Conceptual Scaling of RDFS Ontologies

Jens Kötters[1], Peter W. Eklund[2] and Stefan E. Schmidt[1]

[1]Technische Universität Dresden, Germany
[2]Deakin University, Australia

Abstract

Conceptual scaling is a method, developed in the framework of Formal Concept Analysis (FCA), that transforms a many-valued context (a collection of objects, described by attributes with values, such as name, size or color) into a formal context (a binary schema from which formal concepts are derived). Relational scaling extends conceptual scaling, by taking relations between objects into account. Previous work applies relational scaling to a relational database, transforming the relational database into a relational structure. Relational structures play the role of formal contexts in a relational variant of FCA. In this paper, we present a similar approach, which applies relational scaling to RDFS ontologies.

Keywords

Conceptual Scaling, Formal Concept Analysis, Mediated Queries, RDFS

1. Introduction

Conceptual scaling is a method of preparing data for use with Formal Concept Analysis [1] (FCA), a mathematical theory of concepts. Originally defined for simple object-attribute data, the theory has been further developed to deal with relational data. We present a companion paper for an earlier paper [2], which describes scaling a relational database in the setting of a particular FCA variant [3] that connects FCA with database theory, adapted to the terminology of a classical paper [4] on FCA with relations. Nonetheless, this paper is self-contained; it presents a similar approach for the scaling of RDFS ontologies. Before we describe scaling for RDFS ontologies (Sect. 6) and its connection with querying RDF (Sect. 7), we provide an overview of Formal Concept Analysis (FCA) (Sect. 2), conceptual scaling (Sect. 3), FCA with relations (Sect. 4) and relational scaling (Sect. 5).

2. Formal Concept Analysis

Any introduction to Formal Concept Analysis (FCA) [1] would likely start with the definition of a formal context. A *formal context* is a triple (G, M, I), consisting of a set G of *objects*, a set M of *attributes*, and an *incidence relation* $I \subseteq G \times M$, where $(g, m) \in I$ means that the object g has the attribute m. Formal contexts are conventionally drawn as cross tables, like the "Addams Family" context in Figure 1, which describes a fictional family originally introduced in a TV series of 1964. The object set $G = \{\text{Wednesday}, \text{Gomez}, \text{Pugsley}, \text{Morticia}, \text{Fester}\}$ contains

CAOS VII: Cognition and Ontologies, 9th Joint Ontology Workshops (JOWO 2023), co-located with FOIS 2023, 19–20 July, 2023, Sherbrooke, Québec, Canada

Addams Family	male	female	black hair	psychic visions	electrokinesis
Wednesday		×	×	×	
Gomez	×		×		
Pugsley	×				
Morticia		×	×	×	
Fester	×				×

Figure 1: Example: Formal Context.

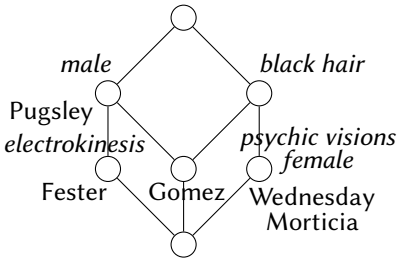


Figure 2: The concept lattice of the "Addams Family" context.

some of the family members; each is represented by a table row. The attributes, collected in the set $M = \{\text{male}, \text{female}, \text{black hair}, \text{psychic visions}, \text{electrokinesis}\}$, are represented by the table columns. A cross in the table indicates that the respective family member has the respective trait.

We summarize the basic definitions in Formal Concept Analysis (FCA) (cf. [1]). With any formal context there are two associated operations $(\cdot)^{\uparrow} : \mathfrak{P}(G) \to \mathfrak{P}(M)$ and $(\cdot)^{\downarrow} : \mathfrak{P}(M) \to \mathfrak{P}(G)$, defined by,

$$A^{\uparrow} := \{m \in M \mid (g, m) \in I \text{ for all } g \in A\} \quad , \tag{1}$$

$$B^{\downarrow} := \{g \in G \mid (g, m) \in I \text{ for all } m \in B\} \quad . \tag{2}$$

The set $A^{\uparrow}$ contains the attributes which are shared by all objects in A, and $B^{\downarrow}$ contains the objects which have all attributes in B. A *formal concept* of a context (G, M, I) is a pair $(A, B) \in \mathfrak{P}(G) \times \mathfrak{P}(M)$ with $A^{\uparrow} = B$ and $B^{\downarrow} = A$. The set A is the *extent* of the concept (A, B), i.e. the set of objects which belong to the concept; the set B is the *intent* of (A, B), i.e. the set of attributes which describe the concept. A concept (A, B) is a *subconcept* of a concept (C, D), written $(A, B) \leq (C, D)$, if $A \subseteq C$, or equivalently $D \subseteq B$. The set of all concepts of (G, M, I) is denoted by $\mathcal{B}(G, M, I)$, and the ordered set $\underline{\mathfrak{B}}(G, M, I) := (\mathcal{B}(G, M, I), \leq)$ is a complete lattice, called the *concept lattice* of (G, M, I).

Figure 2 shows the concept lattice of the "Addams Family" context. The seven nodes represent the concepts. The subconcepts of a concept are those concepts which lie below that concept, i.e. which can be reached by following the edges downward. Dually, by following the edges upward, we reach the *superconcepts* of a concept. For every object $g \in G$, the concept $\gamma(g) := (\{g\}^{\uparrow\downarrow}, \{g\}^{\uparrow})$ is the *object concept* of g; it is the smallest concept which has g in its extent. In Figure 2, the concept with the label g drawn below it is the object concept $\gamma(g)$. Dually, for every attribute $m \in M$, the concept $\mu(m) := (\{m\}^{\downarrow}, \{m\}^{\downarrow\uparrow})$ is the *attribute concept* of m; it is the largest concept which has m in its intent. In Figure 2, the concept with the label m drawn above it is the attribute concept $\mu(m)$. A concept has g in its extent iff it is a superconcept of $\gamma(g)$, and it has m in its intent iff it is a subconcept of $\mu(m)$. So Figure 2 shows the following concepts: the bottom concept is $(\emptyset, M)$; the three concepts directly above it are $(\{\text{Fester}\}, \{\textit{male}, \textit{electrokinesis}\})$, $(\text{Gomez}, \{\textit{male}, \textit{black hair}\})$ and $(\{\text{Wednesday}, \text{Morticia}\}, \{\textit{female}, \textit{black hair}, \textit{psychic visions}\})$; the two concepts directly below the top concept are $(\{\text{Fester}, \text{Gomez}, \text{Pugsley}\}, \{\textit{male}\})$ and $(\{\text{Gomez}, \text{Wednesday}, \text{Morticia}\},$

	nationality	date_of_birth
Lewis Carroll	GB	1832-01-27
Virginia Woolf	GB	1882-01-25
Douglas Adams	GB	1952-03-11
Neil Gaiman	GB	1960-11-10
J. K. Rowling	GB	1965-07-31
Stephen King	US	1947-09-21
Dan Brown	US	1964-06-22

Figure 3: Example: Many-valued context.

$\{$*black hair*$\}$); finally, the top concept is $(G, \emptyset)$. The diagram in Figure 2 is called a *line diagram* (or *Hasse diagram*) with *reduced labeling* (cf. [1, p.23]).

Formal Concept Analysis (FCA) supports a diverse range of applications. If the context is small enough, the concept lattice provides a visualization of the data, which may provide some insights; this is arguably the most straightforward application of FCA [5]. Concept lattices have been used as a data model for document browsing and navigation, pioneered by Godin et al. [6] and further elaborated by others [7]. The theory of attribute implications [1] supports data analysis and knowledge discovery; for example, the attribute implications *psychic visions* $\rightarrow$ *female*, *psychic visions* $\rightarrow$ *black hair* and *electrokinesis* $\rightarrow$ *male* hold in the "Addams Family" context. Association rules, considered in data mining, may be considered a probabilistic variant of attribute implications, described by the measures of support and confidence; Pasquier et al. [8] describe association rule mining in an FCA context. Stumme and Maedche have applied FCA to merge ontologies [9].

3. Conceptual Scaling

Formal Concept Analysis (FCA) also defines another type of context, called a *many-valued context* [1, Sect. 1.3]. A many-valued context can be compared to a single table in a database (with no foreign keys); correspondingly, the attributes in a many-valued context take values, and are thus called *many-valued attributes*. Figure 3 shows the many-valued context "Authors" (adapted from [2]). Its object set G consists of the seven authors, listed in the first row, and it has the attribute set $M = \{$*nationality*, *date_of_birth*$\}$. A concept lattice is not defined on a many-valued context directly; instead, in FCA there is a procedure, called *conceptual scaling*, which transforms a many-valued context into a formal context, and a concept lattice is then obtained from that formal context.

We illustrate conceptual scaling by the example of the "Authors" context. First, we associate a *conceptual scale* with each many-valued attribute, i.e. a formal context whose objects are the possible values of that attribute. For example, the "Regions" scale in Figure 4 can be chosen for the *nationality* attribute, because its objects are ISO 3166 country codes, and the "Centuries" scale in Figure 5 can be chosen for the *date_of_birth* attribute, because its objects are ISO 8601 dates. Figure 5 only indicates the date range, because there are obviously too many dates to be listed. Each scale attribute represents a subset of the values. For example, the attributes *19C*, *20C* and *21C* of the "Centuries" represent certain date intervals (namely the 19th, 20th and

Regions	US	GB	FR	DE	Europe
US	×				
GB		×			×
FR			×		×
DE				×	×

Figure 4: Regions Scale.

Centuries	19C	20C	21C
1800-01-01	×		
1800-01-02	×		
...			
2099-12-30			×
2099-12-31			×

Figure 5: Centuries Scale.

	US	GB	FR	DE	Europe
Lewis Carroll		×			×
Virginia Woolf		×			×
Douglas Adams		×			×
Neil Gaiman		×			×
J.K. Rowling		×			×
Stephen King	×				
Dan Brown	×				

	19C	20C	21C
Lewis Carroll	×		
Virginia Woolf	×		
Douglas Adams		×	
Neil Gaiman		×	
J.K. Rowling		×	
Stephen King		×	
Dan Brown		×	

Figure 6: Realized Scales.

Authors	US	GB	FR	DE	Europe	19C	20C	21C
Lewis Carroll		×			×	×		
Virginia Woolf		×			×	×		
Douglas Adams		×			×		×	
Neil Gaiman		×			×		×	
J.K. Rowling		×			×		×	
Stephen King	×						×	
Dan Brown	×						×	

Figure 7: Scaled Context.

21st century). The scale attributes *US*, *GB*, *FR* and *DE* represent the corresponding singleton sets of values. A scale which identifies precisely the singleton sets of values is called a *nominal scale* (cf. [1, p.42]). However, to make the example more interesting, we have added a *Europe* attribute, which identifies the European countries.

As we have seen, each scale attribute a identifies a subset $V(a)$ of the possible values. When the scale is associated with a many valued attribute m, the scale attributes also apply to the objects of the many-valued context: we say that an object g has the attribute a if and only if the value of g in m lies in $V(a)$. This means, in our example, an author has the attribute *19C* if they were born in the 19th century, and the attribute *Europe* if they have some European nationality. In this way, each conceptual scale translates to a *realized scale* (which has the same attributes), as shown in Figure 6. Note that each object's row is identical to its value's row in the conceptual scale. The *scaled context* (see Figure 7) is obtained by placing the realized scales side-by-side (this is called an *apposition* of contexts [1]).

$\mathbb{K}_1$	male	female	black hair	psychic visions	electrokinesis
Wednesday		×	×	×	
Gomez	×		×		
Pugsley	×				
Morticia		×	×	×	
Fester	×				×

$\mathbb{K}_2$	motherOf	fatherOf	parentOf	brotherOf
(Morticia,Wednesday)	×		×	
(Morticia,Pugsley)	×		×	
(Gomez,Wednesday)		×	×	
(Gomez,Pugsley)		×	×	
(Gomez,Fester)				×
(Fester,Gomez)				×

Figure 8: Extending the "Addams Family" context to a power context family.

4. Formal Concept Analysis with Relations

The Addams Family context in Figure 1 describes individual family members by their traits, but despite the title, the context provides no information about the family relations. This hints at a limitation of formal contexts as a data model: they only support a limited sentence structure ("object has attribute"). A milestone paper by Wille [4] introduces power context families as a means to model relational data in Formal Concept Analysis (FCA): a *power context family* is there defined as a sequence $\vec{\mathbb{K}} = (\mathbb{K}_1, \ldots, \mathbb{K}_n)$ such that, if G denotes the object set of $\mathbb{K}_1$, the object set of $\mathbb{K}_k$ is a subset of G^k for all $2 \leq k \leq n$. For example, Figure 8 shows a power context family $(\mathbb{K}_1, \mathbb{K}_2)$ for the Addams family, where $\mathbb{K}_1$ is the same context as in Figure 1. The context $\mathbb{K}_2$ states that Morticia is the mother of Wednesday and Pugsley, Gomez is their father, and Gomez and Fester are brothers.

In the same way we have obtained a concept lattice $\underline{\mathcal{B}}(\mathbb{K}_1)$ for the context $\mathbb{K}_1$ (cf. Figure 2), we can also obtain a concept lattice $\underline{\mathcal{B}}(\mathbb{K}_2)$ for the context $\mathbb{K}_2$. Its concepts are called *relation concepts*, because their extents are (binary) relations over G. The stated goal of Wille's approach was the formalization of traditional philosophical logic, and its notions of *concepts*, *judgments* and *conclusions*. With concepts already formalized by FCA, Wille introduced *concept graphs* for the formalization of judgments (i.e. "statements"); concept graphs are a variant of Sowa's conceptual graphs [10], where two maps κ and ϱ identify each node v with a concept $\kappa(v)$ and also with a non-empty set $\varrho(v)$ of objects (or tuples, for relation concepts) from the extent of $\kappa(v)$. The map ϱ is called a *realization*; it ensures that the statement holds in the given power context family.

In a concept graph, formal concepts are placed side-by-side, but they are not combined to form new concepts. For example, we can build a concept graph which states that Fester is a brother of Gomez, and Gomez is a parent of Wednesday and Pugsley, but we do not obtain a relational concept *uncle*, with an extent of $\{(\mathsf{Fester}, \mathsf{Wednesday}), (\mathsf{Fester}, \mathsf{Pugsley})\}$. Huchard et al. [11] introduced *Relational Concept Analysis* (RCA), which defines concepts over a *relational context family* (similar to a power context family, but with different formal contexts for objects of different sorts). With RCA, the concepts *uncle* and *nibling* (the generalization of niece and nephew) can be obtained, but only as unary concepts (i.e. the extents are the sets of all uncles and niblings, respectively). Such concepts can also be obtained using a comparable approach by Baader and Molitor [12], which combines Relational Concept Analysis with Description Logics; the approach uses *pattern concepts* in the sense of Ganter and Kuznetsov [13], i.e. it is based on

a variant of FCA where concept intents are not represented by sets of attributes, but e.g. by graphs.

Kötters [3] defines concept lattices over a relational structure using *pattern concepts* with conjunctive queries as intents. Both Relational Concept Analysis and Formal Concept Analysis with Description Logics allow more expressive intents (using e.g. universal quantification); but the limitation to conjunctive queries allows one to obtain, not only unary concepts, but relational concepts of any arity. A subsequent paper [14] rephrases the approach in the terminology of Wille, using a power context family instead of a relational structure, and formalizing conjunctive queries by "windowed intension graphs", which are modeled after concept graphs. More precisely, windowed intension graphs are a kind of *abstract concept graph*, a term introduced by Wille [4] but seldom used thereafter, for concept graphs that are independent from any data (in particular, they lack the realization ϱ), which makes them suitable to formalize queries; we discuss querying in Sect. 7. In summary, every power context family $\vec{\mathbb{K}}$ is associated with a concept lattice $\underline{B}(\mathbb{K})$, and for the power context family in Figure 8, this concept lattice contains for example the (binary) *uncle* concept.

5. Relational Scaling

The term *relational scaling* was coined by Prediger and Wille [15], as an extension of *conceptual scaling*, which transforms a many-valued context with relational data into a power context family. Hereth [16] describes relational scaling of relational databases. The Formal Concept Analysis (FCA) variant of Kötters [3] (cf. Sect. 4) establishes a connection between FCA and database theory; Kötters and Eklund [2] revisit the topic of scaling a relational database in light of this connection, which complements the theoretical results with some practical concerns. Figure 9 illustrates the idea of using the same concept model with differents kinds of relational data, using the power context family as a kind of interface. This requires one to specify, for each kind of relational data, how it is transformed into a power context family, i.e. to devise a method of relational scaling. In Sect. 6, we describe a possible way of scaling an RDFS ontology, which corresponds to the upper right arrow in Figure 9. This approach is more elegant and economic than a similar previous attempt [17], which defines concepts over an RDFS ontology directly.

The scaling affects subsequent applications, which have been developed in the context of Formal Concept Analysis (FCA), and the brief overview at the end of Sect. 2 may serve as an example. It is the basis for the definition of concepts, i.e. it determines how we conceptualize the data. Similarly, it defines the language, within which attribute implications (or pattern implications, in the case of pattern concepts) are expressed, and the same holds for association rules. As described by Priss [18], the scales correspond to facets in a faceted navigation approach (see e.g. [2]).

6. RDFS Ontologies

According to the RDF 1.1 "Concepts and Abstract Syntax specification" [19], an *RDF graph* is a set of *RDF triples*, where each triple consists of a *subject*, a *predicate* and an *object*. The predicate is an

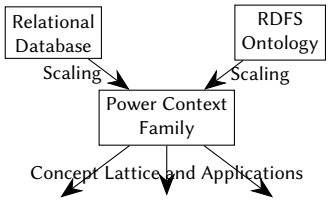


Figure 9: Power context families as an interface between relational data and Formal Concept Analysis applications.

internationalized resource identifier (IRI), such as `<http://schema.org/knows>`, recognizable by the angular brackets. The subject is either an IRI or a *blank node*; a blank node represents some unspecified entity. In the Turtle notation [20] (which we use in this paper), blank nodes are written $\_:x$, where x is an identifier for the blank node. Finally, an object is either an IRI or a blank node or a *literal*, such as "14"`<http://www.w3.org/2001/XMLSchema#integer>`, where the quoted part is the *lexical form*, followed by a double caret and a *datatype IRI* (i.e. an IRI which refers to a datatype); additionally, certain literals can have a language tag; for those, we refer to the specification [19].

Figure 10 shows an excerpt of an RDF document, which describes an RDF graph in Turtle notation [20]. The `@base` and `@prefix` directives at the beginning allow some shorthand notations for IRI's. In particular, the RDF terms with a prefix, such as `rdf:type`, denote IRI's; literals which lack a datatype IRI are assigned the string datatype; and if a triple ends with a semicolon (instead of a period), the next triple has the same subject (so the subject is omitted).

In this section, we describe conceptual scaling for RDF graphs that use the RDF Schema [21] vocabulary (i.e. RDFS ontologies). As the RDF Primer [22] states, the main modeling constructs for RDFS ontologies are the classes `rdfs:Class` and `rdf:Property`, and the properties `rdf:type`, `rdfs:subClassOf`, `rdfs:subPropertyOf`, `rdfs:domain` and `rdfs:range` (using the prefix notation for these IRI's, where the prefixes `rdf` and `rdfs` are resolved as usual, cf. [19, Sect. 1.4]). Every IRI that is used as a predicate denotes a property, so it belongs to the class `rdf:Property`. The property `rdf:type` is used to state that an entity belongs to a certain class; e.g. the triple `<#AliceInWonderland> rdf:type <#Book>` in Figure 10 states that the entity `<#AliceInWonderland>`, which we can see is Lewis Carroll's "Alice in Wonderland", belongs to the class `<#Book>`. The properties `rdfs:subClassOf` and `rdfs:subPropertyOf` are used to define the hierarchies of classes and properties, respectively. The class `rdfs:Resource` is at the top of the class hierarchy; it contains all entities (the term *resource* is used as a synonym for entity). The properties `rdfs:domain` and `rdfs:range` state, for a given property, what classes the entities in the subject and object positions are expected to belong to, respectively.

We now describe how an RDFS ontology is converted into a power context family. For this purpose, we imagine that the schema part of the ontology is represented in the fashion of an API specification as e.g. generated by Javadoc for the Java programming language (where each class is described by its own page, along with its attributes and methods, defined on the class itself, or inherited from a superclass). Indeed, the popular Schema.org ontology comes with a similar documentation (cf. https://schema.org/docs/schemas.html). Evidently, the classes of the ontology are the entities of type `rdfs:Class`, and the class hierarchy is described by the

```
<#Author> rdf:type rdfs:Class .
<#Book> rdf:type rdfs:Class .
<#AliceInWonderland> rdf:type <#Book> ; dcterms:creator <#LewisCarroll> ;
         dcterms:issued "1865-11-26"^^xsd:date .
<#ToTheLighthouse> rdf:type <#Book> ; dcterms:creator <#VirginiaWoolf> ;
         dcterms:issued  "1927-05-05"^^xsd:date .
<#HitchhikersGuide> rdf:type <#Book> ; dcterms:creator <#DouglasAdams> ;
         dcterms:issued  "1979-10-12"^^xsd:date .
<#TriggerWarning> rdf:type <#Book> ; dcterms:creator <#NeilGaiman> ;
         dcterms:issued  "2015-02-03"^^xsd:date .
<#HarryPotter7> rdf:type <#Book> ; dcterms:creator <#JKRowling> ;
         dcterms:issued  "2007-07-21"^^xsd:date .
<#TheCasualVacancy> rdf:type <#Book> ; dcterms:creator <#JKRowling> ;
         dcterms:issued  "2012-09-27"^^xsd:date .
<#TheShining> rdf:type <#Book> ; dcterms:creator <#StephenKing> ;
         dcterms:issued  "1977-01-28"^^xsd:date .
<#DoctorSleep> rdf:type <#Book> ; dcterms:creator <#StephenKing> ;
         dcterms:issued  "2013-09-24"^^xsd:date .
<#TheDaVinciCode> rdf:type <#Book> ; dcterms:creator <#DanBrown> ;
         dcterms:issued  "2003-03-18"^^xsd:date .
<#Inferno> rdf:type <#Book> ; dcterms:creator <#DanBrown> ;
          dcterms:issued  "2013-03-14"^^xsd:date .
<#LewisCarroll> rdf:type <#Author> ; schema:nationality "GB" ;
           schema:birthDate "1832-01-27"^^xsd:date .
<#VirginiaWoolf> rdf:type <#Author> ; schema:nationality "GB" ;
           schema:birthDate "1882-01-25"^^xsd:date .
<#DouglasAdams> rdf:type <#Author> ; schema:nationality "GB" ;
           schema:birthDate "1952-03-11"^^xsd:date .
<#NeilGaiman> rdf:type <#Author> ; schema:nationality "GB" ;
           schema:birthDate "1960-11-10"^^xsd:date .
<#JKRowling> rdf:type <#Author> ; schema:nationality "GB";
           schema:birthDate "1965-07-31"^^xsd:date .
<#StephenKing> rdf:type <#Author> ; schema:nationality "US" ;
           schema:birthDate "1947-09-21"^^xsd:date .
<#DanBrown> rdf:type <#Author> ; schema:nationality "US";
           schema:birthDate "1964-06-22"^^xsd:date .
```

Figure 10: Sample ontology "library.ttl"

`rdfs:subClassOf` property. We divide the properties of the ontology (i.e. the entities of type `rdf:property`) into two classes, which correspond roughly to attributes and methods in Java: if the `rdfs:range` of a property is a datatype (i.e. an entity of type `rdfs:Datatype`), we call the property a *many-valued attribute* (in line with FCA terminology), otherwise we call it a *relation*. Naturally, the IRI's in the data are considered *objects* (in line with FCA terminology), and the RDF literals are considered *values*. A many-valued attribute then associates values to the objects in its `rdfs:domain`, and a relation links objects in its `rdfs:domain` to objects in its `rdfs:range`. Blank nodes are treated as values if they occur in the `rdfs:range` of a many-valued attribute, otherwise they are treated as objects.

In practice, ontologies may prove inconsistent with respect to our strong typing assumptions. In this case, we try to apply workarounds; e.g. if a property is used as a many-valued attribute in one place, and as a relation in another, we may treat it as two distinct properties. Conceptual scaling can be done interactively, by the use of a program, which presents an API-like view of classes, where individual classes, and properties on them, can be toggled on (*enabled*) and off

$\mathbb{K}_0$: Classes	<#Author>	<#Book>
<#LewisCarroll>	×	
<#VirginiaWoolf>	×	
<#DouglasAdams>	×	
<#NeilGaiman>	×	
<#JKRowling>	×	
<#StephenKing>	×	
<#DanBrown>	×	
<#AliceInWonderland>		×
<#ToTheLighthouse>		×
<#HitchhikersGuide>		×
<#TriggerWarning>		×
<#HarryPotter7>		×
<#TheCasualVacancy>		×
<#TheShining>		×
<#DoctorSleep>		×
<#TheDaVinciCode>		×
<#Inferno>		×

$\mathbb{K}_2$: Relations	dcterms:creator
(<#AliceInWonderland>,<#LewisCarroll>)	×
(<#ToTheLighthouse>,<#VirginiaWoolf>)	×
(<#HitchhikersGuide>,<#DouglasAdams>)	×
(<#TriggerWarning>,<#NeilGaiman>)	×
(<#HarryPotter7>,<#JKRowling>)	×
(<#TheCasualVacancy>,<#JKRowling>)	×
(<#TheShining>,<#StephenKing>)	×
(<#DoctorSleep>,<#StephenKing>)	×
(<#TheDaVinciCode>,<#DanBrown>)	×
(<#TheDaVinciCode>,<#DanBrown>)	×

Figure 11: The contexts $\mathbb{K}_0$ and $\mathbb{K}_2$ of the scaled RDFS ontology.

(*disabled*); disabled classes and properties are treated as not present. The number of classes and properties can be overwhelming, so *disabled* will be the default. Any edits or workarounds in case of inconsistencies, as described above, need only be applied to enabled classes and properties. If a property is enabled, its `rdfs:domain` and `rdfs:range` are enabled automatically.

We translate an RDF graph into a power context family $(\mathbb{K}_0, \mathbb{K}_1, \mathbb{K}_2)$, and set $\mathbb{K}_k =: (G_k, M_k, I_k)$ for $k \in \{0, 1, 2\}$. The attributes of $\mathbb{K}_0$ and $\mathbb{K}_2$ are the enabled classes and relations, respectively. The objects of $\mathbb{K}_0$ are the IRIs and blank nodes which belong to at least one enabled class. Each triple `subj rdf:type obj` in the ontology produces a pair $(\texttt{subj}, \texttt{obj}) \in I_0$. Further pairs are added to I_0 where the class membership can be inferred via `rdfs:subClassOf`. Likewise, each triple `subj, pred, obj` in the ontology produces an instance $(\texttt{subj}, \texttt{obj}) \in G_2$ and a pair $((\texttt{subj}, \texttt{obj}), \texttt{pred}) \in I_2$. Further pairs are added to I_2 where the property can be inferred via `rdfs:subPropertyOf`. Figure 11 shows the contexts $\mathbb{K}_0$ and $\mathbb{K}_2$ for the RDFS ontology in Figure 10.

It now remains to produce realized scales for the IRI's classified as many-valued attributes; the context $\mathbb{K}_1$ is then obtained as the apposition of the realized scales (cf. Figures 6 and 7). However, using conceptual scales, like those in Figures 4 and 5, does not seem practical. As we have mentioned, each scale attribute identifies a subset of the values, and if we can specify this subset by other means, we effectively represent the conceptual scale. *Logical scaling* [23] is a variant of conceptual scaling, where the subset of values is represented by a formal expression. For RDFS ontologies, suitable formal expressions are `FILTER` expressions, which can be used in SPARQL queries [24]. For example, the range of values identified by the *20C* attribute of the "Centuries" scale is represented by the expression `FILTER(?value >= "1900-01-01"xsd:date && ?value < "2000-01-01"xsd:date)`, where the prefix `xsd` is resolved, as usual, to `http://www.w3.org/2001/XMLSchema#` (cf. [19, Sect. 1.4]). The FILTER condition for each attribute is then substituted into a SPARQL query

dcterms:issued	19C	20C	21C
<#LewisCarroll>			
<#VirginiaWoolf>			
<#DouglasAdams>			
<#NeilGaiman>			
<#JKRowling>			
<#StephenKing>			
<#DanBrown>			
<#AliceInWonderland>	×		
<#ToTheLighthouse>		×	
<#HitchhikersGuide>		×	
<#TriggerWarning>			×
<#HarryPotter7>			×
<#TheCasualVacancy>			×
<#TheShining>		×	
<#DoctorSleep>			×
<#TheDaVinciCode>			×
<#Inferno>			×

schema:birthDate	19C	20C	21C
<#LewisCarroll>	×		
<#VirginiaWoolf>	×		
<#DouglasAdams>		×	
<#NeilGaiman>		×	
<#JKRowling>		×	
<#StephenKing>		×	
<#DanBrown>		×	
<#AliceInWonderland>			
<#ToTheLighthouse>			
<#HitchhikersGuide>			
<#TriggerWarning>			
<#HarryPotter7>			
<#TheCasualVacancy>			
<#TheShining>			
<#DoctorSleep>			
<#TheDaVinciCode>			
<#Inferno>			

Figure 12: Realized Scales.

```
SELECT DISTINCT ?subj
WHERE { ?subj prefix:mva ?value . FILTER(...) }
```

which determines the column for that attribute in the realized scale (a cross indicates that the object is in the query's result set). Figure 12 shows the realized scales for the many-valued attributes `dcterms:issued` and `schema:birthDate`, which are both scaled with the (logical) "Centuries" scale. Similarly, a realized scale is obtained for the `schema:nationality` attribute (where the `Europe` column represents a union), and the apposition of realized scales produces the context $\mathbb{K}_1$ in Figure 13.

7. Navigation

In this section, we discuss the practical significance of conceptual scaling, in the context of a navigation application. More specifically, we plan to extend the Granada application [25], which allows scaling and navigating in relational databases, to RDFS ontologies. The functionality can be compared with the OpenLink Faceted Browser, which is accessible via the "Browse using" menu on any DBpedia page (e.g. https://dbpedia.org/page/Berlin). At the core, such an application processes SPARQL queries, and presents the results. OpenLink's Faceted Browser uses a text representation of queries, whereas Granada has so far used a graph representation that corresponds to abstract concept graphs (cf. Sect. 4). Figure 14 shows an example for such a graph query; it was originally presented in [2], and it is shown there how it translates into an SQL query. We now show how it translates into a SPARQL query.

The rectangular nodes represent variables, say `?z1` (left node) and `?z2` (right node). The class labels on the nodes are attributes from $\mathbb{K}_0$. The rounded nodes carry attributes from $\mathbb{K}_1$ (one outgoing edge) or from $\mathbb{K}_2$ (two outgoing edges). A rectangular node is colored if it represents a subject of the query; i.e. in Fig. 14, we are asking about the authors, not about the books. Every subject is associated with an output variable (i.e. a column header in the result table), separated

$\mathbb{K}_1$: RealizedScales	dcterms:issued:19C	dcterms:issued:20C	dcterms:issued:21C	schema:birthDate:19C	schema:birthDate:20C	schema:birthDate:21C	schema:nationality:GB	schema:nationality:US	schema:nationality:FR	schema:nationality:DE	schema:nationality:Europe
<#LewisCarroll>				×			×				×
<#VirginiaWoolf>				×			×				×
<#DouglasAdams>					×		×				×
<#NeilGaiman>					×		×				×
<#JKRowling>					×		×				×
<#StephenKing>					×			×			
<#DanBrown>					×			×			
<#AliceInWonderland>	×										
<#ToTheLighthouse>		×									
<#HitchhikersGuide>		×									
<#TriggerWarning>			×								
<#HarryPotter7>			×								
<#TheCasualVacancy>			×								
<#TheShining>		×									
<#DoctorSleep>			×								
<#TheDaVinciCode>			×								
<#Inferno>			×								

Figure 13: The context $\mathbb{K}_1$ of the scaled RDFS ontology.

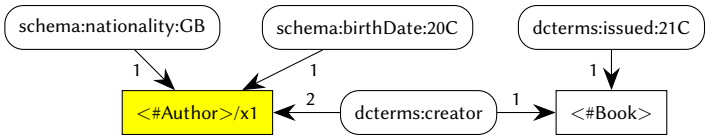
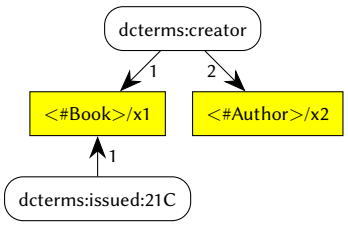


Figure 14: Graph query "20th-century-born British authors who published in the 21st century".

from the class label by a slash; so `?x1` is an output variable for the author node. The graph thus corresponds to the following query:

```
SELECT DISTINCT (?z1 AS ?x1)
WHERE {
  ?z1 rdf:type <#Author> .
  ?z2 rdf:type <#Book> .
  ?z2 dcterms:creator ?z1 .
  ?z1 schema:birthDate ?z3 . FILTER(
    ?z3 >= "1900-01-01"^^xsd:date && ?z3 < "2000-01-01"^^xsd:date)
  ?z1 schema:nationality ?z4 . FILTER(?z4 = "GB")
  ?z2 dcterms:issued ?z5 . FILTER(
    ?z5 >= "2000-01-01"^^xsd:date && ?z5 < "2100-01-01"^^xsd:date)
} .
```

Every attribute in $\mathbb{K}_0$, $\mathbb{K}_1$ or $\mathbb{K}_2$ contributes to one statement in the query's WHERE-clause;



?x1	?x2
<#NeilGaiman>	<#TriggerWarning>
<#JKRowling>	<#HarryPotter7>
<#JKRowling>	<#TheCasualVacancy>
<#StephenKing>	<#DoctorSleep>
<#DanBrown>	<#TheDaVinciCode>
<#DanBrown>	<#Inferno>

Figure 15: Graph query "21st-century books and their authors" and its result table.

additionally, each attributes in $\mathbb{K}_1$ is related to a FILTER-expression, cf. Sect. 6. Obviously, the renaming of `?z1` to `?x1`, in the first line of the query, could have been avoided if we had chosen `?x1` instead of `?z1` in the first place; the present form of the query is slightly more generic, since it allows to associate several output variables with the same node (which seems useless in practice, but has some theoretical significance).

As the example shows, the attributes of $\mathbb{K}_0$, $\mathbb{K}_1$ and $\mathbb{K}_2$, hence the power context family, determine the query language. The power context family thus defines an abstraction from the underlying query language (be it SQL or SPARQL), limiting the possible queries on the one hand, but providing a discrete set of options on the other hand, which should allow for a quicker and more user-friendly way of navigation; comparable to how users access a library catalogue through a search mask, rather than typing queries directly. Essentially, the choice of conceptual scales determines the available fields in the search mask.

As in the OpenLink Faceted Browser, query building is interactive, i.e. the user would start with a single graph node (one of the rectangular nodes, say the author node), and the system provides suggestions how the graph can be extended (by many-valued attributes, or relations to other nodes), while still allowing a non-empty result set. But the use of the power context family, and the connection to FCA it provides, allows for another feature: the computation of commonalities. Mathematically, the power context family associates every query Q to a formal concept $(Q^{\downarrow}, Q^{\downarrow\uparrow})$, where $Q^{\downarrow}$ is the result table for Q, and $Q^{\downarrow\uparrow}$ is the *graph closure* of Q, a graph that shows what the tuples in $Q^{\downarrow}$ have in common; it thus provides additional information. In theory, this information could be computed and presented alongside the result table. In practice, this is not generally possible, because the graph closures are far too complex to compute, or to be read and understood by a user. However, it is possible to compute a weaker form of closure. Specifically, the *node closure* computes closures on a per-facet basis, i.e. individually for each realized scale. An example should make this clear. Consider the query in Fig. 15, and its result table. For the set $A := \{$`<#NeilGaiman>`, `<#JKRowling>`, `<#StephenKing>`, `<#DanBrown>`$\}$ of objects in the first column, and the realized scale `schema:birthDate` in Fig. 12, we obtain $A^{\uparrow} = \{20C\}$ (cf. Sect. 2). In other words, all authors in the result table were born in the 20th century. This information can be presented alongside with the result table. Doing so for all table columns and all realized scales amounts to computing the node closure. Other weaker forms of closure may take the graph structure (i.e. relations) into account (cf. pattern projections in [13]). The power context family can be virtual, i.e. it need not be computed in memory, although of course, obtaining commonalities involves some computational overhead.

8. Conclusion

Figure 9 illustrates an idea how FCA can be applied to different kinds of relational data, using the power context family as an intermediary representation. The idea involves to specify, for each kind of relational data, a method of conceptual scaling, which explains how the power context family is derived. In a previous paper [2], this has been shown for relational databases, and in this paper, we have shown this for RDFS ontologies. The practical use has been demonstrated for the particular case of navigation. On this basis, we can extend Granada [25] to RDFS ontologies.

References

[1] B. Ganter, R. Wille, Formal concept analysis: mathematical foundations, Springer, Berlin, 1999.

[2] J. Kötters, P. W. Eklund, The theory and practice of coupling formal concept analysis to relational databases, in: S. O. Kuznetsov, A. Napoli, S. Rudolph (Eds.), Proceedings of FCA4AI 2018, volume 2149 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2018, pp. 69–80.

[3] J. Kötters, Concept lattices of a relational structure, in: H. D. Pfeiffer, D. I. Ignatov, J. Poelmans, N. Gadiraju (Eds.), Proceedings of ICCS 2013, volume 7735 of *LNCS*, Springer, 2013, pp. 301–310.

[4] R. Wille, Conceptual graphs and formal concept analysis, in: D. Lukose, H. S. Delugach, M. Keeler, L. Searle, J. F. Sowa (Eds.), Proceedings of ICCS 1997, 5th Intl. Conf. on Conceptual Structures, volume 1257 of *LNCS*, Springer, Heidelberg, 1997, pp. 290–303.

[5] P. Eklund, J. Ducrou, P. Brawn, Concept lattices for information visualization: Can novices read line-diagrams?, in: International Conference on Formal Concept Analysis, Springer, 2004, p. 57–73.

[6] R. Godin, E. Saunders, J. Gecsei, Lattice model of browsable data spaces, Inf. Sci. 40 (1986) 89–116.

[7] R. Cole, P. Eklund, Browsing semi-structured web texts using formal concept analysis, in: H. S. Delugach, G. Stumme (Eds.), Conceptual Structures: Broadening the Base, Springer Berlin Heidelberg, Berlin, Heidelberg, 2001, pp. 319–332.

[8] N. Pasquier, Y. Bastide, R. Taouil, L. Lakhal, Efficient mining of association rules using closed itemset lattices, Information Systems 24 (1999) 25–46.

[9] G. Stumme, A. Maedche, FCA-Merge: Bottom-up merging of ontologies, in: B. Nebel (Ed.), Proceedings of IJCAI 2001, Morgan Kaufmann, 2001, pp. 225–230.

[10] J. F. Sowa, Conceptual Structures: Information Processing in Mind and Machine, Addison-Wesley, Reading, MA, 1984.

[11] M. Huchard, C. Roume, P. Valtchev, When concepts point at other concepts: the case of UML diagram reconstruction, in: Proceedings of the 2nd Workshop on Advances in Formal Concept Analysis for Knowledge Discovery in Databases (FCAKDD 2002), 2002, pp. 32–43.

[12] F. Baader, R. Molitor, Building and structuring description logic knowledge bases using least common subsumers and concept analysis, in: B. Ganter, G. W. Mineau (Eds.), Proceedings of ICCS 2000, volume 1867 of *LNCS*, Springer, Berlin, Heidelberg, 2000, pp. 292–305.

[13] B. Ganter, S. O. Kuznetsov, Pattern structures and their projections, in: H. S. Delugach, G. Stumme (Eds.), Proceedings of ICCS 2001, volume 2120 of *LNCS*, Springer, 2001, pp. 129–142.
[14] J. Kötters, Intension graphs as patterns over power context families, in: M. Huchard, S. Kuznetsov (Eds.), Proceedings of CLA 2016, volume 1624 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2016, pp. 203–216.
[15] S. Prediger, R. Wille, The lattice of concept graphs of a relationally scaled context, in: W. R. C. William M. Tepfenhart (Ed.), Proceedings of ICCS 1999, volume 1640 of *LNAI*, Springer, 1999, pp. 401–414.
[16] J. Hereth, Relational scaling and databases, in: U. Priss, D. Corbett, G. Angelova (Eds.), Proceedings of ICCS 2002, volume 2393 of *LNAI*, Springer, Heidelberg, 2002, pp. 62–76.
[17] J. Kötters, Concept lattices of RDF graphs, in: M. Ojeda-Aciego, J. Baixeries, C. Sacarea (Eds.), Proceedings of the International Workshop on Formal Concept Analysis and Applications 2015, volume 1434 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2015, pp. 81–91. URL: http://ceur-ws.org/Vol-1434.
[18] U. Priss, Lattice-based information retrieval, Knowledge Organization 27 (2000) 132–142.
[19] RDF 1.1 Concepts and Abstract Syntax, Technical Report, W3C, 2014. URL: http://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/.
[20] D. Beckett, T. Berners-Lee, E. Prud'hommeaux, G. Carothers, RDF 1.1 Turtle, Technical Report, W3C, 2014. URL: http://www.w3.org/TR/2014/REC-turtle-20140225/.
[21] RDF Schema 1.1, Technical Report, W3C, 2014. URL: http://www.w3.org/TR/2014/REC-rdf-schema-20140225/.
[22] RDF 1.1 Primer, Technical Report, W3C, 2014. URL: http://www.w3.org/TR/2014/NOTE-rdf11-primer-20140624/.
[23] S. Prediger, Logical scaling in formal concept analysis, in: D. Lukose, H. S. Delugach, M. Keeler, L. Searle, J. F. Sowa (Eds.), Proceedings of ICCS 1997, volume 1257 of *LNAI*, Springer, 1997, pp. 332–341.
[24] SPARQL 1.1 Query Language, Technical Report, W3C, 2013. URL: http://www.w3.org/TR/2013/REC-sparql11-query-20130321/.
[25] J. Kötters, P. W. Eklund, Granada: Relational database navigation and scaling, in: D. Cristea, F. L. Ber, R. Missaoui, L. Kwuida, B. Sertkaya (Eds.), Supplementary Proceedings of ICFCA 2019, volume 2378 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2019, pp. 76–81.