Schema Inference on Wikidata

Master's Thesis of

Lucas Werkmeister

at the Department of Informatics
Institute for Program Structures and Data Organization (IPD)

Reviewer:	Prof. Reussner
Second reviewer:	Prof. Sack
Advisor:	Dr. Koutraki

16. April 2018 – 15. October 2018

Karlsruher Institut für Technologie
Fakultät für Informatik
Postfach 6980
76128 Karlsruhe

I declare that I have developed and written the enclosed thesis completely by myself, and have not used sources or means without declaration in the text.

Karlsruhe, 2018-10-15

. . . . . .

(Lucas Werkmeister)

Abstract

Wikidata, the free knowledge base in the Wikimedia movement, is used by various Wikimedia projects and third parties to provide machine-readable information and data. Its data quality is managed and monitored by its community using several quality control mechanisms, recently including formal schemas in the Shape Expressions language. However, larger schemas can be tedious to write, making automatic inference of schemas from a set of exemplary Items an attractive prospect.

This thesis investigates this option by updating and adapting the RDF2Graph program to infer schemas from a set of Wikidata Items, and providing a web-based tool which makes this process available to the Wikidata community. Though the resulting schemas are usually not fit for direct validation, they can still be useful as a form of describing the layout of an area of Wikidata's data model, a way to notice potential issues in the source data, or a basis for a manually curated schema.

Zusammenfassung

Wikidata, die freie Wissensdatenbank in der Wikimedia-Bewegung, wird von verschiedenen Wikimedia- und anderen Projekten als Quelle für maschinenlesbare Informationen und Daten verwendet. Die Datenqualität wird durch die Wikidata-Community verwaltet und überwacht, wobei verschiedene Mechanismen zur Qualitätskontrolle zum Einsatz kommen, in letzter Zeit auch formale Schemata in der Shape Expressions-Sprache. Allerdings ist es langwierig, größere Schemata zu schreiben, was automatischen Rückschluss solcher Schemata aus einem Satz beispielhafter Datenobjekte attraktiv macht.

Diese Arbeit untersucht diese Option, indem das RDF2Graph-Programm aktualisiert und angepasst wird, um Schemata aus einem Satz von Wikidata-Datenobjekten rückzuschließen, und durch das Angebot eines webbasierten Werkzeugs, welches diesen Vorgang der Wikidata-Community zugänglich macht. Obwohl die resultierenden Schemata meist nicht für direkte Validierung geeignet sind, können sie immer noch als Beschreibung eines Bereichs des Wikidata-Datenmodells, als Mittel, mögliche Probleme in den Eingabedaten zu bemerken, oder als Basis für manuell betreute Schemata nützlich sein.

Contents

List of Figures

List of Listings

List of Tables

1. Introduction

As Wikidata, the free knowledge base in the Wikimedia movement, continues to grow in volume and scope [7] and is used by more and more Wikimedia projects and third parties, its data quality has been identified as one of the most important areas of development in the future [10]: in order for Wikidata to be useful, its data must be trustworthy and available in a consistent format. Unchecked vandalism discourages data reuse, while inconsistent data models make it significantly more difficult or even impossible.

To combat these problems, several quality control mechanisms are used on Wikidata. Recently, editors have begun exploring the use of Shape Expressions as another quality control mechanism to use, forming the WikiProject ShEx. Compared to the more established, Wikidata-specific quality constraints system, Shape Expressions are more powerful and expressive, and are also not specific to Wikidata alone. However, schemas for Shape Expressions are tedious to write by hand.

Automatically inferring schemas from Wikidata Items promises to simplify the schema authoring process: instead of manually putting together the schema, describing shapes for different classes of Items, one simply selects a set of Items, and a schema is automatically generated based on the data about these Items. If the selected Items have been carefully edited to conform to a pre-existing schema, perhaps described informally or only present in the minds of the editors, then the result may be a formalization of that schema; alternatively, applying the same process to a less curated set of input Items may result in a coherent summary of the current schema of those Items and possibly even demonstrate problems in the input data.

This thesis investigates the usefulness and applicability of automatically inferring schemas for Wikidata from sets of exemplary Items. It builds on the existing RDF2Graph [3] program, updating and adapting it to support Wikidata and automating the whole inference process. This is then made available to the whole Wikidata community by incorporating it into a web-based tool.

The remainder of this thesis is organized as follows. Chapter 2 explains concepts that are required to understand the thesis. Chapter 3 describes general updates for RDF2Graph as well as changes that were made to add Wikidata support to it. Chapter 4 introduces the Wikidata Shape Expressions Inference tool and describes its design and implementation. Chapter 5 then evaluates the usefulness of the tool and the resulting schemas. Finally, chapter 6 summarizes the results and concludes the main text of the thesis. These are followed by a Bibliography listing sources, a Glossary with short definitions of most acronyms and terms used in this thesis, and an Appendix with some ancillary content that does not belong in the main text.

Its source code is available in the repository at `https://github.com/lucaswerkmeister/master-thesis`.

2. Background

This chapter provides background information on several concepts that are important to the rest of this thesis. The sections below are not intended to be comprehensive introductions to the respective topics, but focus on the aspects that are necessary to understand the thesis, omitting parts that are unnecessary or distracting in this context. Further information, including full introductions to most topics covered here, can be found in the works listed in the bibliography.

2.1. RDF

Resource Description Framework (RDF) [6] is a framework for describing and working with Linked Data, developed by the RDF Working Group under the umbrella of the World Wide Web Consortium (W3C). In RDF, information is arranged in subject-predicate-object triples, such as "<Alan Turing> <is a> <human>" or "<Lima> <was founded in> <18 January 1535>". All three elements of a triple are typically resources, identified by an Internationalized Resource Identifier (IRI) like `http://example.com/Alan_Turing` or `http://www.wikidata.org/entity/Q5`, but the object of a triple can also be some other kind of value, such as a textual, numerical or other literal (e. g. the date literal "18 January 1535" above). A collection of such triples forms a directed, labeled graph, where the triples describe individual edges and the nodes are the subjects and objects of the triples: triples with the same subject constitute different outgoing arcs from the same node. This graph can then be queried using the SPARQL Protocol and RDF Query Language (SPARQL) [15]. For a more detailed introduction to RDF and related technologies, see the RDF 1.1 Primer [13].

To improve readability, the IRIs identifying a resource are usually abbreviated using prefixes. For example, once the prefix `ex:` has been defined to mean `http://shex.example/`, the IRIs `http://shex.example/Person` and `http://shex.example/dateOfBirth` can be abbreviated as `ex:Person` and `ex:dateOfBirth`. (This "example" prefix, also used below for other examples, follows the naming conventions of schema.org [12], in that types (`ex:Person`) and entities (`ex:JohnDoe`) are in upper camel case while predicates (`ex:dateOfBirth`) are in lower camel case.)

While RDF can be used with any resource IRIs, one of its strengths is the ability to reuse the same vocabularies (effectively, sets of resources) in a variety of different graphs. For example, many graphs use the same predicate, `rdfs:label`, to link a resource to a human-readable version of its name (its label); a tool based on RDF can thus offer human-readable names for resources from all these graphs without requiring any specific knowledge about them, and the graphs are more useful when used in combination. One common vocabulary is RDF Schema (RDFS) [4], which among others provides two important

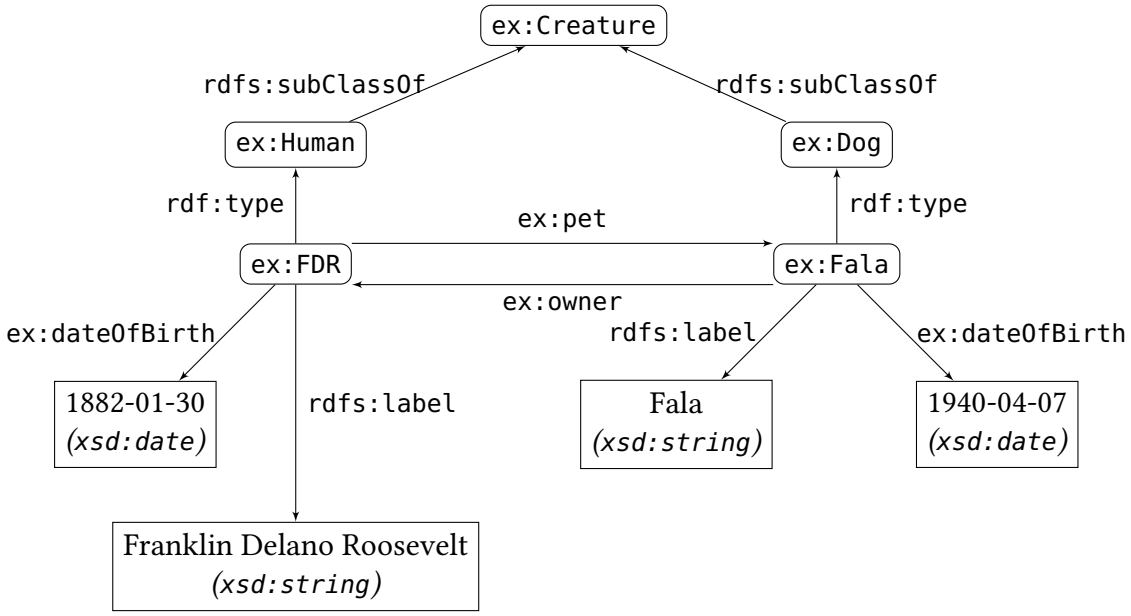


Figure 2.1.: RDF graph for President Franklin D. Roosevelt and his dog

predicates: `rdf:type` and `rdfs:subClassOf`. `rdf:type` connects a resource to its class, and `rdfs:subClassOf` connects a class to its parent class. (The `rdf:` and `rdfs:` prefixes are both part of RDF Schema; the distinction between them is "a somewhat annoying historical artifact" [13] with no real significance today.) Another commonly used vocabulary is XML Schema Definition (XSD) [8], which provides several basic datatypes: for example, `xsd:string` is the datatype for a simple text string (language-agnostic), and `xsd:date` and `xsd:dateTime` are used to notate points in time.

Figure 2.1 shows a small example RDF graph for President Franklin D. Roosevelt and his pet dog, Fala. The nodes with rounded corners represent resources, whereas the nodes with pointed corners are literals, with their datatype given below their value in parentheses. Each arrow denotes a triple, pointing from the subject to the object, with the predicate written next to the arrow. The graph describes the name and date of birth of the president and his dog, their ownership relation, and their types, including subtype relations: Franklin D. Roosevelt is a person, Fala is a dog, and both persons and dogs are kinds of creatures.

The same graph may also be written textually in several syntaxes. The simplest RDF syntax is N-Triples [14], which lists triples one per line, each terminated with a period. IRIs are enclosed in angle brackets, and literals are enclosed in double quotes, optionally followed by two carets and their datatype (otherwise the implied datatype is `xsd:string`). The N-Triples representation of the same graph as in fig. 2.1 is presented in listing 1.

```
<http://shex.example/FDR> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://shex.example/Human>.
<http://shex.example/FDR> <http://www.w3.org/2000/01/rdf-schema#label> "Franklin Delano Roosevelt".
<http://shex.example/FDR> <http://shex.example/dateOfBirth> "1882-01-30"^^<http://www.w3.org/2001/XMLSchema#date>.
<http://shex.example/FDR> <http://shex.example/pet> <http://shex.example/Fala>.
<http://shex.example/Fala> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://shex.example/Dog>.
<http://shex.example/Fala> <http://www.w3.org/2000/01/rdf-schema#label> "Fala".
<http://shex.example/Fala> <http://shex.example/dateOfBirth> "1940-04-07"^^<http://www.w3.org/2001/XMLSchema#date>.
<http://shex.example/Fala> <http://shex.example/owner> <http://shex.example/FDR>.
<http://shex.example/Human> <http://www.w3.org/2000/01/rdf-schema#subClassOf> <http://shex.example/Creature>.
<http://shex.example/Dog> <http://www.w3.org/2000/01/rdf-schema#subClassOf> <http://shex.example/Creature>.
```

Listing 1: RDF graph for President Franklin D. Roosevelt and his dog

2.2. Wikidata

Wikidata [17] is a free knowledge base and part of the Wikimedia family of projects, the most famous of which is Wikipedia, the free encyclopedia. Its contents are created, maintained and managed by the Wikidata community, most of whose members are volunteers, as well as the members of other Wikimedia projects, e. g. Wikipedia. Anyone can contribute to Wikidata, but the community ensures the quality of the contents with various quality control mechanisms. Providing another such mechanism is part of the motivation for this thesis.

Information on Wikidata is collected in Items, which represent things or concepts. There are Items for individual persons, for cities, states, geographical features, for organizations and corporations. There are also Items for books, films, newspapers, journals, scientific articles, for abstract concepts, phenomena, emotions, philosophical movements, political orientations. And there are even Items for conceptual hierarchies, parent classes, biological taxa, and for fictional characters, places, or other entities. Any thing or concept can be represented by an Item, and while Wikidata will never cover everything that exists in the world, at over fifty million Items and counting, it already provides a wealth of content.

All of these Items follow the same structure. They are identified by their Item ID, a consecutive number prefixed with the letter "Q" (e. g. Q188709 for Beethoven's Symphony No. 5). They can have a Label, a Description, and Aliases, each in various languages: for example, the Item Q7251 is labeled "Alan Turing" in English but «Алан Тьюринг» in Russian; is described as a "British mathematician, logician, cryptanalyst, and computer scientist" in English; and may also be found under search aliases like "Alan M. Turing", "Alan Mathison Turing" or simply "Turing". Items also have a set of Sitelinks (links to pages about the same concept in various other Wikimedia projects – Wikipedia articles, Wikiquote pages, Wikimedia Commons galleries, etc.), and most importantly, a set of Statements.

The Statements are where most of the information in Wikidata is stored. They consist of a Property, such as "place of birth" or "author" or "population", and a value, which can be a reference to another Item, a quantity, a point in time, a piece of text, or a few other possible types. A Statement can also have Qualifiers (further property-value pairs, e. g. clarifying when or where the statement is valid) and References (sets of property-value pairs, listing sources for the Statement), but those are mostly ignored in the context of this thesis. Properties are also identified by an ID, their Property ID (prefixed with the letter "P" instead of "Q"), and can also have Labels, Descriptions and Aliases in different languages: for example, P31 is labeled "instance of" in English and „ist ein(e)" in German. The Labels and Descriptions are necessary to understand the meaning of Statements, but they are not themselves part of the Statements: Statements only list references to Property and Item IDs, making most of the information in Wikidata language-agnostic [5].

Figure 2.2 shows two screenshots of the same Wikidata Item, "Montblanc" (Q761735), viewed in different languages, both as of Wikidata revision 704641959. The page structure is the same regardless of language: first, there is a heading, showing the Label in the current language and the Item ID; below it on the left side is the "term box", listing Labels, Descriptions and Aliases in several languages relevant to the user; below that is the list of Statements; to the right are lists of Sitelinks for different Wikimedia projects. The

screenshots are truncated (the page has also been lightly edited for the screenshots to remove some spacing and a few less relevant page elements) and do not show the full list of Statements, but the Statements pictured are:

- A Statement for the Property P31, where the value is an Item, Q33146843.
- A Statement for the Property P18, where the value is a media file on Wikimedia Commons. Image credit: Mariarosafg (`https://commons.wikimedia.org/wiki/File:Ciutat_de_Montblanc.jpg`), "Ciutat de Montblanc", `https://creativecommons.org/licenses/by-sa/3.0/legalcode`
- Two Statements for the Property P1448, where the values are monolingual text strings. Both Statements also have Qualifiers and References.

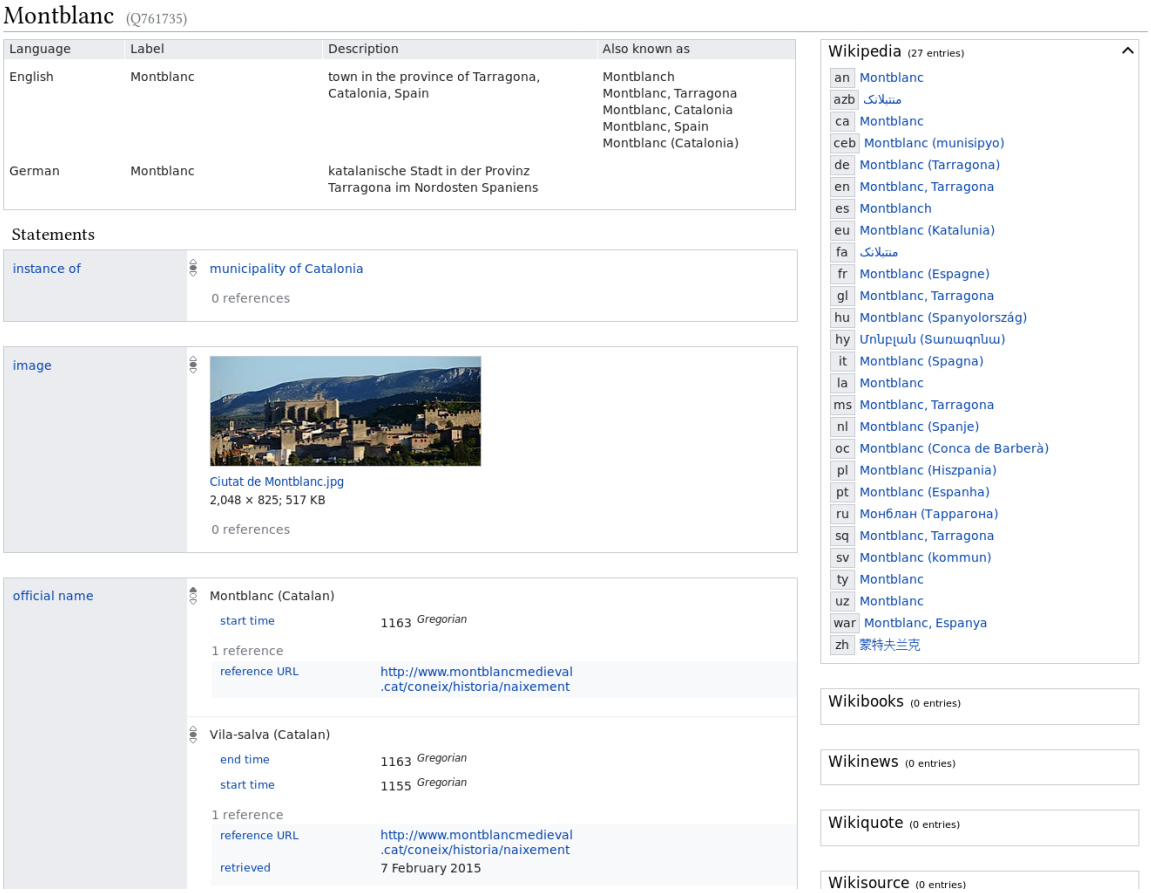
In fig. 2.2a, the Item is viewed as an anonymous user in the default English language (German is shown as a second language in the term box because the request was made from a German IP address; logged in users can configure which languages they want to see here). In fig. 2.2b, the Item is viewed as an anonymous user, explicitly requested in the Catalan language: notice that all referenced Properties and Items are now shown with different Labels.

There is no kind of schema inherent to the Wikidata data model. Any Property can be used on any Item: nothing in the software stops one from adding, say, a "date of birth" Statement to an Item for a lake, or a "parent taxon" Statement to an Item for a movie teaser poster. The community has several ways to describe schemas to varying degrees of formality (such as Property lists on WikiProject pages or the property constraints system), but they are all realized by community consensus, not enforced by the software behind Wikidata. The great flexibility which this lends to the community is considered to be one of Wikidata's greatest strengths [16], and though the use of Shape Expressions on Wikidata will provide another, highly formal way to describe schemas, there is no intention to change this fundamental operating principle of Wikidata.

Wikidata's data model, as described above, is not directly related to RDF. However, to enable usage of RDF technologies and interoperation with RDF-based data sets, Wikidata's data is exported to RDF: the data about any Item can be downloaded in various RDF formats through a Linked Data interface, and a full, up-to-date RDF export of Wikidata is available in the Wikidata Query Service (WDQS), a SPARQL endpoint to which anyone may submit queries. In the Wikidata RDF exports, items use the prefix `wd:` and properties in a statement use the prefix `wdt:`. For example, the statement

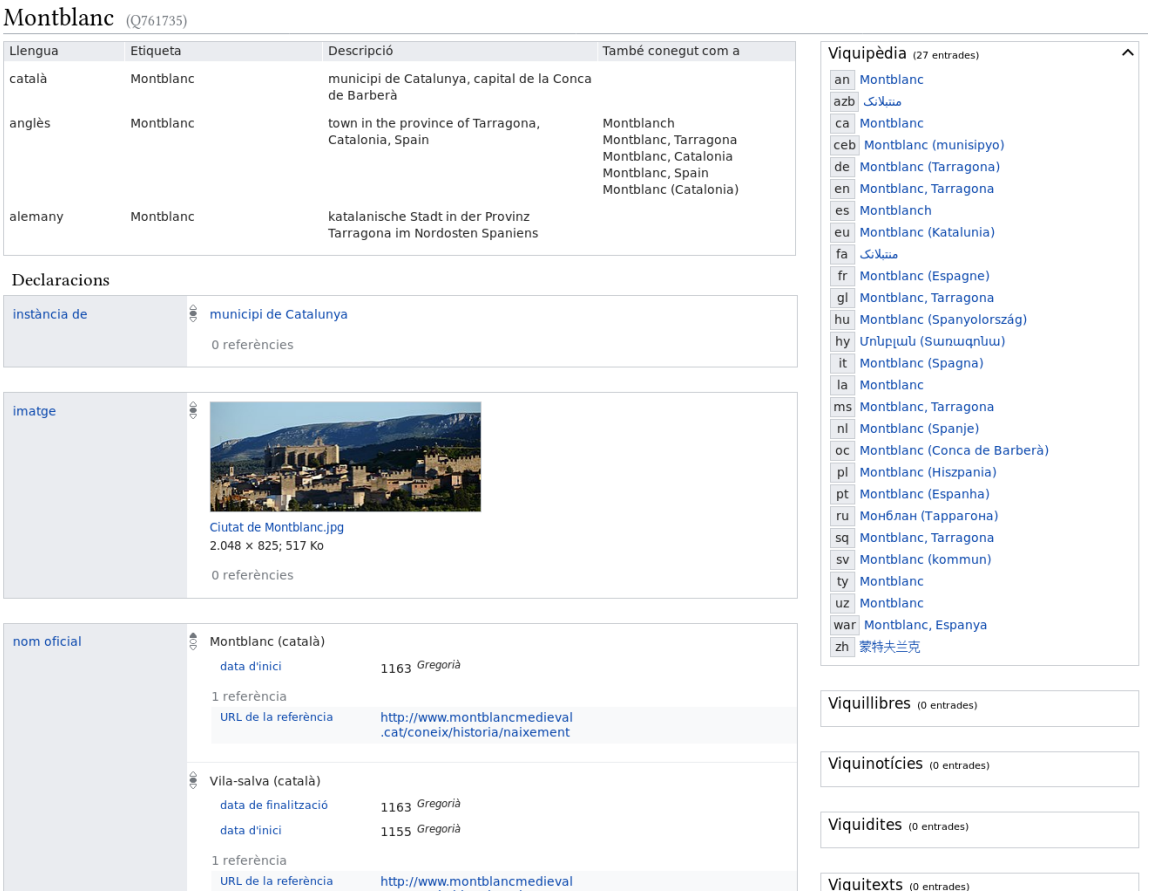
"Alan Turing" (Q7251)'s "place of birth" (P19) was "Maida Vale" (Q122744)

is represented in RDF as the triple `wd:Q7251 wdt:P19 wd:P12274`.



(a) The Item viewed as an anonymous user in the default English language

Figure 2.2.: Two screenshots of the same Item viewed in different languages



(b) The Item viewed as an anonymous user, explicitly requested in the Catalan language

Figure 2.2.: Two screenshots of the same Item viewed in different languages

2.3. Shape Expressions

Shape Expressions (ShEx) [11] is a standard for describing data shapes within an RDF graph, developed by the ShEx Community Group under the umbrella of the W3C. A ShEx schema consists of a number of shapes, each of which describes the layout of an RDF resource, called the focus node. The notation for ShEx schemas used in this thesis is ShEx Compact Syntax (ShExC), and while a full introduction to ShEx and ShExC is beyond the scope of this thesis, the relevant elements are described below. For a more complete and detailed introduction, see the ShEx Primer [1].

In ShExC, a shape definition begins with the shape's name (an IRI) and is followed by a block of triple constraints enclosed in curly braces (`{}`). The triple constraints are separated by semicolons (`;`) and place restrictions on triples with the focus node as the subject and a certain predicate: a triple constraint consists of the IRI for the predicate and then a constraint on the object (the value). The value constraint can require the value to be a literal with a certain datatype, expressed directly via the datatype's IRI, or it can require the value to be a resource matching another shape, indicated by that shape's name (an IRI) following an `@` symbol. The value constraint may be followed by a cardinality for the triple constraint: instead of the default cardinality "must occur exactly once", the symbol `?` may be appended to signify "zero or one times" (an optional triple constraint), `*` means "zero or more" (any number of triples with this predicate may occur, but if they do occur, their objects must still match the value constraint), and `+` means "one or more". Multiple possible triple constraints for the same predicate can be expressed by grouping them inside parentheses (`()`), separated by vertical bars (`|`).

```
ex:CreativeWork {
  ex:title rdf:langString;
  (
    ex:author @ex:Person+ |
    ex:author @ex:Organization
  );
  ex:cites @ex:CreativeWork*;
  ex:publicationDate xsd:dateTime?
}

ex:Person {
  ex:name rdf:langString;
  ex:dateOfBirth xsd:dateTime?
}

ex:Organization {
  ex:name rdf:langString+
}
```

Listing 2: Example schema for creative works and their authors

For example, the schema in listing 2 defines three shapes for creative works and their authors. It describes a creative work as having a title, a string in a certain language; being written by one or more persons or an organization; citing any number of other creative works; and optionally having been published on a certain date and time. (Scholarly articles would typically have human authors, cite multiple works and have a publication date, whereas a documentation page could be attributed to an organization as a whole and have no significant citations or publication date.) A person in this schema has exactly one name in one language (which is not generally true [9], but violations of this should be rare enough that they are worth investigating as potential errors) and may have a date of birth, if known. An organization does not have any properties other than at least one name (it may have several, e. g. in different languages or jurisdictions).

Full examples of data sets matching or violating this schema in some way are beyond the scope of this section, but reasons why a node may not match one of the shapes in listing 2 include:

- A creative work is missing a title.
- A person is missing a name.
- An organization has a name which is a plain string not tagged with a language code.
- A creative work is authored by more than one organization, all of which have more than one name. (Organizations with a single name are indistinguishable from persons as far as this schema is concerned.)
- A creative work cites a work without a title.
- A creative work was authored by a person whose date of birth is a numeric literal.
- A creative work cites a work which cites a work which does not have any authors.

As the last examples show, a node may fail to match a shape not just because of that node's data, but also because of the data of nodes that the node links to, directly or indirectly: in the last example, the chain of citations between the original focus node and the work that is missing authors can be arbitrarily long.

By default, ShEx shapes are not "closed": the graph may contain additional triples with the focus node as the subject whose predicates do not match any of the triple constraints in the shape. For example, an organization with an `ex:dateOfIncorporation` would still match the `ex:Organization` shape in listing 2 even though that schema does not mention an `ex:dateOfIncorporation` predicate anywhere. (ShEx also supports closed shapes, where this would be a violation, but they are never used in the context of this thesis.)

2.4. RDF2Graph

RDF2Graph [3] is a tool to automatically determine the structure of an RDF graph and export it as a ShEx schema (other output formats are also supported). It relies heavily on

the type information of each node and the class hierarchy in the graph, determining the valid predicates and their value types and cardinalities for each type in the graph. There is also an optional step to simplify the resulting schema.

To discover the structure of an RDF graph, RDF2Graph runs a set of queries against a SPARQL endpoint serving that graph. (This can be a local server, perhaps simply based on a single file containing the graph, or a remote server.) It enumerates all the classes that occur in the graph and then collects all the predicates that are used on instances of each class. For each predicate of each class, it then gets the types referenced in the values for those predicates (usually the classes of the referenced nodes, but values can also be literals or external references), as well as forward and reverse multiplicity for each such type link. All this information is stored in a separate RDF graph private to RDF2Graph.

If the simplification step is enabled, RDF2Graph will afterwards apply some transformations to the structure based on the hierarchy of the classes involved, which happens in several steps. The following description uses the same step numbers as [3], but elides several steps which are mostly implementation details, which is why the step numbers are not consecutive here. See [3] and the RDF2Graph source code for the full description. (In the source code, the steps are counted as sub-steps of the general step 7.4 "simplification": for example, step 2 is implemented under a code comment for "7.4.2".)

In step 2, all predicates from child classes are copied to parent classes. For example, in listing 3, the `ex:WebResource` and `ex:CreativeWork` classes are initially empty but then inherit several predicates from their child classes. (The `.` in the `ex:url` value constraint means "any value".)

Step 3 removes predicates which are only found in a single subclass from the parent class again. In the example from listing 3, the `ex:cites` and `ex:url` predicates will be removed from the `ex:CreativeWork` shape because they are only found in one subclass (`ex:ScholarlyArticle` and `ex:WebResource`, respectively), while the other predicates will be kept. In the `ex:WebResource` shape itself, all predicates will be kept because they occur in both subclasses, `ex:BlogPost` and `ex:DocumentationPage`. See listing 4 for the result.

After that, in step 4 all references which are still found in a parent class are removed from the child classes, where they are now redundant. In listing 5, this mostly clears the schema: most predicates are found in `ex:CreativeWork`, and only `ex:WebResource` retains another predicate, `ex:url`. Note that this step was later disabled, see section 3.1.

Step 2 also merges references to parent classes, and the other steps take this into account by respecting subclass relations as well. For example, consider the schema in listing 6. Initially, it states that scholarly articles only cite other scholarly articles, and web resources only cite other web resources; after simplification, it states that creative works cite other creative works of any kind: the references to `ex:ScholarlyArticle` and `ex:WebResource` were merged while copying the `ex:cites` predicate into the `ex:CreativeWork` parent class, and afterwards the individual predicates were removed from the child classes because they are covered by the merged reference in the parent class.

At this point, the information which RDF2Graph extracted is available in a private RDF graph. From there, it can be exported into various formats by different exporters. The ShEx exporter loads parts of these results into a temporary RDF database of its own, applies some transformations to them via SPARQL, converts them to JSON-LD, further transforms the JSON, and finally exports the result into ShExC text via the Jade template engine.

```
ex:CreativeWork {}
ex:ScholarlyArticle
    EXTENDS ex:CreativeWork {
  ex:title rdf:langString;
  ex:author @ex:Person+;
  ex:cites @ex:CreativeWork+;
}
ex:WebResource
    EXTENDS ex:CreativeWork {}
ex:BlogPost
    EXTENDS ex:WebResource {
  ex:url .;
  ex:title rdf:langString;
  (
    ex:author @ex:Person |
    ex:author @ex:Organization
  );
}
ex:DocumentationPage
    EXTENDS ex:BlogPost {
  ex:url .;
  ex:title rdf:langString;
  ex:author @ex:Organization;
}
```

(a) Before step 2

```
ex:CreativeWork {
  ex:url .;
  ex:title rdf:langString;
  (
    ex:author @ex:Person+ |
    ex:author @ex:Organization
  );
  ex:cites @ex:CreativeWork+;
}
ex:ScholarlyArticle
    EXTENDS ex:CreativeWork {
  ex:title rdf:langString;
  ex:author @ex:Person+;
  ex:cites @ex:CreativeWork+;
}
ex:WebResource
    EXTENDS ex:CreativeWork {
  ex:url .;
  ex:title rdf:langString;
  (
    ex:author @ex:Person |
    ex:author @ex:Organization
  );
}
ex:BlogPost
    EXTENDS ex:WebResource {
  ex:url .;
  ex:title rdf:langString;
  (
    ex:author @ex:Person |
    ex:author @ex:Organization
  );
}
ex:DocumentationPage
    EXTENDS ex:WebResource {
  ex:url .;
  ex:title rdf:langString;
  ex:author @ex:Organization;
}
```

(b) After step 2

Listing 3: Simplification step 2 (in ShExC-like pseudo-syntax)

```
ex:CreativeWork {
  ex:url .;
  ex:title rdf:langString;
  (
    ex:author @ex:Person+ |
    ex:author @ex:Organization
  );
  ex:cites @ex:CreativeWork+;
}
ex:ScholarlyArticle
    EXTENDS ex:CreativeWork {
  ex:title rdf:langString;
  ex:author @ex:Person+;
  ex:cites @ex:CreativeWork+;
}
ex:WebResource
    EXTENDS ex:CreativeWork {
  ex:url .;
  ex:title rdf:langString;
  (
    ex:author @ex:Person |
    ex:author @ex:Organization
  );
}
ex:BlogPost
    EXTENDS ex:WebResource {
  ex:url .;
  ex:title rdf:langString;
  (
    ex:author @ex:Person |
    ex:author @ex:Organization
  );
}
ex:DocumentationPage
    EXTENDS ex:WebResource {
  ex:url .;
  ex:title rdf:langString;
  ex:author @ex:Organization;
}
```

(a) Before step 3

```
ex:CreativeWork {
  ex:title rdf:langString;
  (
    ex:author @ex:Person+ |
    ex:author @ex:Organization
  );
}

ex:ScholarlyArticle
    EXTENDS ex:CreativeWork {
  ex:title rdf:langString;
  ex:author @ex:Person+;
  ex:cites @ex:CreativeWork+;
}
ex:WebResource
    EXTENDS ex:CreativeWork {
  ex:url .;
  ex:title rdf:langString;
  (
    ex:author @ex:Person |
    ex:author @ex:Organization
  );
}
ex:BlogPost
    EXTENDS ex:WebResource {
  ex:url .;
  ex:title rdf:langString;
  (
    ex:author @ex:Person |
    ex:author @ex:Organization
  );
}
ex:DocumentationPage
    EXTENDS ex:WebResource {
  ex:url .;
  ex:title rdf:langString;
  ex:author @ex:Organization;
}
```

(b) After step 3

Listing 4: Simplification step 3 (in ShExC-like pseudo-syntax)

```
ex:CreativeWork {
  ex:title rdf:langString;
  (
    ex:author @ex:Person+ |
    ex:author @ex:Organization
  );
}
ex:ScholarlyArticle
    EXTENDS ex:CreativeWork {
  ex:title rdf:langString;
  ex:author @ex:Person+;
  ex:cites @ex:CreativeWork+;
}
ex:WebResource
    EXTENDS ex:CreativeWork {
  ex:url .;
  ex:title rdf:langString;
  (
    ex:author @ex:Person |
    ex:author @ex:Organization
  );
}
ex:BlogPost
    EXTENDS ex:WebResource {
  ex:url .;
  ex:title rdf:langString;
  (
    ex:author @ex:Person |
    ex:author @ex:Organization
  );
}
ex:DocumentationPage
    EXTENDS ex:WebResource {
  ex:url .;
  ex:title rdf:langString;
  ex:author @ex:Organization;
}
```

(a) Before step 4

```
ex:CreativeWork {
  ex:title rdf:langString;
  (
    ex:author @ex:Person+ |
    ex:author @ex:Organization
  );
}
ex:ScholarlyArticle
    EXTENDS ex:CreativeWork {}

ex:WebResource
    EXTENDS ex:CreativeWork {
  ex:url .;
}

ex:BlogPost
    EXTENDS ex:WebResource {}

ex:DocumentationPage
    EXTENDS ex:WebResource {}
```

(b) After step 4

Listing 5: Simplification step 4 (in ShExC-like pseudo-syntax)

```
ex:CreativeWork {}

ex:ScholarlyArticle
    EXTENDS ex:CreativeWork {
  ex:cites @ex:ScholarlyArticle+
}

ex:WebResource
    EXTENDS ex:CreativeWork {
  ex:cites @ex:WebResource*
}
```

(a) Before simplification

```
ex:CreativeWork {
  ex:cites @ex:CreativeWork*
}

ex:ScholarlyArticle
    EXTENDS ex:CreativeWork {}

ex:WebResource
    EXTENDS ex:CreativeWork {}
```

(b) After simplification

Listing 6: Simplification (in ShExC-like pseudo-syntax), with class relations (this example is independent from the previous example)

2.5. Wikimedia Toolforge

Wikimedia Toolforge is a hosting environment provided by the Wikimedia Foundation where trusted members of the Wikimedia community may host and develop their tools or other work. Most tools are web-based and reachable under the tools.wmflabs.org domain, but it is also possible to run bots or analysis jobs on Wikimedia Toolforge. Tools can be written in a variety of languages (e. g. PHP, Python, Java, JavaScript) and have access to live replicas of the Wikimedia production databases, data dumps of Wikimedia projects, custom tool-specific databases, and a Sun Grid Engine job execution system for long-running or resource-intensive tasks.

3. Applying RDF2Graph to Wikidata

This chapter describes the changes that were made to RDF2Graph and related software in the course of this thesis. Some of them are general improvements unrelated to Wikidata (section 3.1), some are motivated by the Wikidata use case but still generally useful (sections 3.3 to 3.5), and some are specific to Wikidata and result in a version of RDF2Graph that cannot be used for other RDF graphs (section 3.2). All the changes are available in the source code repositories at `https://github.com/lucaswerkmeister/RDF2Graph` and `https://github.com/lucaswerkmeister/RDFSimpleCon`, in the `master` and `wikidata` branches; some of them may also be merged into the upstream RDF2Graph and RDFSimpleCon repositories in the future.

3.1. General RDF2Graph Updates

The `master` branch of the RDF2Graph source code repository has not seen any updates since 2015 (when [3] was published), so some updates were needed to make the program run on newer software versions and to target a more recent version of ShEx. Many of these were rather minor in nature, but a few of the major ones concerning the ShEx exporter are described in this section. (I later noticed that the RDF2Graph source code repository also has a `dev` branch, which has some more recent changes including a few fixes, but most of them are not relevant to the issues discussed here.)

The final step of the ShEx export was mostly rewritten from scratch. This is the step that translates a JSON-LD file describing the shapes of the schema into a ShExC file; the previous version did this using Jade, a templating engine for HTML documents (renamed to Pug in 2016), constructing an HTML document containing a single plain-text **`<pre>`** element, then extracting this text from the document using an HTML-to-text converter. The new version directly produces the text from JavaScript and includes several improvements: datatypes are now properly supported, whereas the previous version only supported a handful of hard-coded IRIs as datatypes and exported all other datatypes as if they were actually shapes; prefixes are used everywhere in the output, highly improving readability; and the output is sorted, making the whole generation process more deterministic and thus making it easier to compare the results of multiple inference processes.

The syntax of the generated ShExC schemas also required some changes. First, RDF2Graph originally targeted ShEx version 1.0 whereas the current release is version 2.0, which introduced some syntactic changes (e. g. replacing some commas with semicolons). Second, RDF2Graph emitted syntax for an experimental, in-progress version of ShEx with support for inheritance between shapes: if, for example, the inferred schema contained classes for both “human” and “person”, and “human” was a subclass of “person” in the input data set, then RDF2Graph would declare that the “human” shape extended the “person” shape

and omit predicates from the "person" shape in the "human" shape, since those would be redundant with the predicates inherited from the "person" shape. This feature has not made it into any released version of ShEx yet (it can still be found in `on-shape-expression` branches of related repositories, though the syntax has slightly changed: it now uses the keyword `EXTENDS` whereas RDF2Graph used the symbol `&`), so the ShEx exporter was changed to only mention the parent classes for a shape in a comment, and simplification step 4 (see section 2.4) was disabled so that the redundant predicates are included after all, since they are no longer automatically inherited.

3.2. Wikidata Support

This section outlines the general process that is used when running RDF2Graph against data from Wikidata and then describes several changes to RDF2Graph that were necessary to support this. (The subsequent subsections correspond to those changes, not to the individual steps of the process.)

3.2.1. Overall process

Usually, RDF2Graph is run against a SPARQL endpoint which serves the RDF graph for which one wants to infer the structure. However, simply running RDF2Graph against the Wikidata Query Service (WDQS), the public SPARQL endpoint for Wikidata, would mean attempting to infer a schema from all of Wikidata, which is neither the goal of this thesis (that is to infer schemas from a small set of exemplary Items) nor even remotely feasible given the amount of data in Wikidata. For example, the 2018-10-08 full Wikidata dump, in N-Triples format, is 1.1 TB large after decompression and contains 7 523 105 374 triples.

Instead, a process was set up which, given a query which selects the exemplary Items, downloads the related data for them and runs RDF2Graph against it. The process is outlined in fig. 3.1 and controlled by a Makefile in the RDF2Graph repository, so that after creating an *`example`*`.entities.sparql` file with a SPARQL query selecting the exemplary items, it is sufficient to run `make` *`example`*`.shex` to run the entire process and generate the ShEx file.

First, the query selecting the exemplary Items is transformed into a query selecting all the required data, using the Unix `sed` command. The generated query selects all Statements of the exemplary Items ("direct Statements"), all Statements of Items which occur as values of the direct Statements ("indirect Statements"), and all "instance of" (P31) Statements of Items which occur as values of the indirect Statements. This query is run against WDQS, which returns the results in JSON format; they are then piped into a script for the `jq` tool ("JSON query"), which transforms them into N-Triples format, and stored in an N-Triples file.

Next, the Apache Jena Fuseki SPARQL server is run locally, serving the data from this N-Triples file. As soon as it has finished loading the data file, RDF2Graph is run against this local SPARQL endpoint for the main inference process, including RDF2Graph's simplification step. Once RDF2Graph terminates, the Fuseki server is stopped.

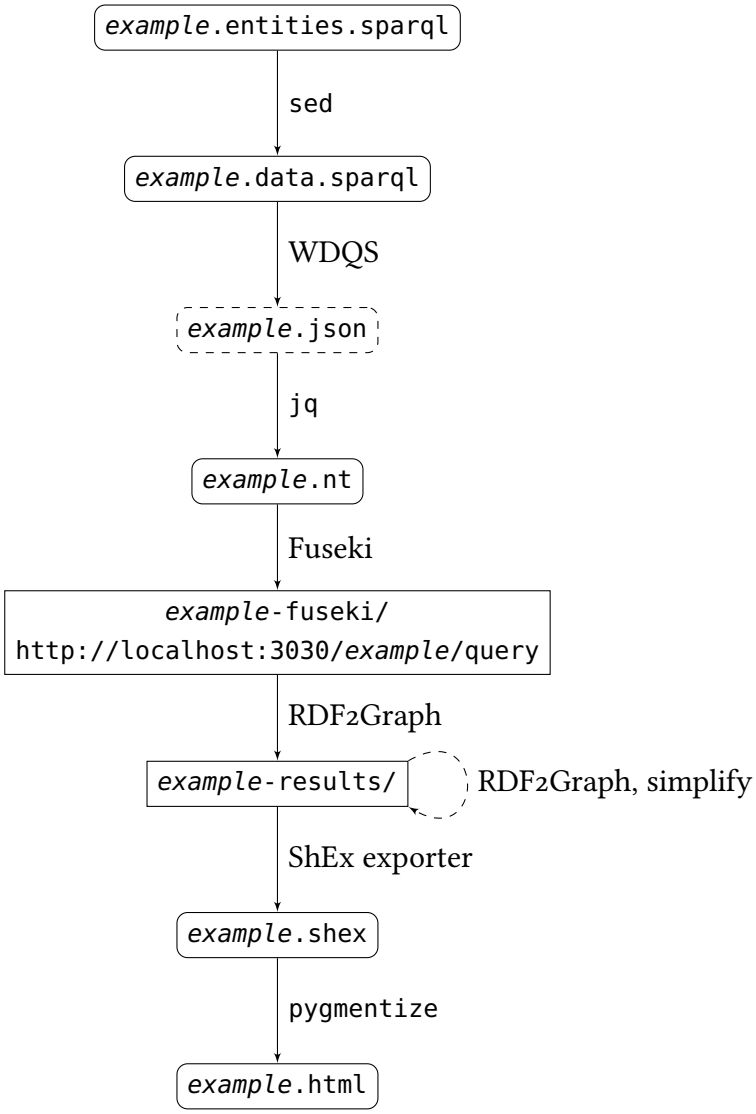


Figure 3.1.: Overview of the process

Afterwards, the results from RDF2Graph are available as a graph of classes with associated information, and the RDF2Graph ShEx exporter is run to produce a ShEx file. If desired (`make example.html`), that file can also be turned into an HTML file using the Pygments syntax highlighter, which makes the ShEx code more readable.

3.2.2. Type predicates

RDF2Graph heavily relies on the type information of the RDF graph it inspects: it uses the `rdf:type` predicate to map a node to its shape (assuming a one-to-one mapping between classes and shapes) and the `rdfs:subClassOf` predicate to determine the parent classes of a class, which are used in the simplification step. However, Wikidata does not use these standard predicates: the class(es) and superclass(es) of an item are regular Statements like any other Wikidata Statement, using the Properties "instance of" (P31) and "subclass of" (P279). In the RDF export, `rdf:type` and `rdfs:subClassOf` are only used as part of the meta-model, assigning each item the class `wikibase:Item`, a subclass of `wikibase:Entity`.

To use the type information within the data instead of this meta-model, the queries which RDF2Graph uses to explore the input graph were changed to use the predicates `wdt:P31` and `wdt:P279` instead of `rdf:type` and `rdfs:subClassOf`. (Note that RDF2Graph still uses `rdf:type` and `rdfs:subClassOf` when writing its internal results graph, so queries operating on that graph as part of the simplification step still use those predicates.)

3.2.3. Data reduction

The Labels, Descriptions and Aliases of an Item can make up a large portion of its data, but are generally not interesting for ShEx schemas, since there is rarely more to say about them than "an Item has one or more Labels, zero or more Descriptions, and zero or more Aliases". (A manually curated schema might go beyond this – perhaps requiring, for example, that Items about Spanish municipalities have a label in Spanish – but such details are beyond the ability of RDF2Graph to infer.) Additionally, Wikidata Items often have a number of Statements listing external identifiers for the Item (that is, identifiers in other databases for the same concept described by this Item): for example, Wikidata's Q80 corresponds to no99010609 in the Library of Congress (LoC), 85312226 in the Virtual International Authority File (VIAF), nm3805083 in the Internet Movie Database (IMDb), @timberners_lee on Twitter, and dozens of other external identifiers. All these external identifiers are technically just ordinary Statements, but since they carry a rather different kind of information than other Statements, they are sorted into a separate section when viewing the Item on the Wikidata website.

In order to reduce the runtime of the RDF2Graph inference process, and to reduce clutter in the inferred schemas, the Labels, Descriptions, Aliases, and external identifiers of an Item are excluded when downloading the data for a set of exemplary Items: the generated query selecting the required data (*`example`*`.data.sparql` in fig. 3.1) only selects triples for statements of non-external identifier properties.

3.2.4. Full type hierarchy

One undesirable consequence of running RDF2Graph against a reduced data set is that RDF2Graph cannot see the full type hierarchy of the classes involved: for example, depending on how much data was downloaded, it may or may not be aware that the classes "island nation" (Q112099) and "sovereign state" (Q3624078) have a common superclass, "state" (Q7275). And if RDF2Graph does not know that two classes have a common superclass, it cannot merge them during the simplification step.

To avoid this, an option was added to RDF2Graph which allows specifying an alternative SPARQL endpoint for all queries that require a full view of the data, and this alternative endpoint is used for the query to get all parent and child classes of a class. (If the option is not specified, it falls back to the default SPARQL endpoint.) The Makefile mentioned in section 3.2.1 specifies the WDQS SPARQL endpoint for this option, so that RDF2Graph can discover the full class hierarchy around all relevant items even when running against a subset of Wikidata.

3.2.5. Simplification

RDF2Graph's simplification step would originally merge all classes into their superclasses, almost completely unconditionally, stopping only at the root class `owl:Thing`. This works well for RDF graphs with a lot of different subclasses directly beneath `owl:Thing`, but is much too aggressive for Wikidata: not only does Wikidata have a different root class ("entity" (Q35120)), but it also has a complex hierarchy of abstract classes below that root class, and merging other classes into those abstract classes (like "concept" (Q151885), "abstract object" (Q7184903) or "subject" (Q830077)) would not result in useful schemas.

Therefore, step 2 in the simplification process was adapted to add a number of Wikidata Items to the set of classes which other classes should not be merged into (originally only containing `owl:Thing`), and to stop looking for common superclasses of two classes after walking up five levels in the class hierarchy. (The limit of five levels is an arbitrary choice and could also be made configurable if deemed necessary in the future, but it seems to work out well enough in practice.)

3.3. Support for Cyclic Type Hierarchies

As outlined in section 2.4, RDF2Graph has an optional simplification feature where the schema will be simplified in several steps, based on the type hierarchy of the classes involved. For this, the type hierarchy is loaded into an in-memory data structure, which in the code is referred to as a "tree". In fact, the data structure forms a generic, directed graph, where each node may have any number of children (subclasses) and any number of parents (superclasses), and no restrictions on the structure are enforced during construction. However, while the algorithms subsequently operating on the data structure support classes with multiple parent classes, they do assume that the graph is acyclic, i. e., that no class is an indirect subclass of itself.

This assumption makes sense in general, since a type hierarchy is usually assumed to be acyclic: if there is a cycle in the subclass relations between a list of classes, that

effectively renders all these classes equivalent, since any instance of any one of these classes is then also an (indirect) instance of all the other classes. However, occasional emergence of subclass cycles is almost unavoidable on Wikidata: each of the subclass links may appear reasonable for just the two Items it connects, and an editor looking at these Items will not necessarily notice that a cycle has been formed, since the other statements forming the cycle are not visible when looking at these two Items. For example, at the time of writing, there exists a cycle in Wikidata between the following classes, each a subclass of the next: "representation" (Q4393498), "property" (Q937228), "category of being" (Q714737), "concept" (Q151885), "mental representation" (Q2145290), "representation" (Q4393498). It was first introduced in Wikidata revision 726473098, when the parent class of "mental representation" (Q2145290) was changed from "representation" (Q1272626) to "representation" (Q4393498), and reinforced in Wikidata revision 726473565; the latter revision was later reverted in Wikidata revision 735108434, citing the created subclass cycle as part of the reason, but as the first revision was not reverted nor the Item otherwise edited since then, the cycle persists so far. The fact that editors work in different languages, and different items may share the same label in some languages (see Q1272626 and Q4393498, both "representation", above), contributes to the problem that such cycles can be introduced accidentally and the correct way to break them is not always obvious.

While, by and large, these cycles are eventually found and fixed by the community, it is possible that, in the meantime, RDF2Graph will run on a subclass graph containing cycles. Therefore, to make the process more robust, several algorithms in the simplification step were adjusted to be able to cope with cyclic graphs. This is valuable even though the results in the cycle may no longer make sense, since it means that a cycle somewhere in the class hierarchy, typically among very abstract classes, will no longer impede simplification in other, more concrete classes, where simplification is much more useful anyways. Without these improvements, if any part of the simplification step failed (typically with a `StackOverflowError`), all simplification results would be lost and the ShEx export would run on the original, completely unsimplified graph.

3.3.1. GetAllChildren

A general component of the new algorithms are methods to get all children and parents of a node, in a matter that is guaranteed to terminate even if there are cycles among the children or parents. The method for collecting all child nodes is outlined in algorithm 1, and the method for collecting all parent nodes is completely analogous. Since each node can only enter the return set once, the queue only grows when nodes are added to the return set, and each iteration takes one node from the queue, the algorithm must eventually exhaust the queue and terminate, provided that the set of nodes reachable from the initial node is finite.

3.3.2. Counting instances

With this in place, it becomes easy to reimplement the method which counts the number of direct and indirect instances of each class, based on the direct instance counts on each class node. Previously, the method operated recursively, aggregating instance counts for

Algorithm 1 Algorithm to collect all direct and indirect child nodes in a graph

```
function GetAllChildren(node)
    initialize set of all children (empty)
    initialize working queue of children (empty)
    add to queue ← direct children of node
    while queue ≠ ∅ do
        child ← take from queue
        if child ∉ set then
            add to set ← child
            add to queue ← direct children of child
        end if
    end while
    return set
end function
```

each indirect subclass of a class all the way up to the class itself, and this recursion would never terminate when encountering a cycle of subclasses. Instead, the new implementation collects all direct and indirect subclasses up-front using algorithm 1 and then sums up their direct instance counts.

3.3.3. Simplification steps 3 and 4

Two steps of the simplification process remove some of the temporary (added) type links of a node, based on the temporary type links of its neighboring nodes: step 3 removes temporary type links from parent classes that are only found in one child class, and step 4 removes temporary type links from child classes that are made redundant by a type link in the parent class. Both steps were originally implemented as inspecting the neighboring nodes' temporary type links and then directly manipulating the node's own temporary type links. This required that each step traversed the nodes in the correct order (parents before children for step 3, children before parents for step 4) and was implemented by traversing the graph recursively, which is problematic if the graph contains cycles.

This problem was solved by splitting up the analysis and modification of the temporary type links: each step now has two parts, where the first part marks all temporary type links that should be removed, and the second part actually removes them from a node's temporary type links. As long as the first part ("mark") for all nodes is run before the second part ("remove") for any node, these parts can visit the nodes in any order, and instead of recursively traversing the graph, both steps now first collect all child nodes of the root node and then iterate them twice, running the first part ("mark") the first time and the second part ("remove") the second time.

3.3.4. Node distances

RDF2Graph also has a method to determine the distance of all indirect parents from a certain node, which is used when searching for common superclasses of a class. Like the

other methods, this was implemented recursively, setting the distance of a parent node to the current value of a counter and then calling the same method on all parent nodes of that node with the counter incremented by one. In this case, the recursion was kept, but the method was adjusted to only visit a parent node if it has not been visited before, or if its current distance is larger than the counter value. The second part of the condition is necessary to ensure that the distance is set correctly if a node is visited first via a longer path and then later again via a shorter path, which is possible since the method implements a depth-first search instead of a breadth-first search.

3.3.5. Parent check

There is also a method to determine whether a certain node is a (direct or indirect) parent node of another node, originally implemented recursively. This was replaced by a version which first retrieves the set of the potential child node's parent nodes, using algorithm 1, and then checks whether the potential parent node is an element of it.

3.4. Schema Reduction

An unexpected problem emerged once the schema inference itself was in place: it proved to be almost impossible to validate Items against the inferred schemas for all but the most simple data sets. Two ShEx implementations were evaluated: shex.js, a JavaScript implementation running in Node.js, and shex-java, a Java implementation offering two different validation algorithms [2]. shex.js would usually crash with an out-of-memory error from Node.js within a few minutes, whereas shex-java would run for hours on end without any output or apparent progress (with either algorithm).

One attempted strategy to be able to validate the inferred schemas was to reduce the size of the schemas by dropping some less-relevant elements. This was part of the motivation for dropping Labels, Descriptions, Aliases, and external identifiers of an Item from the data set that the inference would be run against (see section 3.2.3); however, reducing the input data set is not the only way to reduce the resulting schema: it is also possible to further trim the schema after the inference process has finished. This was implemented in the final step of the ShEx exporter, the `shexbuilder.js` program, which had already been mostly rewritten for different reasons (see section 3.1).

Elements can be removed from a schema on three different levels: an individual type link can be removed from a predicate, a predicate can be removed from a shape, or an entire shape can be removed from the schema. In each case, the decision on whether to keep or drop the element requires an assessment of whether the element *can* be dropped, and whether it *should* be dropped.

Shapes can be removed from a schema as long as they are not referenced by any other shape, and as long as they are not interesting as "entry points" for the schema (i. e., shapes to validate the initial focus node against). The second condition is subjective, but in practice, shapes that are interesting as "entry points" are usually also referred to by other shapes, so the condition can be ignored. The first condition is implemented in the exporter by maintaining two sets of shapes: one for shapes which are referenced from type links,

and one for shapes which should perhaps be dropped. When printing the final schema, only shapes found in the latter set but not in the former are skipped.

Predicates can be removed from a shape unconditionally: since the shapes produced by RDF2Graph are not closed, removing a predicate from a shape means that triples using that predicate will now be ignored by the validator when testing against the shape. However, a shape without predicates is hardly useful, so a balance needs to be struck when deciding whether a predicate *should* be dropped.

Individual type links can, in principle, never be removed from a predicate: if a type link occurs in the schema, then there must have been a corresponding triple in the input data, and removing the type link would turn that triple into a violation, at least if simplification was applied to the schema. (Without simplification, some type links may be redundant due to other type links for related classes.) However, this assumes a perfect input data set which should not have any violations, which is an unrealistic assumption at least for Wikidata input data: even if the exemplary input Items were carefully selected and have no incorrect Statements themselves, it is unlikely that all the Items which those Items link to are also flawless. (See section 5.1 for an example of problems in a high-profile input data set.) Given this situation, it is defensible to drop type links which are likely to reflect errors in the input data.

The decision whether to drop an element or not is implemented as a simple numeric threshold on all three levels. RDF2Graph tracks the number of instances of each class it encounters, and this information is still available to the ShEx exporter, which can therefore drop type links that are used less than m times, predicates that are used less than n times (sum of all type link uses), and shapes for classes with less than o instances. The three limits are independent, though in practice it makes most sense to choose them such that $m > n > o$. An alternative strategy to explore in the future might be to turn those fixed thresholds into ratios, so that the same parameters can be used for vastly different data sets with varying numbers of class instances overall.

Unfortunately, this strategy was not very successful in enabling validation of the schemas. Only at almost draconian thresholds for dropping elements, where only a few shapes and predicates would remain in the schema, would validation succeed; manual bisecting of the schemas which failed to validate would often turn up a fairly small problematic part of a schema which a simple count-based threshold was ill-suited to detect automatically. Therefore, while the code implementing the limits remains in place, all three are disabled by default.

3.5. Depth Limits in Validation

Another attempt to make validation of the inferred schemas more feasible was to limit the maximum depth during the validation process. Consider, for example, a simple shape for humans, which describes a human as having any number of human parents and a date of birth. To validate whether any given node is actually a human, all its direct and indirect parents must also be validated, potentially thousands of ancestors, no matter how remote. If the shape is more complicated, with more predicates potentially linking to other shapes as well, this can get very expensive rather quickly. On the other hand, it

is not clear that the more remote validations here are useful, at least in the context of validating Wikidata Items (where different areas of Items may be edited by very different sets of editors) against automatically inferred schemas (which are likely to have inherited some imperfections from the input data): most inferred schemas tend to include shapes for certain core classes, e. g. "human" (Q5) or "sovereign state" (Q3624078), but someone attempting to validate an Item for a mammalian protein is unlikely to care that the Item for the protein's discoverer's husband's country of citizenship happens to be missing an "inflation rate" (P1279) statement.

Therefore, in an attempt to reduce the amount of irrelevant violations and to make the validation terminate without crashing, a patch to add an optional depth limit to the shex.js implementation was developed. With this change, if the depth limit was reached, the validator would skip recursing into another shape and instead only record the fact that the limit was reached before continuing.

Unfortunately, however, this proved unsuccessful. When testing some Items against several inferred schemas, low limits (2–4) would yield unpredictable results (sometimes validate successfully, sometimes report problems, sometimes crash), whereas higher limits (up to 10) would typically have the same result as validation without any depth limit (either validation failure or crash), though sometimes with wildly varying runtimes. (More details can be found in appendix A.2.) One possible explanation for this is that pruning one part of the validation process when reaching the depth limit may cause the validator to spend more time in other areas, which may previously not have been necessary if the pruned part would otherwise have resulted in a violation.

4. The Wikidata Shape Expressions Inference Tool

This chapter describes the major user-facing result of this thesis: the Wikidata Shape Expressions Inference tool, which makes the version of RDF2Graph adapted to Wikidata easily available to Wikidata editors. This includes the general design and implementation of the tool (section 4.1), some specifics to make it run on the Wikimedia Toolforge platform (section 4.3), and a few utilities to make it more pleasant to use (section 4.2). The tool is online at `https://tools.wmflabs.org/wd-shex-infer/` and its source code may be found at `https://phabricator.wikimedia.org/source/tool-wd-shex-infer/`.

4.1. General Design and Implementation

The goal of the tool is to make the schema inference process available to other Wikidata editors, which is necessary because RDF2Graph, its ShEx exporter, and the Makefile-based process around them require several different programs and programming runtimes to be available, which it would be unreasonable to expect every interested user to install (not to mention that the Makefile assumes a Unix-like environment, whereas the potential users are likely to use Windows). Installing such a program on Wikimedia Toolforge and making it available to users via a web interface is a common approach in the Wikimedia movement.

In this Wikimedia-hosted tool, users enter a SPARQL query selecting the set of exemplary Items from which to infer a schema, and the tool downloads the associated data, run RDF2Graph and the ShEx exporter, and makes the ShExC output available to the user. The simplest process for that would be to directly return the ShExC code in the server's response to the user's HTTP request, but that is not possible: even simple inference jobs from a single Item take at least several minutes, and more complicated jobs over several dozens of Items can take multiple hours – the connection to the user would time out long before that, and the user might also become impatient and simply close their browser window or tab.

Instead, the inference process runs in the background, and after submitting the SPARQL query and starting the process, the user is redirected to a page describing the currently running job. They can periodically reload the page, and each time the page is loaded, the tool checks if the background process is still running. If the background process has finished in the meantime, the tool collects the output, cleans up the temporary files left behind by the process, and finally makes the output (primarily the ShExC code, but also debugging outputs) available to the user.

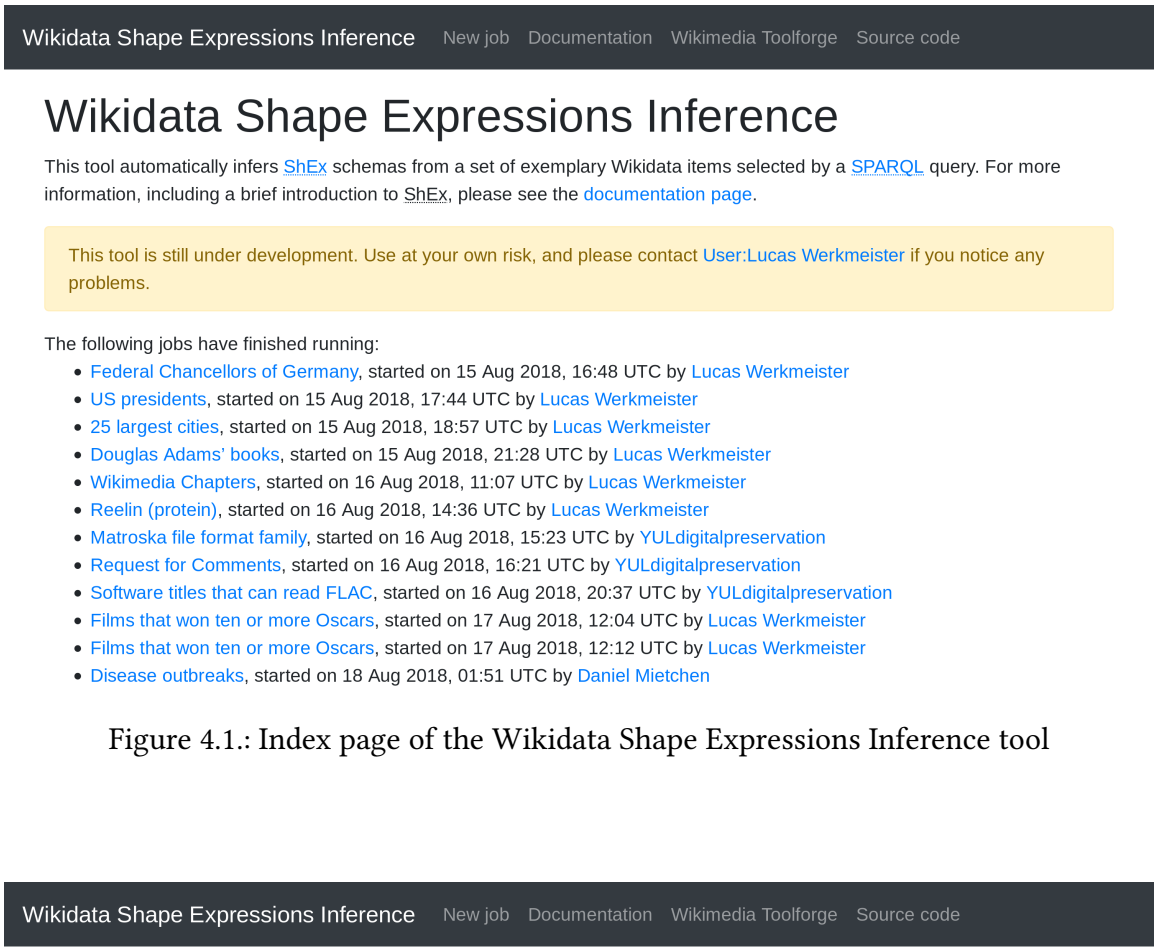


Figure 4.1.: Index page of the Wikidata Shape Expressions Inference tool

Figure 4.2.: Detail page for job #37

Since this already more or less requires storing inputs and outputs persistently, a natural extension of this scheme is to make this job page available to others, so that not only the original submitter but also other users can inspect a running or completed job. (The input data is entirely public data from Wikidata, so there is no reason to restrict the visibility of the outputs.) This facilitates easy collaboration on schemas between multiple users and allows even casual visitors to see how the tool operates, and what kinds of results it produces, without having to start their own inference process. To make each result easier to comprehend for others, users can also submit a title, description, and/or URL for additional information along with their SPARQL query. Figure 4.2 shows the page for one finished job, with its title, description, and URL (the "more information" link), and fig. 4.1 shows the index page of the tool, listing all the finished jobs that were submitted. (At the time of taking the screenshot, no jobs were currently running, otherwise they would also be listed. The screenshot also truncates the list of finished jobs – the full list is several times longer.)

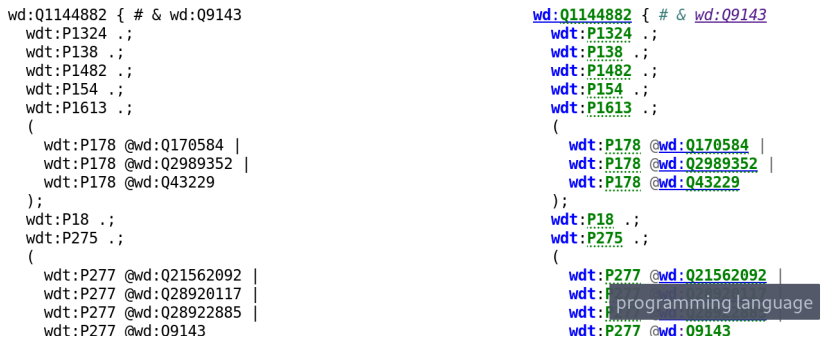
As the inference process consumes a lot of resources, it does not run directly on the same system as the web server providing the tool's front-end. Instead, it is submitted as a job for the Sun Grid Engine system also offered by the Wikimedia Toolforge environment, where it will be run on some execution host with sufficient free resources. To ensure that the system's resources, shared with all other Wikimedia Toolforge users, are not exhausted by this one tool, the tool only allows at most two jobs to run in parallel, and prevents users from submitting any more jobs if too many jobs are currently running, instead asking them to try again later. Job submission is also restricted to users with a valid Wikidata account, authenticated via OAuth. The identity of the submitting user is another attribute of a job, along with its title, description, URL, and SPARQL query, and can be seen by other users who look at the pending or finished jobs of the tool.

Details of each job, such as its title, submitting user, and submission time, are stored in a tool-specific SQL database on the Wikimedia Toolforge database servers. However, the outputs of a job (ShExC file, standard output of the process, standard error of the process), as well as to a lesser degree the SPARQL query forming its input, are too large to store in a database. Instead, they are stored directly on the file system.

4.2. Schema Reading Utilities

To make the ShEx output more readable, support for the ShExC syntax was added to the popular Pygments syntax highlighter (Pygments pull request #770). If the ShEx code for a job is requested by a browser or some other kind of client which declares that it can accept HTML (using HTTP content negotiation), the tool applies the Pygments syntax highlighter to the code on-the-fly and sends a complete HTML document containing the prettified code and associated CSS rules. Clients which do not accept HTML documents receive the plain ShExC code instead.

The HTML document that is sent to browsers also includes a client-side script, which searches for Wikidata Item IDs in the ShExC code, fetches the Labels of all mentioned Item IDs from Wikidata, and adds them to the document in `<abbr>` abbreviation elements. It also hyperlinks each reference of a shape to its definition, and the definition to the



(a) Excerpt of an inferred schema for Linux distributions

(b) The same schema with syntax highlighting and the script applied – the cursor is over the P277 Property ID

Figure 4.3.: Screenshots showing the effects of syntax highlighting and the client-side script

corresponding Item on Wikidata. This makes it much more convenient to read the schema in the browser: the user merely has to hover their mouse over a class to see its Label instead of the Item ID; clicking on it brings the user to the shape for that class; and clicking on the ID there brings the user to the Wikidata page for the class, where they can further explore the related data.

A comparison of the effects of these two features can be seen in fig. 4.3.

4.3. Wikimedia Toolforge Support

Since Wikimedia Toolforge runs on a rather old Linux distribution (Ubuntu 14.04 "Trusty Tahr", originally released April 2014), the versions of various software used by RDF2Graph and the rest of the inference process were too outdated to use out-of-the-box. The response to this was twofold: to cope with older software versions where feasible, and to install newer software versions where not.

RDF2Graph required Java 8, but only Java 7 is installed on Wikimedia Toolforge. Fortunately, RDF2Graph only used Java 8 for syntactic constructs ("lambda" syntax) and those only sparingly, so it was not too much work to make it compatible with Java 7 again. Similarly, Apache Jena and Fuseki have required Java 8 for some time already, but fortunately RDF2Graph is compatible with their older versions which support Java 7, and which could therefore be installed.

On the other hand, there is no obvious way to make the `jq`-based part of the data download step work with `jq` version 1.4, which has no general string replacement capabilities. Therefore, `jq` version 1.5 was installed locally into the tool's home directory – as was the latest version of Node.js, which was similarly outdated. The Pygments syntax highlighter mentioned in section 4.2 was also installed locally, since ShExC syntax support is not yet available in any released version.

Full installation instructions may be found in the `README.md` file in the tool's source code repository.

5. Evaluation

The evaluation of this thesis mainly focuses on the quality and usefulness of the resulting schemas. Unfortunately, without being able to reliably validate data sets against schemas – for example, to verify whether an Item for an author matches a schema inferred from fifty other authors – it is not possible to objectively assess the quality of the inferred schemas. However, inspecting the schemas manually can still give insights on their quality (section 5.1), as well as the quality of the underlying data, and it is also possible to extract smaller subsets from the full schemas, which can then be used for validation (section 5.2). In these ways, the schemas can still be useful even though they cannot be used for validation directly. Additionally, the runtime of the inference process is examined in section 5.3.

5.1. Schema Quality

While the schema quality cannot be assessed objectively without reliable validation, it is possible to simply look at the schemas and see if they "make sense" on their own. Generally, the inferred schemas are much too large to read the entire schema, but one can select the shapes for individual classes (randomly or by searching for the Item IDs of specific classes) and check their type links. For example, listing 7 shows the shape for the class "human" (Q5) from the schema inferred from 50 members of the 13th Riigikogu (the Estonian parliament). In textual form, it means that a human should have the following Statements:

- Member of any number of political parties, where the values are political parties.
- Any number of occupations, where the values are positions. This likely reflects the fact that the input data only contained politicians – other schemas often include other possible classes for this property, such as "occupation" or "profession".
- Any number of spoken, written or signed languages, where the values are languages.
- Optionally, a name in the subject's native language.
- Any number of awards received, where the values are classes of awards. The target class, "class of award", is an artifact of the not completely consistent modeling of awards in Wikidata: there is sometimes confusion about the relationship between different "levels" of awards, such as "award", "peace prize", "Nobel Prize", "Nobel Peace Prize", "2018 Nobel Peace Prize", and whether they should be instances or subclasses of each other.
- Optionally, a place of birth, where the value is a human settlement.

```
wd:Q5 { # & wd:Q215627
  wdt:P102 @wd:Q7278*;
  wdt:P106 @wd:Q4164871*;
  wdt:P1412 @wd:Q34770*;
  wdt:P1559 rdf:langString?;
  wdt:P166 @wd:Q38033430*;
  wdt:P19 @wd:Q486972?;
  (
    wdt:P21 @wd:Q4369513? |
    wdt:P21 @wd:Q48277?
  );
  wdt:P27 @wd:Q1048835?;
  wdt:P373 xsd:string?;
  wdt:P39 @wd:Q4164871*;
  wdt:P569 xsd:dateTime?;
  wdt:P69 @wd:Q2385804*;
  wdt:P734 @wd:Q101352?;
  wdt:P735 @wd:Q202444?;
  wdt:P937 @wd:Q486972*
}
```

Listing 7: Excerpt of a schema inferred from 50 members of the 13th Riigikogu

- Optionally, a sex or gender, where the value is a sex of humans or a gender. The two target classes are the result of the two most common gender Items in Wikidata, “male” (Q6581097) and “female” (Q6581072), each having several “instance of” (P31) statements.
- Optionally, a country of citizenship, where the value is a political territorial entity.
- Optionally, a Wikimedia Commons category.
- Any number of positions held by the subject, where the values are positions.
- Optionally, a date of birth.
- Any number of places where the subject was educated, where the values are educational institutions.
- Optionally, a family name, where the value is a family name.
- Optionally, a given name, where the value is a given name.
- Any number of work locations, where the values are human settlements.

Aside from some quirks of the input data, all of these seem reasonable to me. The most obvious missing predicates are date and place of death, which is again due to the input

data set: all the members of the current Estonian parliament are alive, and while one might expect to see dates of death on some Items they link to, the fact that currently only 33 parents of members of the 13th Riigikogu are known to Wikidata makes it seem plausible enough that there simply happened to be no dead humans in the full input data.

As the input data sets get larger, the schemas also tend to grow: in the number of shapes (classes), the number of predicates in each shape, and also in the number of possible classes for a predicate. For example, before simplification, the "human" (Q5) shape from the schema inferred from the set of US presidents lists *nine* possible classes for someone's "given name" (P735):

- A generic "given name" (Q202444), such as "Boylston" (Q18396847) in Thomas Boylston Adams, the third son of President John Adams. (The more common given names generally have a more specific class, usually one of the next three listed here.)
- A "male given name" (Q12308941), such as "James" (Q677191) in President James A. Garfield.
- A "female given name" (Q11879590), such as "Ida" (Q644599) in Ida Stover Eisenhower, mother of President Eisenhower.
- A "unisex given name" (Q3409032), such as "Anne" (Q564684) in Nancy Reagan (born Anne Frances Robbins), wife of President Reagan. ("Anne" is a female given name in English, but sometimes used as a male given name in the Netherlands or France, and therefore classified as both female and unisex in Wikidata, depending on language.)
- A "compound given name" (Q1243157), such as "George Washington" (Q16275947) in George Washington Adams, the first son of President John Quincy Adams.
- A "hypocorism" (Q1130279) (a diminutive form of a name), such as "Ron" (Q2165388) in Ron Reagan, son of President Reagan.
- A "diminutive" (Q108709), such as "Jimmy" (Q4166211) in President Jimmy Carter. Arguably, that Item should be an instance of the aforementioned hypocorism class instead of the more generic diminutive class, which also includes non-names like "auntie".
- "Initials instead of given names" (Q19803443), such as "S." (Q19803518) in President Harry S. Truman (not an abbreviation).
- A "family name" (Q101352), such as "Simpson" (Q2800825) in President Ulysses S. Grant (the "S." is sometimes believed to be short for "Simpson", his mother's maiden name). This is clearly an error: even if the "S." does stand for "Simpson", which Grant denied, the correct Item to use would be the given name Q50876620 (same label), not the family name Q2800825.

If the two problems pointed out above were to be fixed, all the remaining classes could be merged into the generic "given name" (Q202444) class, as in the earlier schema. However, due to the presence of "family name" (Q101352), the best common superclass is instead

"anthroponym" (Q10856962), the common superclass of given and last names, and the "diminutive" (Q108709) class remains unmerged, as it is not a subclass of any kind of name.

One can continue to pick apart the generated schemas in this manner for hours, and I have done so while working on this thesis in order to find problems and possible improvements in the inference process. Three insights emerge:

1. The simplification step is vital for readable schemas. The schemas are no less correct without simplification as far as the input data set is concerned, but if the numerous subclasses in Wikidata's hierarchy are never merged, the schemas become exceedingly tedious to read for humans, and they will also often match other data sets less well if the other data sets involve subclasses that were not encountered in the original input data.

   This can be seen in some of the older jobs of the Wikidata Shape Expressions Inference tool: if there is a `StackOverflowError` in the standard error of the inference process, it means that RDF2Graph crashed during the simplification step and the ShEx export ran on the original, unsimplified graph. (Later, the inference step was made more robust, which is why newer jobs do not have this problem.)

2. Sometimes, biases in the schema are clearly visible, due to biases in the input data set. (In one particularly egregious case, a schema proclaimed that a given name must always be a *male* given name, since there had been no women with given names in the input data set.) One can presume that at other times biases in the schema are not as clearly visible, which does not mean that they are not present. To arrive at useful schemas, care must be taken when selecting the input data set, and the results must be viewed critically.

3. Sometimes, the schema reflects errors in the input data, as in the case above where a family name Item was used for a given name. This can often be traced back to confusion between several Items with similar labels. Such errors are usually visible in the resulting schema if one takes the time to read it, though they are not always obvious due to the simplification (as when the classes for given and family names were merged into anthroponyms above).

5.2. Manual Schema Extraction

One way to make use of the inferred schemas even though they cannot be used for validation directly is to extract a smaller subset of the schema manually, then validating Items against that. If desired, that schema can then be further altered and augmented as deemed necessary, making the original schema the basis of a manually curated one.

Generally, to extract parts of the schema, one starts with a basic shape for the input Items (e. g. "human" (Q5) if the schema was inferred from a set of humans), copies it into the reduced schema, and repeats this procedure for all shapes (classes) mentioned by that shape which had not been copied yet. Predicates which link to shapes that should not be included can be dropped when copying a shape: for example, the "country of citizenship" (P27) shape should be dropped if one is not interested in including shapes

for countries, states etc. in the reduced schema. Some predicates may also need to be moved between shapes, or the target classes may need to be adjusted, if the results of the automatic simplification are not satisfactory, e. g. if unrelated classes were merged into a very fundamental base class like "property" (Q937228) despite the changes in section 3.2.5.

Listing 8 shows two schemas that were manually extracted in this fashion from schemas inferred by the Wikidata Shape Expressions Inference tool. Listing 8a was extracted from a schema inferred from 100 scientific articles about the Zika virus, and contains shapes for the classes "disease" (Q12136) and "taxon" (Q16521): diseases may be caused (indirectly or immediately) by taxa, and taxa may effect diseases and have any number of parent taxa. Listing 8b was extracted from a schema inferred from the set of films which won three or more Academy Awards ("Oscars") and contains shapes for the classes "film" (Q11424) and "human" (Q5): films have human directors, editor, cast members, screenwriters, etc., and humans may have parents, any number of spouses and children, and dates of birth and death.

These schemas are simple enough that they can be validated using the "Simple Online Validator" at `https://rawgit.com/shexSpec/shex.js/wikidata/doc/shex-simple.html`, which automatically downloads all the required data from Wikidata. Validating 100 arbitrary diseases against the shape for "disease" (Q12136) from listing 8a mostly yields positive results, with only four violations: the Items "X-linked adrenoleukodystrophy" (Q366964), "Morquio Syndrome" (Q580285) and "Sanfilippo syndrome" (Q2200359) have more than one "NCI Thesaurus ID" (P1748) Statement, while the schema says a disease may only have zero or one such IDs, and the Item "nodding disease" (Q895930) fails validation because it has two "has cause" (P828) Statements, "autoimmune disease" (Q8084905) and "parasitic infectious diseases" (Q1601794), whereas the schema says a disease may only have zero or one causes. (The schema also says that those causes should be taxa, not other diseases, but the shape for taxa is sufficiently lax that these diseases match it.) Validating 100 arbitrary films against the shape for "film" (Q11424) from listing 8b also yields positive results with only a single violation – "Detective Conan" (Q185143) has two "logo image" (P154) Statements (French and Japanese) whereas the schema says it should only have one.

A second look at the schemas makes it clear why there are so few violations: the cardinalities for all predicates, without exception, are either ? ("zero or one") or $*$ ("any number"). A completely empty Item with no Statements at all will match all four shapes in listing 8. This is likely an artifact of the schema extraction procedure, which in this case was extremely selective of shapes to ensure that the schema would fit on one page of this document, and therefore only included very few shapes, each of which had had a high number of examples in the input data set: enough that, for any Property, there was an example Item missing Statements for that Property, forcing the lower boundary of the cardinality to be zero. Overall, the sample schemas investigated in appendix A.3 have 17 % to 34 % of non-optional predicates, so if one includes some more shapes during the extraction process, they are bound to include some required predicates soon. (Alternatively, the cardinality of some predicates could be manually raised during the extraction.)

Generally, these extracted schemas appear to be useful to find some mistakes, though their utility can be increased by extracting larger parts of the schemas and by further refining them according to one's personal experience with the data model. A useful side effect is that the kind of careful inspection of the schema which is necessary to extract

useful parts of it will also likely find some problems in the schema where they exist, pointing at errors in the original input data set, as demonstrated in section 5.1.

```
wd:Q12136 {
  wdt:P1478 @wd:Q16521?;
  wdt:P1748 xsd:string?;
  wdt:P2572 xsd:string?;
  wdt:P373 xsd:string?;
  wdt:P667 xsd:string?;
  wdt:P828 @wd:Q16521?
}

wd:Q16521 {
  wdt:P1542 @wd:Q12136?;
  wdt:P171 @wd:Q16521*;
  wdt:P225 xsd:string?;
  wdt:P373 xsd:string?;
  wdt:P935 xsd:string?
}
```

(a) Schema for diseases, based on job #37

```
wd:Q11424 {
  wdt:P1040 @wd:Q5*;
  wdt:P1431 @wd:Q5*;
  wdt:P154 .?;
  wdt:P161 @wd:Q5*;
  wdt:P162 @wd:Q5*;
  wdt:P175 @wd:Q5?;
  wdt:P1809 @wd:Q5*;
  wdt:P2130 xsd:decimal?;
  wdt:P2142 xsd:decimal?;
  wdt:P2515 @wd:Q5*;
  wdt:P2554 @wd:Q5*;
  wdt:P2769 xsd:decimal?;
  wdt:P3092 @wd:Q5*;
  wdt:P3174 @wd:Q5*;
  wdt:P3300 @wd:Q5?;
  wdt:P344 @wd:Q5*;
  wdt:P373 xsd:string?;
  wdt:P57 @wd:Q5*;
  wdt:P58 @wd:Q5*;
  wdt:P725 @wd:Q5*;
  wdt:P86 @wd:Q5*
}

wd:Q5 {
  wdt:P22 @wd:Q5?;
  wdt:P25 @wd:Q5?;
  wdt:P26 @wd:Q5*;
  wdt:P40 @wd:Q5*;
  wdt:P569 xsd:dateTime?;
  wdt:P570 xsd:dateTime?
}
```

(b) Schema for films, based on job #36

Listing 8: Two schemas manually extracted from automatically inferred ones

5.3. Duration of the Inference Process

While attempts to validate data against the inferred schema suffer due to the size of the input data and the schemas, no such problems appear to plague the inference process, which, while slow, has a more reliable runtime. To some degree, this was already apparent in the execution times of the various jobs that were run on the Wikidata Shape Expressions Inference tool: ranging from five or ten minutes to several hours, they roughly follow the size of the input data linearly.

However, the timing data from the Wikidata Shape Expressions Inference tool is not without its problems: as the tool was tested with different jobs, various problems were discovered and subsequently fixed (some of them described in more detail earlier, others too minor to be worth mentioning), so the runtimes available from the tool apply to a range of software versions, with several instances of the same job being repeated (to verify a software fix) with highly different runtimes. Therefore, a subset of the tool's jobs was selected and repeated locally, with a single software version, to get more reliable execution times. Appendix A.3 contains graphs of runtime over various factors, which are explained in more detail in the following paragraphs, each with three different functions fitted to the data: a simple linear function $a + bx$, a quadratic function $a + bx + cx^2$, and a power function $a + bx^c$. Each figure contains two subfigures, one for the full data and one with two outlier records removed – see the text in the appendix for details.

The most obvious possible relation is to directly compare the runtime to the number of Items selected by the user's query, as shown in fig. A.1. However, it is clear from the graphs that there is no direct relation between the number of Items and the runtime: in fact, four of the six resulting functions suggest that execution time shall become negative if one only supplies enough Items, which is clearly nonsensical. This is not surprising, because the amount of work that RDF2Graph has to do highly depends on the size of the Items, as well as the number of Items indirectly selected as the values of Statements on the original Items (and their size).

Instead, a much more sensible relation is the total amount of input data: the number of triples which Fuseki serves to RDF2Graph (that is, the number of lines in the N-Triples file). As can be seen in fig. A.2, this results in a fairly linear relation, especially if two outliers are removed, in which case the functions fit the data very well. However, with the outliers included the fit is much less satisfactory.

Since RDF2Graph heavily relies on type information, another possible factor for execution time is just the number of `wdt:P31` triples in the input, rather than the overall number of triples. (Recall that "instance of" (P31) is the Wikidata property linking an Item to its class.) The number of `wdt:P31` triples was counted using the GNU AWK snippet found in listing 9, and the result is shown in fig. A.3: the functions are more linear and fit the data better, both with and without outliers. However, with outliers included the fit is still not completely satisfactory.

Alternatively, instead of counting `wdt:P31` triples it is also possible to count the distinct number of classes in the input data. Classes were counted using the GNU AWK snippet found in listing 10, and the result is shown in fig. A.4: the functions are significantly less linear now (though this is not visible in the function equations shown in the graphs, due

to rounding), but they finally fit the data well without excluding the outliers: in fact, the graph with outliers has slightly better fits than the one without outliers.

Two conclusions are possible from this: the execution time could derive linearly from the number of input triples or `wdt:P31` triples, or it could derive nonlinearly from the number of input classes. The first conclusion has no satisfactory explanation for the outliers which need to be excluded to get good function fits, whereas the second conclusion requires no cherry-picking in the data but lacks an explanation for the nonlinear runtime. However, while more data may be necessary to decide which conclusion is more accurate for larger input data sets than investigated, most jobs are expected to use smaller input data sets, where the conclusions do not significantly disagree and a prediction of execution time is possible with reasonable confidence.

One might think that these relations are not very useful because they only predict the duration of the whole process partway through the process. However, all of the possible predictor variables – number of Items, number of triples, number of `wdt:P31` triples, number of distinct classes – can be determined after the data download step, which is both the very first step in the whole process and also does not take a long time: in the jobs listed in appendix A.3, the download never takes more than twenty seconds, and for most jobs it finishes within about five seconds. This means that it takes less than half a minute to be able to mostly predict the duration of the full job, which can be several hours. This suggests two possible future improvements for the Wikidata Shape Expressions Inference tool: to report to the user how long a job is expected to take, as soon as the download step has finished; and to reject jobs which resulted in too much data, and are expected to take far too long to be tolerable. Currently, the tool merely suggests to its users that their queries should not select more than about fifty Items, but does not implement any kind of hard limit beyond this suggestion.

```
BEGINFILE {
  count = 0;
}

$2 == "<http://www.wikidata.org/prop/direct/P31>" {
  count++;
}

ENDFILE {
  print FILENAME, count
}
```

Listing 9: GNU AWK script to count the number of `wdt:P31` triples in the input

```
BEGINFILE {
  count = 0;
  delete classes;
}

$2 == "<http://www.wikidata.org/prop/direct/P31>" {
  classes[$3]++
}

ENDFILE {
  for (class in classes)
    count++;
  print FILENAME, count;
}
```

Listing 10: GNU AWK script to count distinct classes in the input

6. Conclusion

The goal of this thesis was to investigate how ShEx schemas can be automatically inferred for Wikidata, and how useful the resulting schemas are. This was done using an updated and adapted version of the RDF2Graph software, which was made available to the Wikidata community through a new web-based tool.

In addition to the changes specific to Wikidata, many general improvements to RDF2-Graph were made over the course of this thesis, making it easier to use and more robust on any graph. All these changes are available under the same free software license as the original RDF2Graph, and I hope that some of them will be included in the original source code repository in the future.

When attempting to validate other Items against the inferred schemas, an unexpected problem arose: none of the existing ShEx validators were able to reliably perform the validation without crashing. Several strategies were attempted to remediate this, both in the schema extraction and in the validators, but this was ultimately unsuccessful.

However, this does not mean the schemas are not useful. Sometimes, problems in the input data can manifest themselves in the form of unusual predicates or target classes in a schema, which an attentive reader can notice and trace back to the problematic Items in the input. And the full, automatically inferred schemas can also form the basis for shorter schemas manually extracted from the longer ones, which can either be validated directly (now without problems from the validators) or be further refined by users familiar with the data model, making the automatically inferred schemas a useful basis for manually curated schemas.

There are plenty of options of further improvements on this work. Section 5.3 already mentioned how the Wikidata Shape Expressions Inference tool could be improved to notify the user of the estimated total runtime of a job once the data download step is complete and to reject jobs which are expected to take too long. Many other improvements could be made to the user interface as well, such as exposing some more configuration options to users of the tool, e. g. the thresholds for schema reduction mentioned in section 3.4.

A significant improvement over the current state would be to make RDF2Graph work on full Statement nodes instead of just the "truthy" Statements. A full explanation of full Statement nodes is beyond the scope of this thesis, but in brief, they offer much more information about Wikidata Statements in exchange for a slightly more complicated data format. Using full Statement nodes would allow RDF2Graph to take not just the Statement's values but also their Qualifiers and References into account. However, this would require major changes to the way RDF2Graph looks at the input graph, because the subject and object of a Statement are no longer linked via a single triple in the full Statement nodes.

Clearly, it is not a satisfactory final state that the inferred schemas cannot be directly used for validation. It may be possible to find optimizations in the validators and/or useful

criteria for reducing the inferred schemas which would enable direct validation against the schemas without human intervention to manually extract their most useful parts. There is also some room for improvement in the simplification step regardless of the ability to validate against the inferred schemas: sometimes, despite the changes in section 3.2.5, it still merges mostly-unrelated classes into very abstract base classes, so the criteria on when to merge or not merge classes could use some improvements. It may also be useful to apply some of the thresholds for schema reduction (section 3.4) before the simplification step starts, so that, for instance, a type link of "family name" (Q101352) for the "given name" (P735) predicate is filtered out before it is merged with "given name" (Q202444) into "anthroponym" (Q10856962) in simplification.

The detailed timings for the jobs listed in appendix A.3 show that the ShEx export step usually takes up about half of the total wall-clock time, and often significantly more than half of the CPU time. This thesis does not investigate this step more closely except for its very last part (see section 3.1), but it seems unlikely that this is necessary: it may be possible to change or rewrite the rest of the ShEx exporter to be much more efficient.

Overall, this thesis results in a promising new tool for the Wikidata community, which will hopefully prove useful to several WikiProjects in the future. Further work is anticipated to make the inferred schemas more useful yet by reducing them to their most important parts and making them suitable for automatic validation. The improvements to RDF2Graph that were implemented along the way will hopefully also benefit other RDF2Graph users once they are merged into the upstream repository.

Bibliography

[1] Thomas Baker and Eric Prud'hommeaux. *Shape Expressions (ShEx) Primer*. W3C Draft Community Group Report. W3C, July 2017. URL: http://shex.io/shex-primer-20170713/.

[2] Iovka Boneva, Jose G Labra Gayo, and Eric G Prud'hommeaux. "Semantics and Validation of Shapes Schemas for RDF". In: *ISWC2017 - 16th International semantic web conference*. Vienna, Austria, Oct. 2017. URL: https://hal.archives-ouvertes.fr/hal-01590350.

[3] Jesse CJ van Dam et al. "RDF2Graph a tool to recover, understand and validate the ontology of an RDF resource". In: *Journal of Biomedical Semantics* 6.1 (Oct. 2015), p. 39. ISSN: 2041-1480. DOI: 10.1186/s13326-015-0038-9. URL: https://doi.org/10.1186/s13326-015-0038-9.

[4] Ramanathan Guha and Dan Brickley. *RDF Schema 1.1*. W3C Recommendation. W3C, Feb. 2014. URL: http://www.w3.org/TR/2014/REC-rdf-schema-20140225/.

[5] Lucie-Aimée Kaffee et al. "A Glimpse into Babel: An Analysis of Multilinguality in Wikidata". In: *Proceedings of the 13th International Symposium on Open Collaboration*. OpenSym '17. Galway, Ireland: ACM, 2017, 14:1–14:5. ISBN: 978-1-4503-5187-4. DOI: 10.1145/3125433.3125465. URL: http://doi.acm.org/10.1145/3125433.3125465.

[6] Markus Lanthaler, David Wood, and Richard Cyganiak. *RDF 1.1 Concepts and Abstract Syntax*. W3C Recommendation. W3C, Feb. 2014. URL: http://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/.

[7] Andrew Lih and Robert Fernandez. *Wikidata, a rapidly growing global hub, turns five*. Oct. 2017. URL: https://blog.wikimedia.org/2017/10/30/wikidata-fifth-birthday/ (visited on 10/11/2018).

[8] Ashok Malhotra and Paul V. Biron. *XML Schema Part 2: Datatypes Second Edition*. W3C Recommendation. http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/. W3C, Oct. 2004.

[9] Patrick McKenzie. *Falsehoods Programmers Believe About Names*. June 2010. URL: https://www.kalzumeus.com/2010/06/17/falsehoods-programmers-believe-about-names/ (visited on 10/09/2018).

[10] Lydia Pintscher. "WikidataCon 2017 – State of the Project". In: WikidataCon 2017. 2017. URL: https://commons.wikimedia.org/wiki/File:WikidataCon_2017-_State_of_the_Project.pdf.

[11] 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*. SEM '14. Leipzig, Germany: ACM, 2014, pp. 32–40. ISBN: 978-1-4503-2927-9. DOI: 10.1145/2660517.2660523. URL: http://doi.acm.org/10.1145/2660517.2660523.

[12] schema.org. *Extending Schemas*. 2011. URL: https://schema.org/docs/old_extension.html (visited on 09/26/2018).

[13] Guus Schreiber and Yves Raimond. *RDF 1.1 Primer*. W3C Note. W3C, June 2014. URL: http://www.w3.org/TR/2014/NOTE-rdf11-primer-20140624/.

[14] Andy Seaborne and Gavin Carothers. *RDF 1.1 N-Triples*. W3C Recommendation. http://www.w3.org/TR/2014/REC-n-triples-20140225/. W3C, Feb. 2014.

[15] *SPARQL 1.1 Overview*. W3C Recommendation. W3C, Mar. 2013. URL: http://www.w3.org/TR/2013/REC-sparql11-overview-20130321/.

[16] Denny Vrandečić. *Restricting the World*. Feb. 2013. URL: https://blog.wikimedia.de/2013/02/22/restricting-the-world/ (visited on 09/05/2018).

[17] Denny Vrandečić and Markus Krötzsch. "Wikidata: A Free Collaborative Knowledgebase". In: *Commun. ACM* 57.10 (Sept. 2014), pp. 78–85. ISSN: 0001-0782. DOI: 10.1145/2629489. URL: http://doi.acm.org/10.1145/2629489.

Glossary

Alias additional term by which an Item or Property should be found. 6, 20, 24, 49

CSS Cascading Style Sheets. 29

Description a short clarifying or disambiguating text for an Item or Property. 6, 20, 24, 49

focus node the RDF resource currently being matched against a ShEx shape. 10, 11, 24, 47

HTML Hypertext Markup Language. 17, 20, 29

HTTP Hypertext Transfer Protocol. 27, 29

IMDb Internet Movie Database. 20

IRI Internationalized Resource Identifier, Unicode-aware generalization of URIs. 3, 4, 10, 17, 46

Item representation of a thing or concept in Wikidata. i, vii, xi, 1, 6–9, 18, 20–22, 24–27, 30–35, 38, 39, 41, 45–47, 49–53, 55

Item ID unique, language-agnostic identifier of an Item, a consecutive number prefixed with the letter "Q". 6, 29–31, 47

Java programming language for portable applications. 16, 24, 30, 47

JavaScript programming language, originally for the web, can be run outside the browser using Node.js. 16, 17, 24, 45–47

job one inference process in the Wikidata Shape Expressions Inference tool, launched by a Wikidata user by providing a SPARQL query selecting exemplary Items. vii, xi, 27–29, 34, 37–39, 41, 50–53

JSON JavaScript Object Notation. 12, 18, 45

JSON-LD JSON for Linked Data, a format to represent RDF graphs. 12, 17

Label the primary term for an Item or Property in a language. 6, 7, 20, 24, 29, 30, 49

Linked Data data that is interlinked, using URIs to identify things and returning data about those things in standard formats when a URI is dereferenced. 3, 7, 45

LoC Library of Congress. 20

lower camel case combining multiple words into one by capitalizing the first letter of each word except the first word; also known as camelCase, lowerCamelCase, mixedCase. 3

N-Triples simple, line-based syntax for RDF graphs. 4, 18, 38

Node.js platform to run JavaScript programs outside the browser. 24, 30, 45

object the third element of an RDF triple, the "<human>" in "<Alan Turing> <is a> <human>". 3, 4, 10, 41, 47

predicate the second element of an RDF triple, the "<is a>" in "<Alan Turing> <is a> <human>". 3, 4, 10–12, 18, 20, 24, 25, 32–35, 41, 42, 47

prefix abbreviation of a common part of a resource IRI. 3, 4, 7, 17, 47

Property a possible feature, characteristic or quality of an Item. 6, 7, 20, 35, 45–47, 49

Property ID unique, language-agnostic identifier of a Property, a consecutive number prefixed with the letter "P". 6, 30

Qualifier an additional Property-value pair further refining a Statement. 6, 7, 41, 49

RDF Resource Description Framework. vii, ix, 3–5, 7, 10–12, 18, 20, 21, 45–47

RDF2Graph tool to automatically determine a schema from an RDF graph. i, iii, v, 1, 11–13, 15, 17–27, 30, 34, 38, 41, 42, 46, 47, 53, 54

RDFS RDF Schema. 3, 4

RDFSimpleCon library used by RDF2Graph. 17

Reference a collection of Property-value pairs recording a source for a Statement. 6, 7, 41, 49

resource representation of a thing or concept in RDF, identified by an IRI. 3, 4, 10, 45–47

RfC Request for Comments. 53

schema formal description of the structure of a linked data set; in ShEx, a collection of shapes. i, ix, xi, 1, 7, 10–12, 17, 18, 20, 21, 24–27, 29–38, 41, 42, 46, 47, 49–52

shape element of a ShEx schema, describing the structure of one node category. xi, 10–12, 17, 18, 20, 24–26, 29–31, 33–35, 45–47, 50–52

ShEx Shape Expressions. i, iii, 1, 7, 10–12, 17–20, 22, 24, 25, 27, 29, 34, 41, 42, 45–47, 49, 54

shex-java ShEx implementation in Java. 24

shex.js ShEx implementation in JavaScript. 24, 26

ShExC ShEx Compact Syntax. 10, 12–17, 27, 29, 30

simplification optional RDF2Graph step to simplify (usually, shorten) an inferred ShEx schema by merging references to related classes. 12–16, 18, 20–23, 25, 33–35, 42, 54

Sitelink a link from an Item to a page in a Wikimedia project about the thing or concept the Item represents. 6, 49, 53

SPARQL SPARQL Protocol and RDF Query Language. 3, 7, 12, 18, 21, 27, 29, 45, 47

Statement a unit of information in Wikidata, consisting of a subject Item, a predicate Property, and a value (Item ID, quantity, point in time, etc.). 6, 7, 18, 20, 25, 31, 35, 38, 41, 46, 49, 53

subject the first element of an RDF triple, the "<Alan Turing>" in "<Alan Turing> <is a> <human>". 3, 4, 10, 11, 41, 47

triple a unit of information in RDF, consisting of a subject resource, a predicate resource, and an object value (resource or literal). vii, 3, 4, 7, 10, 11, 25, 38, 39, 41, 46, 47, 56, 57

triple constraint element of a ShEx shape, restricting triples with the focus node as the subject and a certain predicate. 10, 11, 47

type link one possible type (datatype or shape) for a predicate in a shape; this term is only used by RDF2Graph. 12, 23–25, 31, 42

upper camel case combining multiple words into one by capitalizing the first letter of each word, including the first word; also known as CamelCase, UpperCamelCase, PascalCase. 3

URI Uniform Resource Identifier, usually a URL. 45

URL Uniform Resource Locator. 29, 47

value constraint element of a ShEx triple constraint, restricting the object (value) of a triple to match a certain datatype or other shape. 10, 12

VIAF Virtual International Authority File. 20

vocabulary set of resources useful in combination, often sharing a common prefix. 3, 4

W3C World Wide Web Consortium. 3, 10

WDQS Wikidata Query Service. 7, 18, 19, 21, 54

Wikidata free knowledge base in the Wikimedia movement. i, iii, 1, 6, 7, 17, 18, 20–22, 25–27, 29–35, 38, 41, 42, 45, 47–49

Wikidata Shape Expressions Inference tool web-based tool to make the inference process available to the Wikidata community. vii, 1, 27, 28, 34, 35, 38, 39, 41, 45, 53

Wikimedia free knowledge movement and community. i, iii, 1, 6, 16, 27, 47, 48, 53

Wikimedia Commons free media repository in the Wikimedia movement. 6, 7, 32

Wikimedia Foundation non-profit charitable organization in the Wikimedia movement, hosting Wikipedia and other sites. 16, 48, 53

Wikimedia Toolforge hosting environment provided by the Wikimedia Foundation. v, 16, 27, 29, 30

Wikipedia free encyclopedia in the Wikimedia movement. 6, 48, 53

Wikiquote free quotation collection in the Wikimedia movement. 6

XSD XML Schema Definition. 4

A. Appendix

A.1. A Note on Orthography

Throughout this thesis, the words “Item”, “Property”, “Label”, “Description”, “Alias”, “Sitelink”, “Statement”, “Qualifier” and “Reference” are capitalized as proper nouns when referring to elements of the Wikidata data model, following a common (though not universal) convention in the Wikidata community, and the word “schema” is pluralized into “schemas” rather than “schemata” for consistency with other ShEx texts [1, 11].

Both decimal and binary unit prefixes are used in this thesis, following ISO/IEC 80000: GB means 10^9 bytes and GiB means 2^{30} bytes.

A.2. Results of Validation With Depth Limit

The following tables show the results when attempting various validations against inferred schemas with different depth limits. The meaning of the result column is as follows:

solved validation completed successfully (found a solution)

fail validation completed unsuccessfully (reported violations)

limit the result is directly “depth limit reached”

core the process crashed and dumped core (out-of-memory error)

abort the process was manually killed after a long time with no apparent progress

limit	real time	CPU time	result
-	2 m 50 s	8 m 51 s	core
1	0 m 0 s	0 m 0 s	limit
2	4 m 41 s	17 m 22 s	core
3	3 m 57 s	13 m 31 s	core
4	2 m 51 s	9 m 1 s	core
5	2 m 48 s	8 m 54 s	core
6	2 m 50 s	8 m 58 s	core
7	2 m 48 s	8 m 41 s	core
8	2 m 47 s	8 m 44 s	core
9	2 m 48 s	8 m 48 s	core
10	2 m 48 s	8 m 53 s	core

Table A.1.: Results when validating the Item “Titanic” (Q44578) against the shape for the class “film” (Q11424) from a schema inferred from the set of films that won ten or more Oscars (job #29)

limit	real time	CPU time	result
-	0 m 1 s	0 m 3 s	fail
1	0 m 0 s	0 m 0 s	limit
2	0 m 16 s	0 m 21 s	core
3	0 m 1 s	0 m 2 s	fail
4	0 m 1 s	0 m 3 s	fail
5	0 m 1 s	0 m 3 s	fail
6	0 m 1 s	0 m 3 s	fail
7	0 m 1 s	0 m 3 s	fail
8	0 m 1 s	0 m 2 s	fail
10	0 m 1 s	0 m 3 s	fail

Table A.2.: Results when validating the Item “Douglas Adams” (Q42) against the shape for the class “human” (Q5) from a schema inferred from the set of films that won ten or more Oscars (job #29)

limit	real time	CPU time	result
-	0 m 17 s	0 m 7 s	fail
1	0 m 0 s	0 m 0 s	limit
2	0 m 0 s	0 m 1 s	solved
3	0 m 18 s	0 m 7 s	fail
4	0 m 19 s	0 m 8 s	fail
5	0 m 17 s	0 m 7 s	fail
6	0 m 18 s	0 m 8 s	fail
7	0 m 18 s	0 m 8 s	fail
8	0 m 18 s	0 m 8 s	fail
9	0 m 18 s	0 m 7 s	fail
10	0 m 18 s	0 m 7 s	fail

Table A.3.: Results when validating the Item “Douglas Adams” (Q42) against the shape for the class “human” (Q5) from a schema inferred from the members of the 13th Riigikogu (the Estonian parliament; job #30)

limit	real time	CPU time	result
-	2 m 50 s	8 m 51 s	core
1	0 m 0 s	0 m 0 s	limit
2	4 m 41 s	17 m 22 s	core
3	3 m 57 s	13 m 31 s	core
4	2 m 51 s	9 m 1 s	core
5	2 m 48 s	8 m 54 s	core
6	2 m 50 s	8 m 58 s	core
7	2 m 48 s	8 m 41 s	core
8	2 m 47 s	8 m 44 s	core
9	2 m 48 s	8 m 48 s	core
10	2 m 48 s	8 m 53 s	core

Table A.4.: Results when validating the Item “Mailis Reps” (Q449851) against the shape for the class “human” (Q5) from a schema inferred from the members of the 13th Riigikogu (job #30)

limit	real time	CPU time	result
-	1 m 25 s	4 m 14 s	core
1	0 m 1 s	0 m 1 s	limit
2	0 m 1 s	0 m 1 s	fail
3	0 m 55 s	2 m 6 s	core
4	7 m 34 s	13 m 54 s	core
5	0 m 10 s	0 m 27 s	core
6	0 m 30 s	1 m 5 s	core
7	0 m 51 s	1 m 59 s	core
8	5 m 29 s	12 m 54 s	core
9	0 m 9 s	0 m 19 s	core
10	86 m 39 s	153 m 35 s	abort

Table A.5.: Results when validating the Item “United States of America” (Q30) against the shape for the class “sovereign state” (Q3624078) from a schema inferred from a set of Items for bus stops (job #15)

A.3. Job Execution Times

The following charts show the total execution time of the inference process (in seconds) for various input queries collected from the Wikidata Shape Expressions Inference tool:

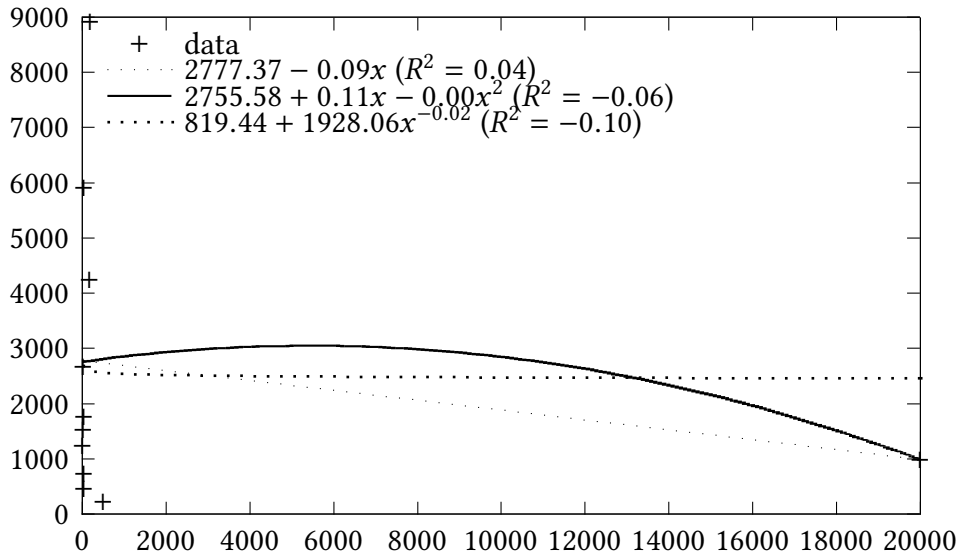
- Members of the 30th Riigikogu (the Estonian parliament), limited to 50 Items. Originally job #30.
- The Federal Chancellors of Germany. Originally job #1.
- Films that won three or more Academy Awards ("Oscars"). Originally job #32, later repeated as job #34, job #35, job #36. (Some earlier jobs used a similar query with a limit of ten or five Oscars.)
- The human Items with most Sitelinks to other Wikimedia projects (including but not limited to Wikipedia articles in different language editions), limited to 50 Items. Originally job #24, later repeated as job #26, job #27.

  This is one of the two outliers removed for the first version of each chart, since it has an unusually high execution time compared to the amount of input data. My best explanation for that is that this is a less "coherent" set of Items: since the selection criterion (the number of sitelinks) is not directly related to the data, this includes politicians, authors, religious figures, scientists, philosophers, celebrities, composers, and other kinds of people (though none of these are *classes* in the sense relevant to RDF2Graph).
- The largest cities in the world by population, limited to 25 Items. Originally job #3.
- Mammal taxa (including species but also genera, families etc.). Limited to 20 000 Items, a limit that was experimentally determined to produce a high but not exorbitant number of triples. Originally job #33.

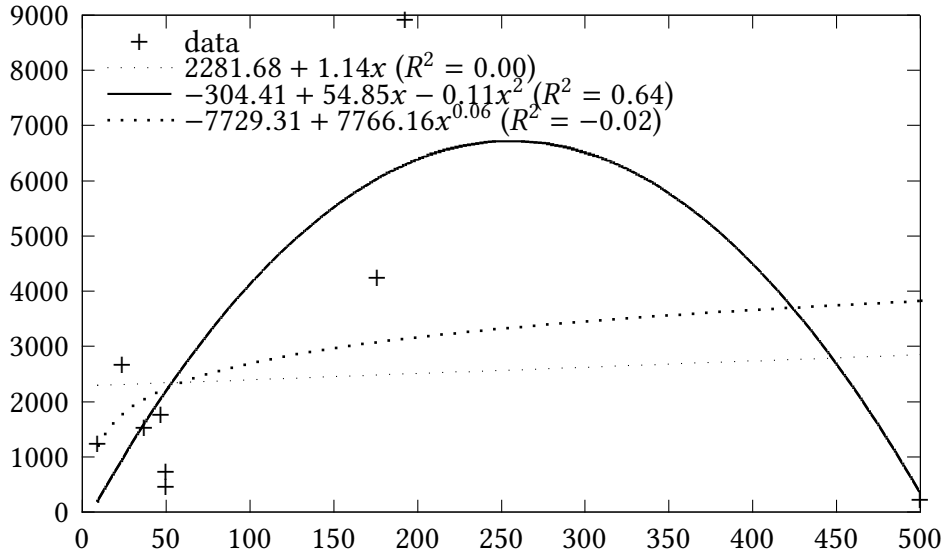
  This is the second of the two outliers removed for the first version of each chart, due to the absurd number of input Items: compare especially the X-axes of figs. A.1a and A.1b.
- Programming languages with the highest number of Statements, limited to 50 Items. Originally job #17, later repeated as job #18.
- Recently edited Items for Requests for Comments (RfCs), limited to 500 Items. Originally job #21, based on the earlier job #20 (equivalent but with a lower limit), which in turn was inspired by job #8.
- The member states of the United Nations. Originally job #23.
- The presidents of the United States of America. Originally job #2.
- The chapter organizations of the Wikimedia Foundation. Originally job #5.

For each input query, the different steps of the inference process (data download, initial RDF2Graph run, RDF2Graph simplification, ShEx export) were timed individually, recording the real ("wall-clock") time, user CPU time and system CPU time separately. Only the sum of the real time was used for the charts below, but the full data is available in the source code repository for this thesis. All the steps were performed on a system with an Intel® Core™ i7-4771 CPU (3.50 GHz clock rate), ca. 25.2 GB (23.5 GiB) of memory, and a network connection allowing download rates of up to 23.6 MiB/s from WDQS, with no significant other load of any kind at the time of measurement.

In addition to the graphs shown here, the full data also shows that the simplification step of the process never takes more than a minute, which means that while the changes in section 3.3 potentially slowed this step down to some degree, it does not make a big difference for the total execution time, which is mostly dominated by other steps (RDF2Graph without simplification and ShEx export) anyways.

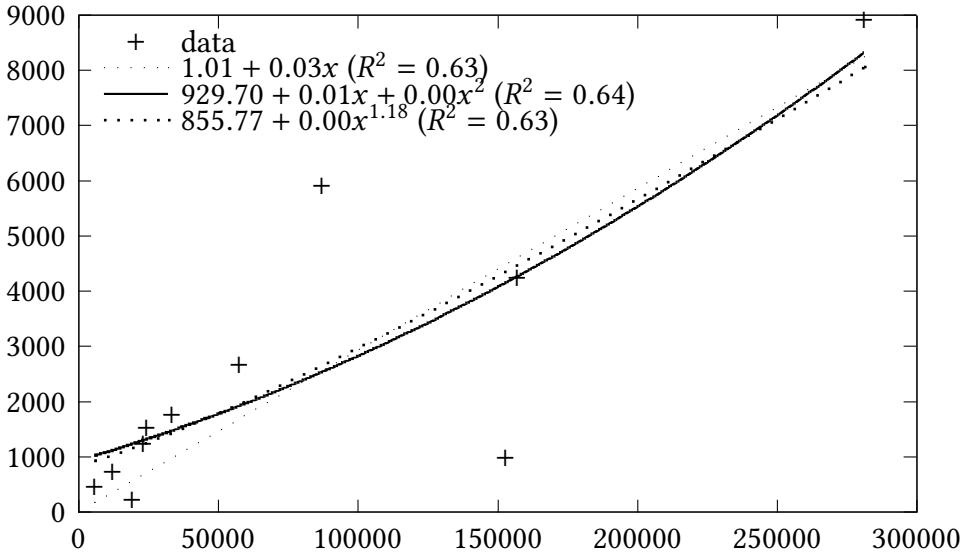


(a) Job execution time over number of Items, with outliers

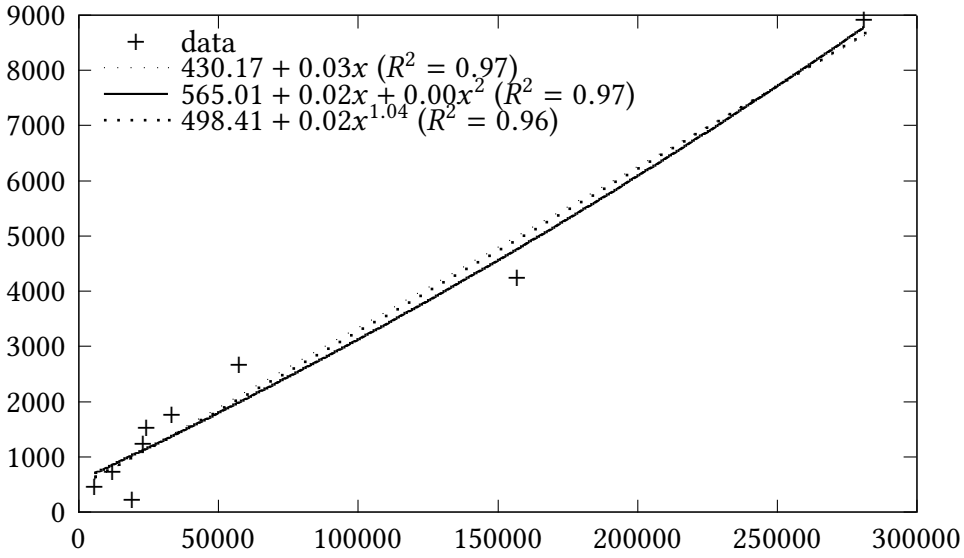


(b) Job execution time over number of Items, without outliers

Figure A.1.: Job execution time over number of Items selected by the query

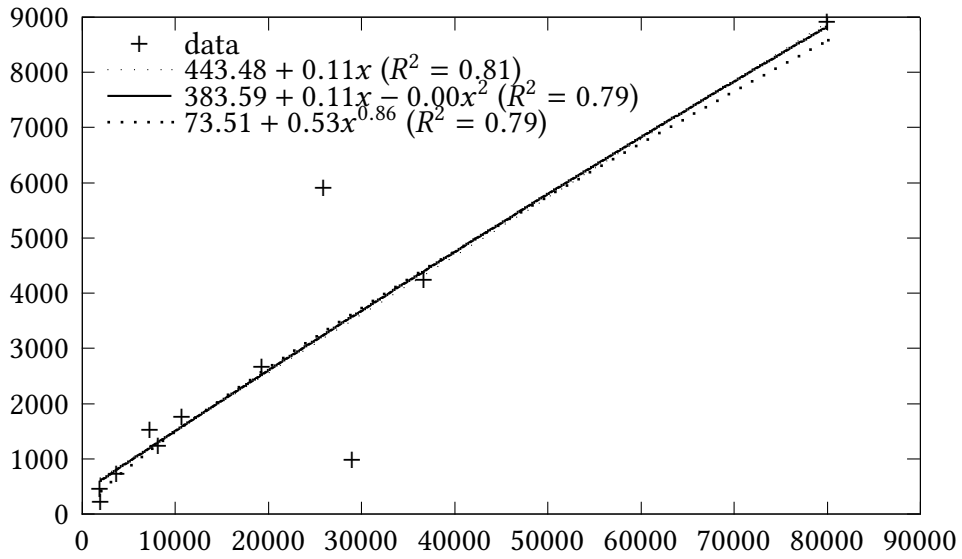


(a) Job execution time over number of triples, with outliers

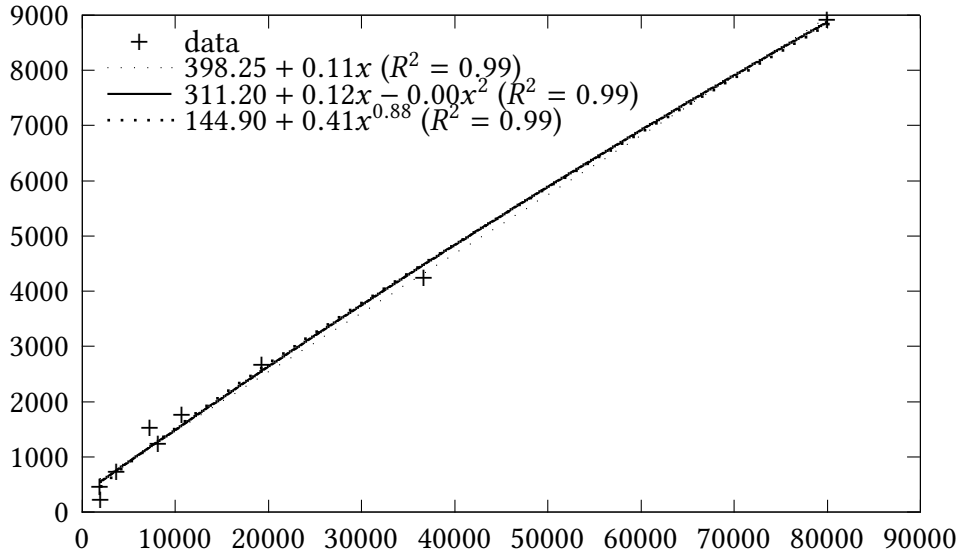


(b) Job execution time over number of triples, without outliers

Figure A.2.: Job execution time over number of triples in the input data set

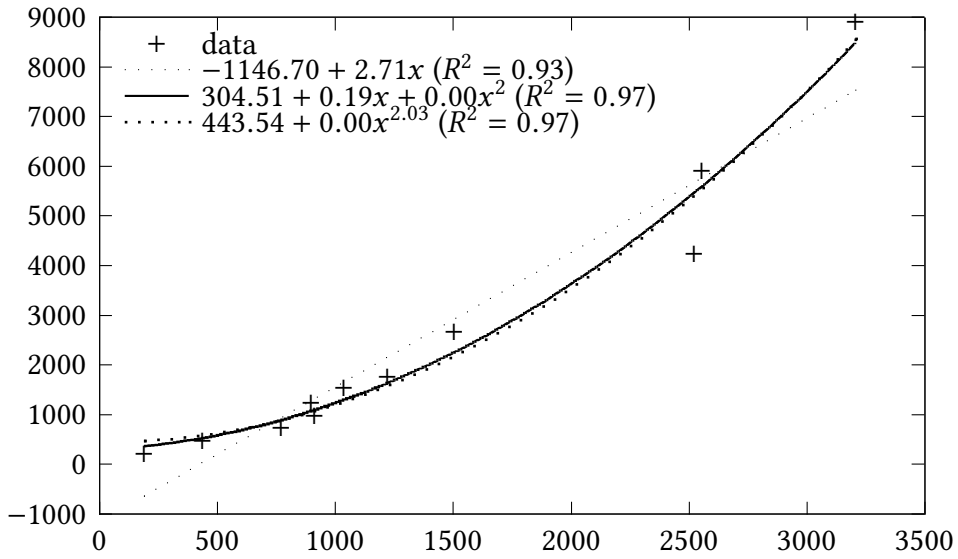


(a) Job execution time over number of `wdt:P31` triples, with outliers

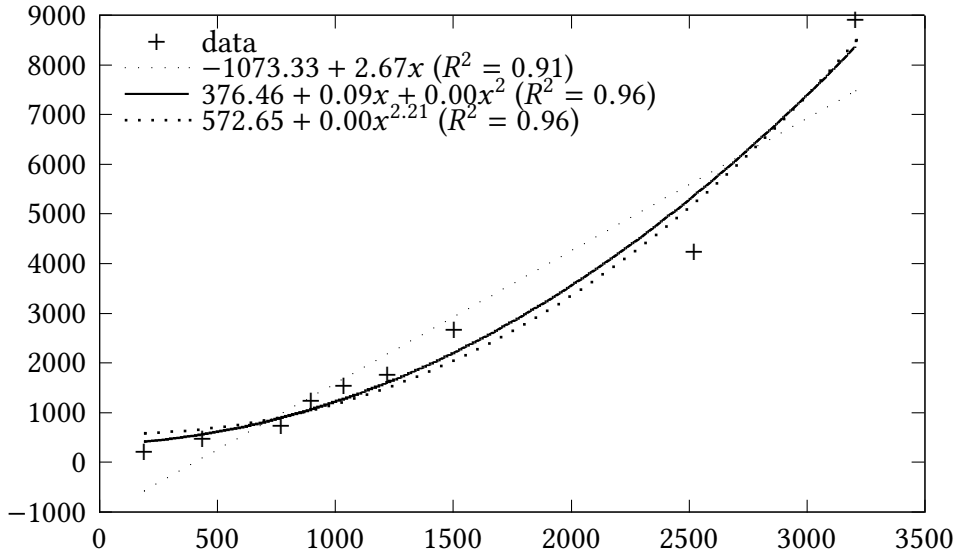


(b) Job execution time over number of `wdt:P31` triples, without outliers

Figure A.3.: Job execution time over number of `wdt:P31` triples in the input data set



(a) Job execution time over number of classes, with outliers



(b) Job execution time over number of classes, without outliers

Figure A.4.: Job execution time over number of distinct classes in the input data set