the vertices which have a value `(id,m)` with the vertex `v` identified by `id` and replaces that vertex by `f(id,v,m)`.

The GraphX Pregel algorithm is defined iteratively where each iteration is usually called a superstep as follows:

Algorithm 1: Pregel algorithm pseudocode as implemented in GraphX

Input parameters:
- `g`: `Graph`[$\mathcal{V}$,$\mathcal{E}$]
- `initialMsg`: $\mathcal{M}$
- `vProg`: (`Id`,$\mathcal{V}$,$\mathcal{M}$)$\rightarrow\mathcal{V}$
- `sendMsg`: `Triplet`$\rightarrow$[(`Id`,$\mathcal{M}$)]
- `mergeMsg`: ($\mathcal{M}$,$\mathcal{M}$)$\rightarrow\mathcal{M}$

Output: `g`:`Graph`[$\mathcal{V}$,$\mathcal{E}$]

```
g = mapVertices(g,λ(id,v)→vProg(id,v,initialMsg))
msgs = mapReduceTriples(g,sendMsg,mergeMsg)
while size(msgs) > 0 do
    g = joinVertices(g,msgs,vProg)
    msgs = mapReduceTriples(g,sendMsg,mergeMsg)
return g
```

It takes as input a `Graph`[$\mathcal{V}$,$\mathcal{E}$] and the following parameters:

- `initialMsg`: initial message sent to all the vertices
- `vprog` is the vertex program. It is run by each vertex at the beginning of the algorithm using the `initialMsg` and in each superstep using the collected messages sent by the neighbors in the previous superstep.
- `sendMsg` takes as parameter an triplet and returns an iterator with a pair `(id, msg)` where `id` represents the id of the vertex which will receive the message and `msg` represents the message that will be sent.
- `mergeMsg` is a function that defines how to merge 2 messages into one. This function must be associative and commutative, and will be invoked to collect all the messages that are sent to a vertex in each superstep.

We have implemented a ShEx validation algorithm based on the Pregel algorithm. The algorithm assumes that there is a ShEx schema $\langle\mathcal{L},\delta\rangle$ where each label $l\in\mathcal{L}$ identifies a shape expression.

The algorithm annotates each node $n\in\mathcal{V}$ with a status map that represents the validation status with regards to some labels. The new nodes in the graph will be tuples (n,m) where $n\in\mathcal{V}$, and $m:\mathcal{L}\mapsto Status$ associates a status for each shape label.

A $Status$ is defined as:

$Status$	::=	$Undefined$	Default status
	$\mid$	Ok	Node conforms
	$\mid$	$Failed$	Node doesn't conform
	$\mid$	$Pending$	Requested to conform
	$\mid$	$WaitingFor(ds,oks,fs)$	Waiting for some neighbours
			ds = list dependants neighbours
			oks = list of conformant neighbours
			fs = list of non conformant neighbours
			where $ds,oks,failed\in\mathcal{V}\times\mathcal{P}\times\mathcal{L}$

The status can be $Undefined$ if there is no information yet (this is the default value) Ok if the node conforms to the shape identified by l, $Failed$ if it doesn't conform to the shape, $Pending$ if the node has been requested to be validated with that label or $\mathsf{WaitingFor}(ds, oks, failed)$ if the validation of node n depends on the validation of a set of neighbour nodes ds. Each neighbour node is represented by a triple (v, p, l) where v is the neighbour node, p is the property which links n with v, and l is the shape label that the node must conform. During the validation, we may receive information that some of those neighbour nodes have been validated or failed. That information is collected in the set oks which is the set of conforming neighbour nodes and $failed$ is the set of failed neighbour nodes.

A message can be represented as a map which assigns to each label the following requests:

Msg	::=	$\mathsf{Validate}$	Request to validate
	|	$\mathsf{Checked}(oks, fs)$	Some neighbours have been checked
			oks = neighbours that have been checked as conformant
			fs = neighbours that have been checked as non-conformant
			where $oks, fs \in \mathcal{V} \times \mathcal{P} \times \mathcal{L}$
	|	$\mathsf{WaitFor}(ds)$	Request to wait for some neighbours
			where $ds \in \mathcal{V} \times \mathcal{P} \times \mathcal{L}$

The Pregel-ShEx-validation traversal is defined with the following pseudocode.

Algorithm 2: Pregel-based ShEx validation pseudocode

Input parameters:

- `g`: `Graph`$[\mathcal{V}, \mathcal{E}]$
- `initialLabel`: $\mathcal{L}$
- `checkLocal`: $(\mathcal{L}, \mathcal{V}) \to \mathsf{Ok} | \mathsf{Failed} | \mathsf{Pending}(\texttt{Set}[\mathcal{L}])$
- `checkNeighs`: $(\mathcal{L}, \texttt{Bag}[(\mathcal{E}, \mathcal{L})], \texttt{Set}[(\mathcal{E}, \mathcal{L})]) \to \mathsf{Ok} | \mathsf{Failed}$
- `tripleConstraints`: $\mathcal{L} \to \texttt{Set}[(\mathcal{E}, \mathcal{L})]$

Output: `g:Graph`$[(\mathcal{V}, \mathcal{L} \mapsto \text{Status}), \mathcal{E}]$

`gs` = `mapVertices`(`g`, λ(`id`, `v`)$\to$(id, (v, λv$\to\mathsf{Undefined}$)))

`gs` = `pregel`($\mathsf{Validate}$, `gs`, `vProg`, `sendMsg`, `mergeMsg`)

`gs` = `mapVertices`(`gs`, `checkUnsolved`)

return `gs`

def `checkUnsolved`(v,m) = (v,m') where

$$m'(l) = \begin{cases} \texttt{checkNeighs}(l, \emptyset, \emptyset) & \text{if } m(l) = \mathsf{Pending} \\ \texttt{checkNeighs}(l, \text{oks}, \text{fs} \cup \text{ds}) & \text{if } m(l) = \mathsf{WaitingFor}(ds, oks, fs)\} \\ m(l) & \text{otherwise} \end{cases}$$

def `vProg`:(`Id`,$\mathcal{V}$,$\mathcal{M}$)$\to\mathcal{V}$ = ...see 9

The algorithm takes as input the parameters:

- `initialLabel` is the initial shape label that is requested to validate every node in the graph. In Shape Expressions, this label is usually annotated with the `start` keyword.
- `checkLocal` checks if the shape expression associated with a label can validate a node locally. It returns Okif the node validates without further dependencies, Failed, if it doesn't validate, and $\mathsf{Pending}(ls)$ if the validation of the node depends on a list of shape labels ls.
- `checkNeighs` checks if the bag of neighbors of a node matches the regular bag expression associated with the label in the schema.
- `tripleConstraints` returns the list of triple constraints associated with the shape expression indicated by the label.

The algorithm starts by mapping every node to the status which associates any label $l \in \mathcal{L}$ to undefined ($\mathsf{Undefined}$). After that, it runs the iterative Pregel algorithm using the `vProg`, `sendMsg` and `mergeMsg` functions defined as above. Once the Pregel algorithm finishes, it replaces the status

$$\frac{(n,m),l \rightsquigarrow \mathsf{Validate} \quad m(l) = s \in \{\mathsf{Undefined}, \mathsf{Pending}\} \qquad \mathtt{checkLocal}(l,n) = r \in \{\mathsf{Ok}, \mathsf{Failed}\}}{m'(l) = r}$$

$$\frac{(n,m),l \rightsquigarrow \mathsf{Validate} \quad m(l) = r \in \{\mathsf{Undefined}, \mathsf{Pending}\} \qquad \mathtt{checkLocal}(l,n) = \mathsf{Pending}(ls)}{m'(l) = \mathsf{Undefined} \quad m'(l') = \mathsf{Pending}\ \forall l' \in ls}$$

$$\frac{(n,m),l \rightsquigarrow \mathsf{Validate} \quad m(l) = r \in \{\mathsf{Ok}, \mathsf{Failed}\}}{m'(l) = r}$$

$$\frac{(n,m),l \rightsquigarrow \mathsf{Validate} \quad m(l) = \mathsf{WaitingFor}(ds, oks, fs)}{m'(l) = \mathsf{Ok}}$$

$$\frac{(n,m),l \rightsquigarrow \mathsf{Checked}(oks, fs) \quad m(l) = \mathsf{WaitingFor}(ds, oks', fs') \qquad ds \setminus (oks \cup fs) \neq \emptyset}{m'(l) = \mathsf{WaitingFor}(ds, oks \cup oks', fs \cup fs')}$$

$$\frac{(n,m),l \rightsquigarrow \mathsf{Checked}(oks, fs) \quad m(l) = \mathsf{WaitingFor}(ds, oks', fs') \qquad ds \setminus (oks \cup fs) = \emptyset}{m'(l) = \mathtt{checkNeighs}(l, oks \cup oks', fs \cup fs')}$$

Table 3.1: Definition of `vProg` for Pregel-based ShEx validation

of any node that is pending or waiting for some neighbours by a last check based on the current information of the neighbours, assuming that if the node didn't receive information that a pending neighbour has validated, it means that there was no evidence of it's validation, and it failed.

`vProg` changes the status map of a node with regards to a label when it receives a message for that label. It can be defined as:

`vProg`(id,(n,m), msg) = (n,m') where $m'(l) = m(l)$ except for the cases indicated by the following rules:

Figure 3.6 represents a state diagram which shows the different status that a node can have with regards to a shape label. Initially, all nodes have status $\mathsf{Undefined}$ until they get a message request to validate against some label. If it is possible to validate locally those nodes, then they will go directly to the end state which can be Ok or Failed. Otherwise, if their validation depends on the neighbours, they will enter the status $\mathsf{Pending}$ whose nodes are active in the Pregel algorithm and will be activated in the messages generation phase. If they receive a request to wait for some other nodes to be validated, they will go to the state $\mathsf{WaitingFor}(ds, oks, fs)$ which means that they are waiting for the status of the neighbour nodes ds.

In subsequent phases, they can receive notifications that some of those neighbour nodes have either been validated or not updating the corresponding values of oks and fs. Once all the pending neighbours have either been validated or failed, it will invoke $\mathtt{checkNeighs}(l, oks, fs)$ to check if the regular expression matches taking into account which neighbours conform or don't conform and passing to the state Ok or Failed which is inactive.

Once executed the Pregel algorithm, it is possible that some nodes are in state $\mathsf{Pending}$and don't receive any message, which means that their validation depends on the existence of some arcs

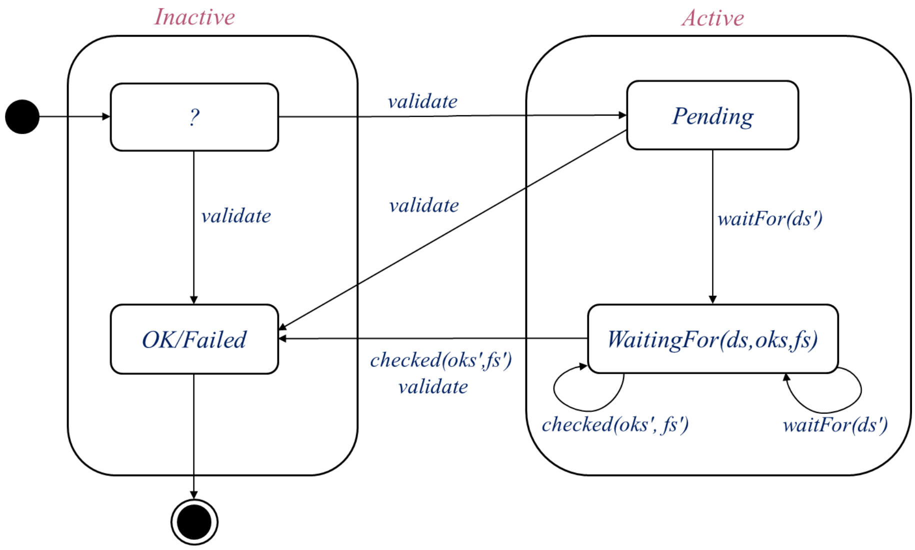

Figure 3.6: State diagram representing the different states in vProg

$$\frac{\langle (s,m_s),p,(o,m_o)\rangle \in \mathcal{G} \qquad m_s(l) = \mathtt{Pending} \qquad tcs(l,\mathcal{S}) = \_ \xrightarrow{p} @l'}{\begin{array}{c}(s,m_s),l \rightsquigarrow \mathtt{WaitFor}((o,p,l')) \\ (o,m_o),l \rightsquigarrow \mathtt{Validate}\end{array}}$$

$$\frac{\langle (s,m_s),p,(o,m_o)\rangle \in \mathcal{G} \qquad m_s(l) = \mathtt{WaitingFor}(ds,oks,fs) \qquad (o,p,l') \in ds \qquad m_o(l') = \mathtt{Ok}}{(s,m_s),l \rightsquigarrow \mathtt{Checked}((o,p,l'),\emptyset)}$$

$$\frac{\langle (s,m_s),p,(o,m_o)\rangle \in \mathcal{G} \qquad m_s(l) = \mathtt{WaitingFor}(ds,oks,fs) \qquad (o,p,l') \in ds \qquad m_o(l') = \mathtt{Failed}}{(s,m_s),l \rightsquigarrow \mathtt{Checked}(\emptyset,(o,p,l'))}$$

Table 3.2: Definition of `sendMsg` for Pregel-based ShEx validation

pointing to some neighbours and they didn't receive messages from those arcs, i.e. there are no arcs in the graph. In that case, a last step in the algorithm checks if those nodes can validate with an empty neighbourhood.

The messages that are sent in the message generation phase are defined using the following presented in table 9.

In order to define `sendMsg`(Triplet):[(Id,Msg)] we will use the notation $(x,m_x),l \rightsquigarrow Msg$ to represent that message Msg is sent to the node x with satsus map m_x for label l. Table 9 represents the rules that declare which messages are sent for each triplet view which is represented as $\langle (s,m_s),p,(o,m_o)\rangle$ where (s,m_s) is the subject, p the predicate and (o,m_o) the object:

Finally `mergeMsg` merges the messages that arrive to the same node and can be defined as:

$$\mathtt{mergeMsg}((n,m),l \rightsquigarrow msg_1,\ (n,m),l \rightsquigarrow msg_2) \quad = \quad (n,m),l \rightsquigarrow msg_1 \oplus msg_2$$

where

$$
\begin{aligned}
\mathsf{Validate} \oplus y &= y \\
\mathsf{Validate} \oplus \mathsf{Checked}(oks, fs) &= \mathsf{Checked}(oks, fs) \\
\mathsf{Validate} \oplus \mathsf{WaitFor}(ds) &= \mathsf{WaitFor}(ds) \\
\mathsf{Checked}(oks, fs) \oplus \mathsf{Validate} &= \mathsf{Checked}(oks, fs) \\
\mathsf{Checked}(oks, fs) \oplus \mathsf{Checked}(oks', fs') &= \mathsf{Checked}(oks \cup oks', fs \cup fs') \\
\mathsf{Checked}(oks, fs) \oplus \mathsf{WaitFor}(ds) &= \mathsf{Checked}(oks \cup ds, fs \cup fs) \\
\mathsf{WaitFor}(ds) \oplus \mathsf{Validate} &= \mathsf{WaitFor}(ds) \\
\mathsf{WaitFor}(ds) \oplus \mathsf{Checked}(oks, fs) &= \mathsf{Checked}(oks \cup ds, fs) \\
\mathsf{WaitFor}(ds) \oplus \mathsf{WaitFor}(ds') &= \mathsf{WaitFor}(ds \cup ds')
\end{aligned}
$$

The algorithm presented in figure 2 required as parameters a function `checkLocal`: $(\mathcal{L}, \mathcal{V}) \to \mathsf{Ok} | \mathsf{Failed} | \mathsf{Pending}(\texttt{Set}[\mathcal{L}])$ that returns Ok if it is possible to check that the node conforms to a shape label locally, Failed if it is possible to check that a node doesn't conform to a shape label locally, and $\mathsf{Pending}(ls)$ if the conformance of a node depends on the arcs in ls.

ar la definición
kLocalOpen
Local para

Figure 3 presents a possible implementation of `checkLocal` for WShEx.

Algorithm 3: Definition of `checkLocal` for a WShEx schema $\langle \mathcal{L}, \delta \rangle$ and wikibase graph $\mathcal{G} = \langle \rho \rangle$

def $\texttt{checkLocal}(l, (n, m)) = \texttt{checkLocal}(\delta(l), (n, m))$
def $\textit{checkLocal}(se, (n, m))$=**match** *se*
  case se_1 `AND` $se_2 \Rightarrow \texttt{checkLocal}(se_1, (n, m)) \wedge \texttt{checkLocal}(se_2, (n, m))$
  case se_1 `OR` $se_2 \Rightarrow \texttt{checkLocal}(se_1, (n, m)) \vee \texttt{checkLocal}(se_2, (n, m))$
  case `NOT` $se \Rightarrow \neg\texttt{checkLocal}(se, (n, m))$
  case $@l \Rightarrow \texttt{checkLocal}(\delta(l), (n, m))$
  case `CLOSED`? { te } $\Rightarrow$ **let**
    $(oks, fs) = \texttt{checkLocalOpen}(te, neighs(m, n)))$
    s_2 = **if** ***CLOSED*** **then**
      $fs = \emptyset$
    else
    in todo...
def $\textit{checkLocalOpen}(te, (n, m))$ = **match** te
  case $te_1; te_2 \Rightarrow \texttt{checkLocalOpen}(te_1, (n, m)) \wedge \texttt{checkLocalOpen}(te_2, (n, m))$
  case $te_1 \mid te_2 \Rightarrow \texttt{checkLocalOpen}(te_1, (n, m)) \vee \texttt{checkLocalOpen}(te_2, (n, m))$
  case $\_ \xrightarrow{p} @l\{min, max\} \Rightarrow \mathsf{Pending} l$ **case** $\_ \xrightarrow{p} cond\{min, max\} \Rightarrow cond(m)$

`checkNeighs`: $(\mathcal{L}, \texttt{Bag}[(\mathcal{E}, \mathcal{L})], \texttt{Set}[(\mathcal{E}, \mathcal{L})]) \to \mathsf{Ok}|\mathsf{Failed}$ can be implemented as in figure **??**.

Algorithm 4: Definition of `checkNeighs` for a WShEx schema $\langle \mathcal{L}, \delta \rangle$ and wikibase graph $\mathcal{G} = \langle \mathcal{Q}, \mathcal{P}, \mathcal{D}, \rho \rangle$

def $\texttt{checkNeighs}(l, bag, fs) = \texttt{checkNeighs}(\delta(l), bag, fs)$
def $\textit{checkNeighs}(se, bag, fs)$=**match** *se*
  case se_1 `AND` $se_2 \Rightarrow \texttt{checkNeighs}(se_1, bag, fs) \wedge \texttt{checkNeighs}(se_2, bag, fs)$
  case se_1 `OR` $se_2 \Rightarrow \texttt{checkNeighs}(se_1, bag, fs) \vee \texttt{checkNeighs}(se_2, bag, fs)$
  case `NOT` $se \Rightarrow \neg\texttt{checkNeighs}(se, bag, fs)$
  case $@l \Rightarrow \texttt{checkNeighs}(\delta(l), bag, fs)$
  case `CLOSED`? { te } $\Rightarrow$ matchRbe(bag,rbe(te))

Finally, the parameter `tripleConstraints`: $\mathcal{L} \to \texttt{Set}[(\mathcal{E}, \mathcal{L})]|$ is defined in figure **??**

3.6.7 Representing Wikidata in Spark GraphX

Spark GraphX supports a kind of property graphs where vertices and edges can have an associated value. The type `Graph[VD,ED]` is used to represent a graph whose vertices are pairs of values of type `(VertexId,VD)` where `VertexId=Long` represents a unique vertex identifier and `VD` represents the value associated with the vertex.

When representing Wikidata graphs in Spark GraphX it is necessary to take into account which entities will be represented as vertices. We opted to include as nodes in the graph only those nodes that can be subjects of statements, i.e. wikidata entities (items and properties), leaving out primitive values like literals, dates, etc. We separated the statements associated with an entity in two kind of statements: local statements, which are embedded inside the value of a node and whose values can be accessed without traversing the graph, and entity statements, whose values are other entities which have their corresponding vertex in the graph.

In this way, the WShEx also needs to distinguish triple expressions between local ones and entity ones.

3.7 Use cases and Applications

In this section we describe some use cases and applications were generating wikidata subsets is necessary.

3.7.1 Scholia

Scholia is a project that is based on visualizing scientific information from Wikidata [49][30]. It offers visualizations about researcher's contributions which are currently generated from direct SPARQL queries to the Wikidata Query Service.

Given the size of Wikidata, visualizations that need to are obtained from large number of results can not be shown for several reasons:

- The queries require a lot of intermediate results which can not be manipulated
- The complexity of the queries require more time than the time slot provided by the wikidata query service.
- ...

As an example, the aspect that visualizes authors by country only works for small countries.

talk a bit more about this so we can later include experiment results...

3.7.2 GeneWiki project

The GeneWiki project has been used as an running example with domain-specific data to generate wikidata subsets in [35].

Talk more about this...compounds example?

3.8 Results

3.9 Related work

Knowledge graphs

An introduction to Knowledge graphs is provided by [27], which cites other books [18, 30, 52] and surveys like [2, 20, 53, 68, 69, 70]. In this paper, we follow the graph models provided in that survey: directed labeled graphs and property graphs and add a new one: wikibase graphs.

[30] `https://scholia.toolforge.org/`

KGRAM (Knowledge Graph Abstract Machine) [11] defines a framework for querying knowledge graphs in different models.

Knowledge graph descriptions

Since the appearance of ShEx in 2014, there has been a lot of interest about RDF validation and description. In 2017, the data shapes working group proposed SHACL (Shapes Constraint Language) as a W3C recommendation [32]. Although SHACL can be used to describe RDF, its main purpose is to validate and check constraints about RDF data makes it less usable to describe RDF subsets.

this para-

ShEx was adopted by Wikidata in 2019 to define entity schemas [65].

We could talk more about: - Constraints and validation - Schemas and ontologies - SHACL - Rules and inference

MARS (Multi-Attributed Relational Structures) [47] has been proposed as a logical framework capable to reason about Wikibase data models.

this...citing
atel Schneider
l on MARS

SQID [46] added an inference layer combined with visualization on top of Wikidata.

[42] proposes the use of an extension of MARS to define constraints on wikidata.

Big data processing and graphs

nt Distributed
s (RDDs) [71]
([22]
[38] -
ab [37] - Power-
21]

A functional definition of Spark using Haskell has been proposed in [10]. It would be interesting to use that functional specification to prove the correctness of the algorithm proposed in this paper.

Knowledge graphs subsets

In the case of RDF, RDFSlice [43, 44] was proposed as a system that generated RDF data fragments from large endpoints like DBpedia. It defines a subset of SPARQL called sliceSPARQL. A new version, called Torpedo, was proposed in [45].

this paragraph

In the case of Wikidata, WDumper [31] was created as a tool that generates filtered wikidata from dumps. It takes two inputs: a JSON compressed dump and a JSON configuration file that describes the different filters, and generates as output an RDF compressed dump. The tool can be run as a web service locally and is also deployed at `https://tools.wmflabs.org/wdumps`. It web service allows the user to introduce the different filters filling a form which generates a dump generation request that is added to a queue. It is possible to see the list of previously requested dumps. Once the dump has been generated, it can be uploaded to Zenodo. WDumper is divided in two main modules, the backend, which has been implemented in Java using the Wikidata Toolkit library [32] and the frontend that has been implemented in Typescript. The use of WDumper to generate Wikidata subsets is described in [28] where 4 subsets are created about the topics: politicians, military politicians, UK universities and GeneWiki data. That paper also presents several use case scenarios and discusses some strengths and weaknesses. In this paper, we present a formal definition of WDumper in the context of other subset generation approaches like the ShEx-based ones.

proach is com-
RDF graph ex-
on with physi-
mentation of
iples which of-
ood balance
n relational stor-
dels like Virtu-
l graph-based
models like g-
31]

The Python library WikidataSets [7] generates Wikidata subsets from specific topics. In the paper the authors generated subsets for the following topics: humans, countries, companies, animal species and films. The tool obtains items following the instances of a topic or subclasses of the topic.

about RDF
artitioning [66]

KGTK (Knowledge graph toolkit) [29] presents a tool that works with knowledge graphs by defining a common format called KGTK format based on hypergraphs. It is possible to import

Iypergraph

[31] `https://github.com/bennofs/wdumper`

[32] `https://github.com/Wikidata/Wikidata-Toolkit`

and export data from different formats like Wikidata or ConceptNet, do several operations over those graphs like: validation, cleaning, graph manipulation (sort, column removal, edge filtering) and graph merging (join, cat) operations. The tool also supports graph querying and analytics operations. Given that KGTK can take as input Wikidata dumps and generate Wikidata dumps as output, it is possible to use KGTK to generate subsets of Wikidata. Further work needs to be done about comparing the performance and usability of KGTK against the approach presented in this paper.

3.10 Conclusions

Some ideas:
Should we talk about future work?
- ShEx expressivity
- Discriminators
- Restricts ?? The other day, Andra asked an interesting question...if we can extend a shape adding more properties...could we do the reverse operation, modify a shape removing some properties? We may add a paragraph about this to a discussion section, or talk about it as future work...

3.11 Acknowledgments

We want to thank Eric Prud'hommeaux, Andra Waagmeester, Tom Baker, Kat Thornton and Iovka Boneva as well as all the members of the Shape Expressions Community Group. The Wikidata and Wikibase community provided a great environment and software whose potential we are still discovering, special thanks to Lydia Pintscher, Adam Shorland, Daniel Mietchen, Egon Willighagen, etc. Some parts of this research took inspiration from several events like 2019 and 2020 biohackathons and SWAT4HCLS. Special thanks to Dan Brickley for helping with the coordination and insights on the Wikidata subsetting project. Markus Krötzsch's lectures and papers have also been a source of inspiration for the Wikibase formal model and Tim Berners-lee's running example. Guillermo Facundo Colunga, Pablo Menéndez Suárez, Jorge Álvarez Fidalgo and Daniel Fernández Álvarez from the WESO research group have also helped with the experiments on Wikidata Subsetting and Scholia.

Universidad de Oviedo

Bibliography

[1] Bilal Abu-Salih. "Domain-specific knowledge graphs: A survey". In: *Journal of Network and Computer Applications* 185 (July 2021), page 103076. DOI: `10.1016/j.jnca.2021.103076` (cited on page 12).

[2] Tareq Al-Moslmi et al. "Named Entity Extraction for Knowledge Graphs: A Literature Overview". In: *IEEE Access* 8 (2020), pages 32862–32881. DOI: `10.1109/ACCESS.2020.2973928`. URL: `https://doi.org/10.1109/ACCESS.2020.2973928` (cited on page 39).

[3] Renzo Angles et al. "Foundations of Modern Query Languages for Graph Databases". In: *ACM Computing Surveys* 50.5 (2017), 68:1–68:40. DOI: `10.1145/3104031` (cited on page 12).

[4] Adrian Bielefeldt, Julius Gonsior, and Markus Krötzsch. "Practical Linked Data Access via SPARQL: The Case of Wikidata". In: *Proceedings of the WWW2018 Workshop on Linked Data on the Web (LDOW-18)*. Edited by Tim Berners-Lee et al. Volume 2073. CEUR Workshop Proceedings. CEUR-WS.org, 2018 (cited on page 20).

[5] Iovka Boneva, Jose Emilio Labra Gayo, and Eric Prud'hommeaux. "Semantics and Validation of Shapes Schemas for RDF". In: *International Semantic Web Conference*. 2017 (cited on pages 11, 22).

[6] Iovka Boneva, Jose Emilio Labra Gayo, and Eric G. Prud'hommeaux. "Semantics and Validation of Shapes Schemas for RDF". In: *The Semantic Web - ISWC 2017 - 16th International Semantic Web Conference, Vienna, Austria, October 21-25, 2017, Proceedings, Part I*. Edited by Claudia d'Amato et al. Volume 10587. Lecture Notes in Computer Science. Springer, Oct. 2017, pages 104–120. ISBN: 978-3-319-68287-7 (cited on pages 21, 24).

[7] Armand Boschin and Thomas Bonald. "WikiDataSets: Standardized sub-graphs from Wikidata". In: *arXiv:1906.04536 [cs, stat]* (Oct. 2019). arXiv: 1906.04536. URL: `http://arxiv.org/abs/1906.04536` (cited on page 40).

[8] Sebastian Burgstaller-Muehlbacher et al. "Wikidata as a semantic framework for the Gene Wiki initiative". In: *Database* 2016 (2016), baw015. DOI: `10.1093/database/baw015` (cited on page 17).

[9] Spencer Chang. *Scaling Knowledge Access and Retrieval at Airbnb*. AirBnB Medium Blog. `https://medium.com/airbnb-engineering/scaling-knowledge-access-and-retrieval-at-airbnb-665b6ba21e95`. Sept. 2018 (cited on page 10).

[10] Yu-Fang Chen et al. "An Executable Sequential Specification for Spark Aggregation". In: *Networked Systems*. Springer International Publishing, 2017, pages 421–438. DOI: `10.1007/978-3-319-59647-1\_31` (cited on page 40).

[11] Olivier Corby and Catherine Faron Zucker. "The KGRAM Abstract Machine for Knowledge Graph Querying". In: *2010 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology*. IEEE, Aug. 2010. DOI: `10.1109/wi-iat.2010.144` (cited on page 40).

[12] Constance Crompton et al. "Familiar Wikidata: The Case for Building a Data Source We Can Trust". en. In: (2020). DOI: `10.48404/POP.2020.02` (cited on page 17).

[13] Richard Cyganiak, David Wood, and Markus Lanthaler. *RDF 1.1 Concepts and Abstract Syntax, W3C Recommendation 25 February 2014*. W3C Recommendation. World Wide Web Consortium, Feb. 2014. URL: `https://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/` (cited on pages 12, 14).

[14] Richard Cyganiak, David Wood, and Markus Lanthaler. *RDF-star and SPARQL-star*. Draft Community Group Report. World Wide Web Consortium, July 2021. URL: `https://w3c.github.io/rdf-star/cg-spec` (cited on page 14).

[15] Deepika Devarajan. *Happy Birthday Watson Discovery*. IBM Cloud Blog. `https://www.ibm.com/blogs/bluemix/2017/12/happy-birthday-watson-discovery/`. Dec. 2017 (cited on page 10).

[16] Martin Dürst and Michel Suignard. *Internationalized Resource Identifiers (IRIs)*. RFC 3987. Internet Engineering Task Force, Jan. 2005. URL: `http://www.ietf.org/rfc/rfc3987.txt` (cited on page 12).

[17] Fredo Erxleben et al. "Introducing Wikidata to the Linked Data Web". In: *Proceedings of the 13th International Semantic Web Conference (ISWC 2014)*. Edited by Peter Mika et al. Volume 8796. LNCS. Springer, Oct. 2014, pages 50–65. DOI: `10.1007/978-3-319-11964-9\_4` (cited on pages 14, 20).

[18] Dieter Fensel et al. *Knowledge Graphs - Methodology, Tools and Selected Use Cases*. Springer, 2020. ISBN: 978-3-030-37438-9. DOI: `10.1007/978-3-030-37439-6` (cited on page 39).

[19] Nadime Francis et al. "Cypher". In: *Proceedings of the 2018 International Conference on Management of Data*. ACM, May 2018. DOI: `10.1145/3183713.3190657` (cited on page 16).

[20] Genet Asefa Gesese, Russa Biswas, and Harald Sack. "A Comprehensive Survey of Knowledge Graph Embeddings with Literals: Techniques and Applications". In: *Proceedings of the Workshop on Deep Learning for Knowledge Graphs (DL4KG2019) Co-located with the 16th Extended Semantic Web Conference 2019 (ESWC 2019), Portoroz, Slovenia, June 2, 2019*. Edited by Mehwish Alam et al. Volume 2377. CEUR Workshop Proceedings. Portoroz, Slovenia: Sun SITE Central Europe (CEUR), 2019, pages 31–40. URL: `http://ceur-ws.org/Vol-2377` (cited on page 39).

[21] Joseph E. Gonzalez et al. “PowerGraph: Distributed Graph-Parallel Computation on Natural Graphs”. In: *10th USENIX Symposium on Operating Systems Design and Implementation (OSDI 12)*. Hollywood, CA: USENIX Association, Oct. 2012, pages 17–30. ISBN: 978-1-931971-96-6. URL: https://www.usenix.org/conference/osdi12/technical-sessions/presentation/gonzalez (cited on page 40).

[22] Joseph E. Gonzalez et al. “GraphX: Graph Processing in a Distributed Dataflow Framework”. In: *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*. Broomfield, CO: USENIX Association, Oct. 2014, pages 599–613. ISBN: 978-1-931971-16-4. URL: https://www.usenix.org/conference/osdi14/technical-sessions/presentation/gonzalez (cited on page 40).

[23] Ferras Hamad, Isaac Liu, and Xian Xing Zhang. *Food Discovery with Uber Eats: Building a Query Understanding Engine*. Uber Engineering Blog. https://eng.uber.com/uber-eats-query-understanding/. June 2018 (cited on page 10).

[24] Qi He, Bee-Chung Chen, and Deepak Agarwal. *Building The LinkedIn Knowledge Graph*. LinkedIn Blog. https://engineering.linkedin.com/blog/2016/10/building-the-linkedin-knowledge-graph. Oct. 2016 (cited on page 10).

[25] Daniel Hernández, Aidan Hogan, and Markus Krötzsch. “Reifying RDF: What Works Well With Wikidata?” In: *Proceedings of the 11th International Workshop on Scalable Semantic Web Knowledge Base Systems*. Edited by Thorsten Liebig and Achille Fokoue. Volume 1457. CEUR Workshop Proceedings. CEUR-WS.org, 2015, pages 32–47 (cited on page 13).

[26] Aidan Hogan et al. “Everything you always wanted to know about blank nodes”. In: *Journal of Web Semantics* 27–28 (2014), pages 42–69. DOI: 10.1016/j.websem.2014.06.004 (cited on page 12).

[27] Aidan Hogan et al. “Knowledge Graphs”. In: *ACM Computing Surveys* 54.4 (July 2021), pages 1–37. DOI: 10.1145/3447772 (cited on pages 11, 39).

[28] Seyed Amir Hosseini Beghaeiraveri, Alasdair J. G. Gray, and Fiona J. McNeill. “Experiences of Using WDumper to Create Topical Subsets from Wikidata”. English. In: *CEUR Workshop Proceedings* 2873 (June 2021). Publisher Copyright: Copyright © 2021 for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).; 2nd International Workshop On Knowledge Graph Construction : Co-located with the ESWC 2021 ; Conference date: 06-06-2021 Through 06-06-2021. ISSN: 1613-0073 (cited on page 40).

[29] Filip Ilievski et al. “KGTK: A Toolkit for Large Knowledge Graph Manipulation and Analysis”. In: *Lecture Notes in Computer Science*. Springer International Publishing, 2020, pages 278–293. DOI: 10.1007/978-3-030-62466-8_18 (cited on page 40).

[30] Mayank Kejriwal. *Domain-Specific Knowledge Graph Construction*. Springer Briefs in Computer Science. Springer, 2019. ISBN: 978-3-030-12374-1. DOI: 10.1007/978-3-030-12375-8 (cited on page 39).

[31] Abdallah Khelil et al. “Combining Graph Exploration and Fragmentation for Scalable RDF Query Processing”. In: *Information Systems Frontiers* 23.1 (Mar. 2020), pages 165–183. DOI: 10.1007/s10796-020-09998-z (cited on page 40).

[32] Holger Knublauch and Dimitris Kontokostas. *Shapes Constraint Language (SHACL), W3C Recommendation 20 July 2017*. W3C Recommendation. World Wide Web Consortium, June 2017. URL: https://www.w3.org/TR/2017/REC-shacl-20170720/ (cited on page 40).

[33] Arun Krishnan. *Making search easier: How Amazon's Product Graph is helping customers find products more easily*. Amazon Blog. `https://blog.aboutamazon.com/innovation/making-search-easier`. Aug. 2018 (cited on page 10).

[34] Jose Emilio Labra Gayo, Daniel Fernández Álvarez, and Herminio García-González. "RDFShape: An RDF Playground Based on Shapes". In: *Proceedings of the ISWC 2018 Posters and Demonstrations, Industry and Blue Sky Ideas Tracksco-located with 17th International Semantic Web Conference*. Volume 2180. CEUR Workshop Proceedings. 2018 (cited on page 13).

[35] Jose Emilio Labra Gayo et al. "Knowledge graphs and wikidata subsetting". In: *BioHackrXiv Preprints* (2021). DOI: `https://doi.org/10.37044/osf.io/wu9et` (cited on page 39).

[36] Jens Lehmann et al. "DBpedia - A large-scale, multilingual knowledge base extracted from Wikipedia". In: *Semantic Web Journal* 6.2 (2015), pages 167–195 (cited on page 12).

[37] Yucheng Low et al. "Distributed GraphLab". In: *Proceedings of the VLDB Endowment* 5.8 (Apr. 2012), pages 716–727. DOI: `10.14778/2212351.2212354` (cited on page 40).

[38] Grzegorz Malewicz et al. "Pregel: a system for large-scale graph processing". In: *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*. ACM, June 2010. DOI: `10.1145/1807167.1807184` (cited on pages 33, 40).

[39] Stanislav Malyshev et al. "Getting the Most out of Wikidata: Semantic Technology Usage in Wikipedia's Knowledge Graph". In: *Proceedings of the 17th International Semantic Web Conference (ISWC'18)*. Edited by Denny Vrandečić et al. Volume 11137. LNCS. Springer, 2018, pages 376–394 (cited on page 20).

[40] Stanislav Malyshev et al. "Getting the most out of Wikidata: Semantic technology usage in Wikipedia's knowledge graph". In: *The Semantic Web - ISWC 2018 - 17th International Semantic Web Conference, Monterey, CA, USA, October 8-12, 2018, Proceedings, Part II*. Edited by Denny Vrandecic et al. Volume 11137. Lecture Notes in Computer Science. Springer, Oct. 2018, pages 376–394. ISBN: 978-3-030-00667-9 (cited on page 17).

[41] Frank Manola and Eric Miller. *Resource Description Framework: RDF Primer*. W3C Recommendation. World Wide Web Consortium, Feb. 2004. URL: `https://www.w3.org/TR/2004/REC-rdf-primer-20040210` (cited on page 13).

[42] David Martin and Peter P. Schneider. "Wikidata Constraints on MARS". In: *Proceedings of the 1st Wikidata Workshop (Wikidata 2020)*. Volume 2773. 2020. URL: `http://ceur-ws.org/Vol-2773/paper-12.pdf` (cited on page 40).

[43] Edgard Marx et al. "Large-scale RDF Dataset Slicing". In: *7th IEEE International Conference on Semantic Computing, September 16-18, 2013, Irvine, California, USA*. 2013. URL: `http://svn.aksw.org/papers/2013/ICSC_SLICE/public.pdf` (cited on page 40).

[44] Edgard Marx et al. "Towards an Efficient RDF Dataset Slicing". In: *International Journal of Semantic Computing* 07.04 (2013), pages 455–477. DOI: `10.1142/S1793351X13400151`. eprint: `http://www.worldscientific.com/doi/pdf/10.1142/S1793351X13400151`. URL: `http://www.worldscientific.com/doi/abs/10.1142/S1793351X13400151` (cited on page 40).

[45] Edgard Marx et al. "Torpedo: Improving the State-of-the-Art RDF Dataset Slicing". In: *11th IEEE International Conference on Semantic Computing, Jan 30-Feb 1, 2017, San Diego, California, USA*. 2017. URL: `https://svn.aksw.org/papers/2017/Torpedo_ICSC/public.pdf` (cited on page 40).

[46] Maximilian Marx and Markus Krötzsch. “SQID: Towards Ontological Reasoning for Wikidata”. In: *Proceedings of the ISWC 2017 Posters & Demonstrations Track*. Edited by Nadeschda Nikitina and Dezhao Song. CEUR Workshop Proceedings. CEUR-WS.org, Oct. 2017 (cited on page 40).

[47] Maximilian Marx, Markus Krötzsch, and Veronika Thost. “Logic on MARS: Ontologies for generalised property graphs”. In: *Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI’17)*. Edited by Carles Sierra. International Joint Conferences on Artificial Intelligence, Aug. 2017, pages 1188–1194. DOI: `10.24963/ijcai.2017/165` (cited on pages 18, 40).

[48] Vinh Nguyen, Olivier Bodenreider, and Amit Sheth. “Don’t like RDF reification?” In: *Proceedings of the 23rd international conference on World wide web - WWW ’14*. ACM Press, 2014. DOI: `10.1145/2566486.2567973` (cited on page 14).

[49] Finn Årup Nielsen, Daniel Mietchen, and Egon Willighagen. “Scholia, Scientometrics and Wikidata”. In: *Lecture Notes in Computer Science*. Springer International Publishing, 2017, pages 237–259. DOI: `10.1007/978-3-319-70407-4\_36` (cited on page 39).

[50] Natasha F. Noy et al. “Industry-scale Knowledge Graphs: Lessons and Challenges”. In: *ACM Queue* 17.2 (2019), page 20 (cited on pages 10, 12).

[51] Ora Lassila and Ralph R. Swick. *Resource Description Framework (RDF) Model and Syntax Specification*. https://www.w3.org/TR/1999/REC-rdf-syntax-19990222/. 1999. URL: `https://www.w3.org/TR/1999/REC-rdf-syntax-19990222/` (cited on page 10).

[52] Jeff Z. Pan et al., editors. *Exploiting Linked Data and Knowledge Graphs in Large Organisations*. Springer, 2017. ISBN: 978-3-319-45652-2. DOI: `10.1007/978-3-319-45654-6` (cited on page 39).

[53] Heiko Paulheim. “Knowledge graph refinement: A survey of approaches and evaluation methods”. In: *Semantic Web Journal* 8.3 (2017), pages 489–508. DOI: `10.3233/SW-160218` (cited on page 39).

[54] R. J. Pittman et al. *Cracking the Code on Conversational Commerce*. eBay Blog. `https://www.ebayinc.com/stories/news/cracking-the-code-on-conversational-commerce/`. Apr. 2017 (cited on page 10).

[55] Eric Prud’hommeaux, Jose Emilio Labra Gayo, and Harold Solbrig. “Shape Expressions: An RDF Validation and Transformation Language”. In: *Proceedings of the 10th International Conference on Semantic Systems, SEMANTICS 2014, Leipzig, Germany, September 4-5, 2014*. Edited by Harald Sack et al. ACM Press, Sept. 2014, pages 32–40. ISBN: 978-1-4503-2927-9. DOI: `10.1145/2660517.2660523` (cited on page 21).

[56] Eric Prud’hommeaux et al. *Shape Expressions Language 2.0*. https://shexspec.github.io/spec/. Apr. 2017. URL: `https://shexspec.github.io/spec/` (cited on pages 11, 21, 22).

[57] Edward W. Schneider. “Course Modularization Applied: The Interface System and Its Implications For Sequence Control and Data Analysis”. In: *Association for the Development of Instructional Systems (ADIS), Chicago, Illinois, April 1972*. 1973 (cited on page 11).

[58] Dan Scott and Stacy Allison-Cassin. “Wikidata: a platform for your library’s linked open data”. In: *Code4Lib Journal* 40 (May 2018). URL: `https://journal.code4lib.org/articles/13424` (cited on page 17).

[59] Philipp Seifer, Ralf Lämmel, and Steffen Staab. *ProGS: Property Graph Shapes Language (Extended Version)*. 2021. arXiv: `2107.05566 [cs.DB]` (cited on page 15).

[60] Saurabh Shrivastava. *Bring rich knowledge of people, places, things and local businesses to your apps*. Bing Blogs. https://blogs.bing.com/search-quality-insights/2017-07/bring-rich-knowledge-of-people-places-things-and-local-businesses-to-your-apps. July 2017 (cited on page 10).

[61] Amit Singhal. *Introducing the Knowledge Graph: things, not strings*. Google Blog. https://www.blog.google/products/search/introducing-knowledge-graph-things-not/. May 2012 (cited on pages 10, 11).

[62] Slawek Staworko et al. "Complexity and Expressiveness of ShEx for RDF". In: *18th International Conference on Database Theory, ICDT 2015*. Volume 31. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015, pages 195–211 (cited on page 11).

[63] Fabian M. Suchanek, Gjergji Kasneci, and Gerhard Weikum. "YAGO: A core of semantic knowledge unifying WordNet and Wikipedia". In: *Proceedings of the 16th International Conference on World Wide Web, WWW 2007, Banff, Alberta, Canada, May 8-12, 2007*. Edited by Carey L. Williamson et al. ACM Press, May 2007, pages 697–706. ISBN: 978-1-59593-654-7 (cited on page 12).

[64] Thomas Pellissier Tanon et al. "From Freebase to Wikidata". In: *Proceedings of the 25th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, Apr. 2016. DOI: 10.1145/2872427.2874809 (cited on page 16).

[65] Katherine Thornton et al. "Using Shape Expressions (ShEx) to Share RDF Data Models and to Guide Curation with Rigorous Validation". In: *The Semantic Web - 16th International Conference, ESWC 2019, Portorož, Slovenia, June 2-6, 2019, Proceedings*. Edited by Pascal Hitzler et al. Volume 11503. Lecture Notes in Computer Science. Springer, June 2019, pages 606–620. ISBN: 978-3-030-21347-3. DOI: 10.1007/978-3-030-21348-0_39 (cited on page 40).

[66] Dominik Tomaszuk, Łukasz Skonieczny, and David Wood. "RDF Graph Partitions: A Brief Survey". In: *Beyond Databases, Architectures and Structures*. Springer International Publishing, 2015, pages 256–264. DOI: 10.1007/978-3-319-18422-7_23 (cited on page 40).

[67] Denny Vrandečić and Markus Krötzsch. "Wikidata: A Free Collaborative Knowledgebase". In: *Communications of the ACM* 57.10 (2014), pages 78–85 (cited on pages 12, 16).

[68] Xiu-Qing Wang and Shun-Kun Yang. "A Tutorial and Survey on Fault Knowledge Graph". In: *International 2019 Cyberspace Congress, CyberDI and CyberLife, Beijing, China, December 16–18, 2019, Proceedings, Part II*. Edited by Huansheng Ning. Springer, Dec. 2019, pages 256–271 (cited on page 39).

[69] Guohui Xiao et al. "Virtual Knowledge Graphs: An Overview of Systems and Use Cases". In: *Data Intelligence* 1.3 (2019), pages 201–223. DOI: 10.1162/dint_a_00011. URL: https://doi.org/10.1162/dint%5C_a%5C_00011 (cited on page 39).

[70] Jihong Yan et al. "A retrospective of knowledge graphs". In: *Frontiers of Computer Science* 12.1 (2018), pages 55–74. DOI: 10.1007/s11704-016-5228-9. URL: https://doi.org/10.1007/s11704-016-5228-9 (cited on page 39).

[71] Matei Zaharia et al. "Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing". In: *9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*. San Jose, CA: USENIX Association, Apr. 2012, pages 15–28. ISBN: 978-931971-92-8. URL: https://www.usenix.org/conference/nsdi12/technical-sessions/presentation/zaharia (cited on page 40).

.1 GeneWiki Schema

In this annex, we include the GeneWiki ShEx schema that has been used to create the Wikidata subset related with the GeneWiki project.

```
PREFIX wde: <http://www.wikidata.org/entity/>
PREFIX :    <http://example.org/>

start=@:disease

:active_site EXTRA wde:P31 {
  wde:P31   [ wd:Q423026 ]      ;
  wde:P361 @:protein_family * ;
}

:anatomical_structure EXTRA wde:P31 {
  wde:P31   [ wd:Q4936952 ] ;
  wde:P361 @:anatomical_structure * ;
  wde:P527 @:anatomical_structure *
}

:binding_site EXTRA wde:P31 {
  wde:P31   [ wd:Q616005 ] ;
  wde:P361 @:protein_family *
}

:biological_pathway EXTRA wde:P31 {
  wde:P31 [ wd:Q4915012 ] ;
  wde:P361 @:biological_pathway * ;
  wde:P361 @:gene * ;
  wde:P527 @:biological_pathway * ;
  wde:P527 @:gene * ;
}


:biological_process {
}

:chemical_compound EXTRA wde:P31 {
  wde:P31   [ wd:Q11173 ] ;
  wde:P2868 @:therapeutic_use * ;
  wde:P2868 @:pharmacologic_action * ;
  wde:P769  @:therapeutic_use * ;
  wde:P769  @:pharmacologic_action * ;
  wde:P279  @:pharmacologic_action * ;
  wde:P3780 @:pharmaceutical_product * ;
  wde:P2175 @:disease * ;
  wde:P361  @:biological_pathway * ;
  wde:P361  @:medication * ;
  wde:P703  @:taxon * ;
  wde:P3364 @:chemical_compound *
}

:chromosome EXTRA wde:P31 {
  wde:P31 [ wd:Q37748 ]
}
```

```
:disease EXTRA wde:P31 {
  wde:P31    [ wd:Q12136 ] ;
  wde:P780  @:disease * ;
  wde:P780  @:symptom * ;
  wde:P828  @:taxon * ;
  wde:P2293 @:gene * ;
  wde:P927  @:anatomical_structure * ;
  wde:P2176 @:medication * ;
  wde:P2176 @:chemical_compound * ;
}

:gene EXTRA wde:P31 {
  wde:P703  @:taxon * ;
  wde:P684  @:gene * ;
  wde:P682  @:biological_process ;
  wde:P688  @:protein * ;
  wde:P527  @:biological_pathway *;
  wde:P1057 @:chromosome ;
}

:mechanism_of_action EXTRA wde:P31 {
  wde:P31 [ wd:Q3271540 ]
}

:medication EXTRA wde:P31 {
  wde:P31 [ wd:Q12140 ] ;
  wde:P2175 @:disease * ;
  wde:P3780 @:pharmaceutical_product * ;
  wde:P527  @:medication * ;
  wde:P361  @:biological_pathway * ;
  wde:P769  @:pharmacologic_action * ;
  wde:P769  @:chemical_compound * ;
  wde:P769  @:therapeutic_use * ;
  wde:P2868 @:pharmacologic_action * ;
  wde:P2868 @:therapeutic_use * ;
  wde:P279  @:pharmacologic_action * ;
  wde:P279  @:therapeutic_use * ;
}

:molecular_function {
}

:pharmaceutical_product EXTRA wde:P31 {
  wde:P31   [ wd:Q28885102 ] ;
  wde:P3781 @:therapeutic_use * ;
  wde:P3781 @:pharmacologic_action * ;
  wde:P3781 @:chemical_compound * ;
  wde:P4044 @:disease *
}

:pharmacologic_action EXTRA wde:P31 {
  wde:P31 [wd:Q50377224 ] ;
  wde:P3780 @:pharmaceutical_product * ;
  wde:P2175 @:disease *
}
```

```
:protein_domain EXTRA wde:P31 {
  wde:P31   [ wd:Q898273 ] ;
  wde:P527 @:protein_domain * ;
  wde:P361 @:protein_domain * ;
}

:protein_family EXTRA wde:P31 {
  wde:P31   [ wd:Q417841 ] ;
  wde:P527 @:protein * ;
}

:protein EXTRA wde:P31 {
  wde:P31   [ wd:Q8054 ] ;
  wde:P129 @:medication * ;
  wde:P129 @:protein  * ;
  wde:P129 @:chemical_compound  * ;
  wde:P702 @:gene * ;
  wde:P361 @:protein_family * ;
  wde:P527 @:active_site * ;
  wde:P527 @:binding_site * ;
  wde:P680 @:molecular_function * ;
  wde:P682 @:biological_process * ;
  wde:P703 @:taxon * ;
  wde:P681 @:anatomical_structure * ;
  wde:P681 @:protein * ;
}

:ribosomal_RNA EXTRA wde:P31 {
  wde:P31   [ wd:Q28885102 ] ;
  wde:P703 @:taxon *
}

:sequence_variant EXTRA wde:P31 {
  wde:P31    [ wd:Q15304597 ] ;
  wde:P3355  @:chemical_compound * ;
  wde:P3354 @:chemical_compound * ;
  wde:P3354 @:medication * ;
  wde:P3355 @:chemical_compound * ;
  wde:P3355 @:medication * ;
  wde:P3433 @:gene * ;
  wde:P1057 @:chromosome * ;
}


:supersecondary_structure EXTRA wde:P31 {
  wde:P31 [ wd:Q7644128 ] ;
  wde:P361 . *
}

:symptom EXTRA wde:P31 {
  wde:P31 [ wd:Q169872 ]
}

:taxon EXTRA wde:P31 {
  wde:P31 [ wd:Q16521 ] ;
}
```

```
:therapeutic_use EXTRA wde:P31 {
  wde:P31    [ wd:Q50379781 ] ;
  wde:P3781 @:pharmaceutical_product * ;
  wde:P2175 @:disease *
}
```

Universidad de Oviedo

Índice alfabético