Towards Scalingless Generation of Formal Contexts from an Ontology in a Triple Store

Frithjof Dau
SAP AG, Germany

ABSTRACT.

The EU-funded research project CUBIST investigates how Formal Concept Analysis can be applied as a Visual Analytics tool on top of information stored in a Triple Store (TS). This paper provides first steps for utilizing SPARQL in order to generate formal contexts out of the data in the TS, where the emphasis is put on using object-properties between individuals. Thus is complements FcaBedrock, which will be used in CUBIST as well and focuses on the scaling of datatype-properties between individuals and literals. It is discussed how the approaches of this paper and FcaBedrock can be combined.

1 INTRODUCTION

The EU funded research project CUBIST[1] targets new approaches to Business Intelligence (BI) by combining essential features of Semantic Technologies, Business Intelligence and Visual Analytics based on FCA (Formal Concept Analysis).

The Visual Analytics part of CUBIST is complementing traditional BI-means by utilizing FCA for analyzing the data in the triple store. FCA is a well-known theory of data analysis which allows to conceptually clustering objects with respect to a given set of attributes and then visualize the (lattice-ordered) set of clusters, e.g. by means of Hasse-diagrams. The starting point of FCA is a *formal context (O,A,I)* consisting of a set O of formal objects, a set A of formal attributes, and an incidence-relation $M \subseteq O \times A$ between the formal objects and attributes.

[1] www.cubist-project.eu

There exists a variety of FCA-tools[2], but nearly all of them take a formal context as input. Real data to be analyzed, however, often comes in different forms:

- conceptually, often attributes are not binary, but have values like numbers, strings, or dates (e.g. we have *many-valued attributes*)
- technically, data can come in form of csv-files, databases, triple stores, etc

For dealing with many-valued attributes, the most-used method is *conceptual scaling* [1]. Essentially, for a given-many valued attribute, a conceptual scale is a specific context with the values of the many-valued attribute as objects. The choice of the attributes of the scale is a question of the design of the scale: The attributes are meaningful attributes to describe the values; they might be different entities or they might even be the values of the property again. Using a conceptual scale, a dataset with a many-valued attribute can be "translated" into a formal context, where the objects are the objects of the dataset and the attributes are the attributes of the conceptual scale.

To the author's knowledge, there are essentially two tools which allow for scaling real datasets:

- ToscanaJ [2] is a suite of tools which allows to creating conceptual scales out of data from a relational database and then interactively visualizing and exploring the generated concept lattices
- FcaBedrock [3,4] is a tool which converts csv-files into formal contexts. It is "taking each many-valued attribute and converting it into as many Boolean attributes as it has values and converting continuous values using ranges." [4]

The approach of ToscanaJ is a two-step approach: First, in the design phase of a conceptual information system, the conceptual scales are created by an FCA-experienced designer. In the run-time phase, these scales are used by the user to explore the data. The downside of this approach is that the scales are predefined in the design-phase, which in turn implies that the lattice-structure of the scales is fixed, thus ToscanaJ does not really allow new *structural* insights into the data to be obtained. In the beginning of CUBIST, a modified version of ToscanaJ, called ToscanaJTS ("TS" for "Triple Store") has been developed which acts on a triple store instead of a database [5,6]. ToscanaJTS shows the

[2] See http://www.upriss.org.uk/fca/fcasoftware.html for a maintained list of FCA-software

applicability of FCA on top of a triple store, but it inherits the above discussed downsides of ToscanaJ as well.

The approach of FcaBedrock slightly differs from ToscanaJ. Similar to ToscanaJ, there is information needed on how to convert real data into a formal context. As stated in [4] "FcaBedrock solves these problems by documenting data conversions in re-usable, editable, metadata-files called bedrock files." The difference to ToscanaJ is that the formal attributes of the generated formal context are not manually defined during the design phase, but they are created on the fly from the real data and the meta-information in the bedrock files. In this respect, FcaBedrock better serves the purpose to generate formal contexts out of the real data on the fly. For CUBIST, though, there are the following disadvantages: First, FcaBedrock puts a focus on many-valued attributes, but –as it will be discussed in the next sections- there are indeed possibilities to obtain binary relations directly (without scaling) out of the information in a triple store. Secondly, the existing version of FcaBedrock needs csv-files as input. This disadvantage, though, is targeted: A new version of FcaBedrock which acts directly on a triple store is currently developed within CUBIST.

Data in a triple store modeled with RDFS is essentially structured as follows:

- First of all, we have *individuals* in a triple store. Individuals are instances of RDFS-types, which are hierarchically ordered classes
- There are binary relationships called *object properties* between individuals. Similarly to types, these properties can be hierarchically ordered.
- Finally, we can assign values (strings, numbers, dates, etc) to individuals by means of *datatype properties.*

Having said this, a triplestore-enabled version of FcaBedrock can essentially deal with the conversion of datatype properties into formal contexts. On the other hand, as object properties are binary relations between indivuals, they give naturally rise to formal contexts, where the domain of an object property can serve as the set of formal objects and the range as the set of formal attributes. This first idea is too narrow. This paper discusses first steps on how to utilize object properties in different ways in order to generate formal contexts between resources in a triple store. As this paper deals with utilizing object properties for FCA, and as FcaBedrock deals with utilizing datatype proper-

ties, the ideas presented in the paper are complementing the ideas which underly the creation of context in FcaBedrock. It is planned to combine the approach of FcaBedrock and the ideas in this paper to provide in CUBIST a full-fledged approach to generate formal contexts out of the data (individuals, object properties, datatype properties) in a triple store.

2 PREREQUISITES

We will exemplify most of the ideas with the data from the HWU use case [7] in CUBIST. In order to understand the examples, the use case and the underlying ontology [8,9] shall be briefly introduced.

The HWU use case deals with data about gene expressions in mouse embryos. The core information are triples of the form (gene, tissue, level of expression), where:

- A *gene* is a unit of instruction providing directions for tasks in the development of a mouse, e.g. the creation of a protein.
- A *tissue* is a anatomical part of a mouse embryo. Tissues are ordered via a part-of-relation. Moreover, each tissue is uniquely assigned to a *Theiler Stage*, being a "time-slot" in the development of a mouse.
- The *level of expression* or *strength* states whether a gene is expressed in a tissue, or whether it is known that it is not expressed (this can even be more fine-grained described as weakly, moderate or strongly expressed), or whether it is unknown whether the gene is expressed or not (this can even be more fine-grained described as "not examined" or "possible).

Such a triple is called *textual annotation* and concluded from some *experiments*. Each experiment consists of one or more textual annotations.

The data of the HWU use case has been converted to an RDFS-ontology and stored in a triple store (OWLIM[3]). The schema of the HWU-ontology is provided in Fig. 1.

In order to create formal contexts out of the HWU-data in the triple store, we have developed a small tool which takes a SPARQL-query as input and converts the result of the SPARQL-query into a context. Essentially, the tool works as follows: The result of a SPARQL-query is a table, where the columns correspond to the query variables. It is possi-

[3] http://www.ontotext.com/owlim

ble that cells in the table are not filled (this can happen if the OPTIONAL-clause of SPARQL is used). The names of the query variables determine whether the variable is used to generate a formal object or a formal attribute: Variables starting with “o” generate objects, variables starting with “a” generate attributes, and all other variables have no impact on the generated context. If more than one variable starts with “o”, then the result for these variables are simply concatened (with a divider ‘--‘) to generate an object name. The case of more than one variable starting with “a” is handled in a similar manner. Finally, if we have a row in the SPARQL-result which generates both an object and an attribute, a cross in the corresponding incidence relation is set. An (artificial) example for this algorithm is provided in Fig. 2.

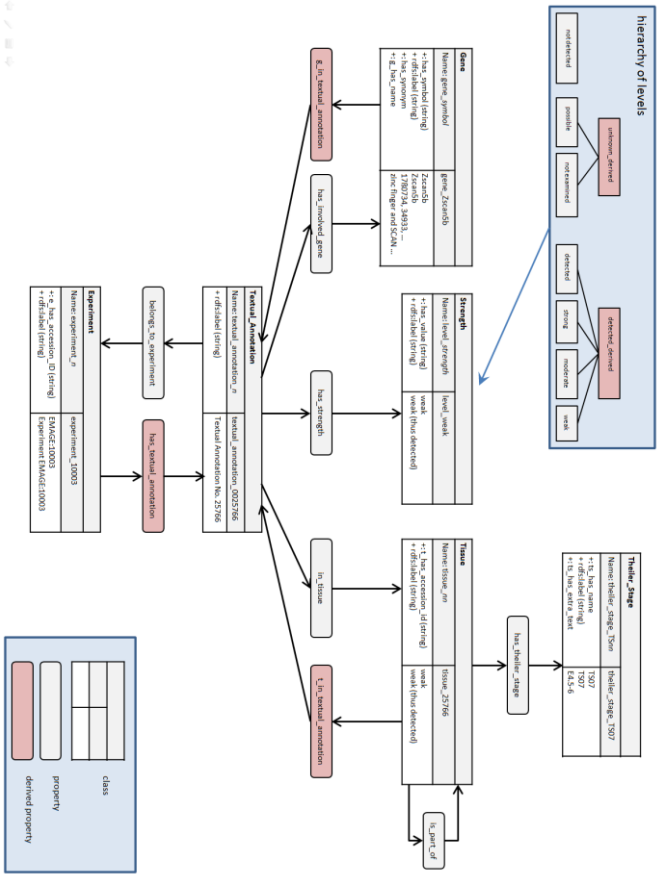


Fig. 1. The ontology for the HWU use case in CUBIST

The result of a SPARQL-query

Obj1	Obj2	Att1	Att2
O1			
O1	O2		
	O3		
		A1	
		A1	A2
			A2
O4		A3	
O5	O6	A4	
O5	O7	A5	A6
O5	O7	A4	

The generated formal context

	A1	A1—A2	A2	A3	A4	A5—A6
O1						
O1—O2						
O3						
O4				X		
O5—O6					X	
O5—O7					X	X

Fig. 2. From SPARL-query-results to formal contexts

Before we come to the next section, let us finally state two general assumptions:

1. URIs are the unique identifiers for resources, but they might be too clumsy to be used as names for the resources. We will use labels instead, thus we assume that each resource in the triple store is appropriately labeled using rdfs:label, and we more over assume that different entities have different labels.
2. Secondly, as SPARQL is agnostic to inferencing, we naturally assume that the information in the triple store is closed under RDFS-entailment.

Both assumptions hold in the given HWU-dataset.

3 SPARQRL-QUERIES

3.1 Simple Pattern: Linking entities with a chain of properties

The very essence of a formal context is the incidence relation, being a binary relation between objects and attributes. RDF-properties in turn are binary relations as well (between RDF-resources). Thus any RDF-property *linkingproperty* already gives rise to a formal context, via the following SPARQL-query:

Most basic SPARQL-query for generating a formal context

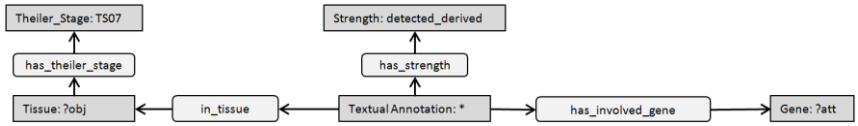
```
SELECT DISTINCT ?obj ?att WHERE {
?objRessource rdfs:label ?obj .
?attRessource rdfs:label ?att .
?objRessource :linkingproperty ?attRessource . }
```

This pattern is anyhow too simple as a general pattern for generating contexts, and we have to extend it. For the following considerations, let

us assume we want to investigate the tissues of Theiler stage 07 and which genes are detected in those tissues.

1. First of all, as RDF is graph-based, we can have in RDF chains of properties between resources. This particularly applies when we have reified relations. For our example investigation, note that there is no direct property linking tissues and genes: Instead, we have textual annotations linking them. So we have to consider the following chain of properties:[4]

2. In the following, resources which are not directly queried, like the (unknown) textual annotations, will be called "intermediate resources".
3. The property `in_tissue` in the example above moreover shows that in the chain of properties, some properties might be traversed in their opposite direction.
4. It is sensible to assume that all queried resources as well as intermediate resources are retrieved from some RDF-type. In our example, we have Tissue and Gene for the queried resources and Textual_Annotation for the intermediate resource.
5. Finally, we might further restrict the set of objects, or the set of attributes, by some constraints. In our example, we are interested in investigating a specific Theiler stage and thus instead of taking all tissues into account, we restrict ourselves only to those tissues from that Theiler stage. Similarly, we might impose constraints on the intermediate resources as well. In our example, we are only interested into combinations of tissues and genes where the gene is detected (maybe weakly, moderate or strongly) in the tissue, but we are not interested into combinations where the gene is not detected. Thus we have the following additional constraints:



We are now prepared write a SPARQL-query which generates the wished context.

[4] We use the notion of conceptual graphs [10] and query graphs [11] to diagrammatically depict the queries.

SPARQL Query 1	Explanation
SELECT DISTINCT ?obj ?att WHERE {	Select Clause for objects and attributes
?x1 rdf:type :Tissue ; rdfs:label ?obj .	Type of objects (see 3))
?x1 :has_theiler_stage :theiler_stage_TS07 .	Additional Constraint for objects (see 4))
?x3 rdf:type :Gene ; rdfs:label ?att .	Type of attributes (see 3))
?x2 rdf:type :Textual_Annotation.	Type of intermediate resource (see 3))
?x2 :in_tissue ?x1 .	1st and 2ndproperty in the chain of properties (see 1),
?x2 :has_involved_gene ?x3 .	first prop. is traversed in opposite direction. (see 2))
?x2 :has_strength :level_detected_derived .}	Add. constraint for intermediate ressources (see 4))

It should be noted that this SPARQL-query does not retrieve **all** tissues of Theiler Stage 07: Instead, only those tissues are retrieved where a gene is detected. Let us call such a query "object-restricted". Vice versa, not all genes are retrieved, but only those who are detected in some tissue of Theiler stage 07. Let us call those queries "attribute-restricted". In other words: In the formal context, we have by definition neither empty rows nor empty columns. One might want to change this to object-unrestricted queries, i.e. retrieving all tissues of Theiler stage 07, allowing empty rows, and/or attribute-unrestricted queries, i.e. retrieving all genes, allowing empty columns. So we have four variants of query 1 to consider.

As there are nearly 7000 genes, a query which retrieves all of them and adds them as formal attributes does not seem sensitive. But as there are only 16 tissues in Theiler stage 07, retrieving all of them and adding them as formal objects is reasonable. So let us consider the object-unrestricted variant of query 1. There are two ways to obtain this in a SPARQL-query: Either via utilizing the OPTIONAL-clause of SPARQL, or using the UNION-operator. Both queries are given below.

SPARQL Query 1a, utilizing OPTIONAL	SPARQL Query 1b, utilizing UNION

SELECT DISTINCT ?obj ?att WHERE {
?x1 rdf:type :Tissue ; rdfs:label ?obj .
?x1 :has_theiler_stage :theiler_stage_TS07 .
OPTIONAL
{ ?x3 rdf:type :Gene ; rdfs:label ?att .
?x2 rdf:type :Textual_Annotation.
?x2 :in_tissue ?x1 .
?x2 :has_involved_gene ?x3 .
?x2 :has_strength :level_detected_derived .}
}
ORDER BY ?obj ?att
	SELECT DISTINCT ?obj ?att WHERE {
	{ ?x1 rdf:type :Tissue ; rdfs:label ?obj .
	?x1 :has_theiler_stage :theiler_stage_TS07 . }
	UNION
	{ ?x1 rdf:type :Tissue ; rdfs:label ?obj .
	?x1 :has_theiler_stage :theiler_stage_TS07 .
	?x3 rdf:type :Gene ; rdfs:label ?att .
	?x2 rdf:type :Textual_Annotation.
	?x2 :in_tissue ?x1 .
	?x2 :has_involved_gene ?x3 .
	?x2 :has_strength :level_detected_derived . }
	}
	ORDER BY ?obj ?att

Beginning of resultset	**Beginning of resultset**
obj att --- EMAP:25772 EMAP:42 Etv5 EMAP:42 Smad2 …	obj att --- EMAP:25772 EMAP:42 EMAP:42 Etv5 EMAP:42 Smad2 …

Table 1. SPARQL-query for gene-tissue combinations of TS 07

From an RDF-point of view, these queries are semantically (slightly) different: In query 1a, we have a row in the resultset with a tissue *t* and

without a gene if and only if the gene belongs to Theiler stage 07 and no gene is detected in that tissue, whereas in query 1b we have a row in the resultset with a tissue *t* for any *t* belonging to Theiler stage 07. An example where the resultsets differ is tissue EMAP:42, as it can be seen in Table 1. So the resultset of query 1b is a superset of the resultset of query 1a. Anyhow, the formal contexts generated with queries 1a and 1b are indeed the same, thus from an FCA-perspective, the queries are equivalent.

Please note moreover that in query 1b, the clause querying the tissues is repeated in the UNION clause, whereas a repetition of the clause is not. This renders query 1b (slightly) more complicated.

Having these two differences in mind, one can conclude that query 1a has to be preferred over query 1b.

The patterns of both queries can easily be transferred to the case of attribute-unrestricted queries. For queries which are both object- and attribute-restricted, only the UNION-variant can be easily extended.

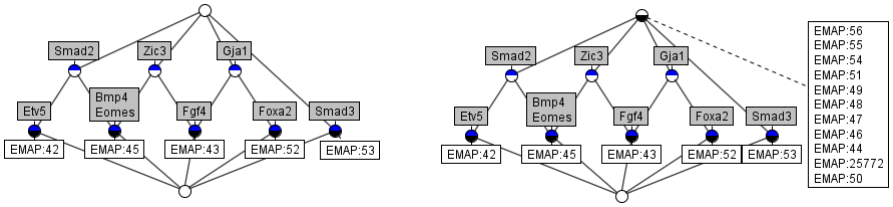


Fig. 3. Concept lattices retrieved from query 1 (left), and query 1a or query 1b (right)

3.2 Combining different variables to objects or attributes

In the previous section, we have investigated for Theiler stage 07 tissue-gene combinations such that the gene is detected in the respective tissue. There are anyhow different levels of being detected: weak, moderate, and strong. Moreover, in some experiments a gene is detected without information on how strong the expression of the gene in that tissue is. So we have four kinds of being detected, called “weak”, “moderate”, “strong” and “detected”. All of them are subsumed by an artificial strength called “detected derived”. See Fig. 1, where these levels are depicted. The information on the level of expression is not provided by the SPARQL-queries of the last section, but it can easily be added by slightly altering the queries. We adopt query 1b as following, highlighting the changes in the query:

SPARQL query 1b with level of expression added

```
SELECT DISTINCT ?obj ?att1 ?att2  WHERE {
?x1 rdf:type :Tissue ; rdfs:label ?obj .
?x1 :has_theiler_stage :theiler_stage_TS07 .
OPTIONAL
{     ?x3 rdf:type :Gene ; rdfs:label ?att1 .
      ?x2 rdf:type :Textual_Annotation.
      ?x2 :in_tissue ?x1 .
      ?x2 :has_involved_gene ?x3 .
      ?x2 :has_strength :level_detected_derived .
      ?x2 :has_strength ?x4 .
      ?x4 rdf:type :Strength ; rdfs:label ?att2 . }
}
ORDER BY ?obj ?att1 ?att2
```

In the next figure, the resulting concept lattice is provided.

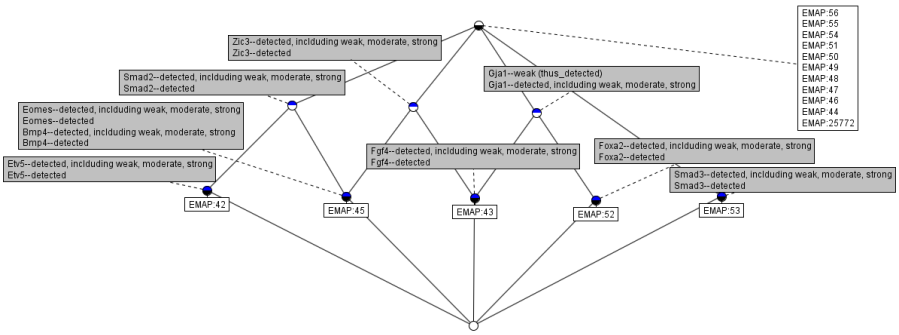


Fig. 4. Lattice of altered query 1b with information about strengths

The lattices of Fig.3 and Fig. 4 have the same structure: Essentially, only the information in the attribute labels are more fine-grained. In other cases, combining more result variables to attributes might yield in valuable, new structural insights. To provide an HWU-example for this effect, we consider the following query to retrieve contradicting textual annotations, which was possible in the previous version of the tool:

SPARQL query 2 for finding contradictions

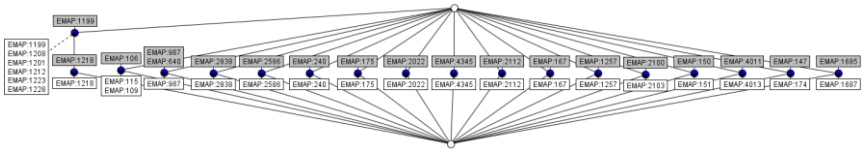
```
select distinct ?o0 ?a0   where {
  ?x0 rdf:type :Tissue ; rdfs:label ?o0 .
  ?x1 rdf:type :Tissue ; rdfs:label ?a0 .
  ?x2 rdf:type :Gene ;   rdfs:label ?o1 .
  ?ta1 :in_tissue ?x0; :has_involved_gene ?x2; :has_strength :level_detected_derived .
  ?ta2 :in_tissue ?x1; :has_involved_gene ?x2 ; :has_strength :level_not_detected .
  {
  { ?x0 :is_part_of ?x1 .  Filter(!sameTerm(?x1,?x0)) }
  UNION
  {  Filter(sameTerm(?x0,?x1)  )  }   }}
```

This query retrieves pairs of tissues t_1 and t_2, where

- t1 is_part_of or the same tissue as t2 (that is, we are using propagation of tissues), and

- there exists a gene g which is (possibly weakly, moderate, or strong) detected in t_1 and not detected in t_2

Of course, if a gene g is expressed in a t_1, then one can conclude that it is expressed in t_2 as well. That is, the query finds out pairs of tissues (t_1, t_2) where different experiments concerning the gene g come to contradicting results. As we will discuss the example further, we fix the following notation describing the roles of the tissues: The tissue t_1 will be called lower or detected tissue, and t_2 will be called upper or undetected tissue. Next, the concept lattice generated by query 2 is provided.

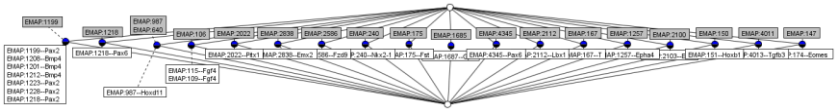


Most of the lattice does not provide, apart from the tissue-names, any structural information. On the left hand side, however, this lattice does reveal some insights:

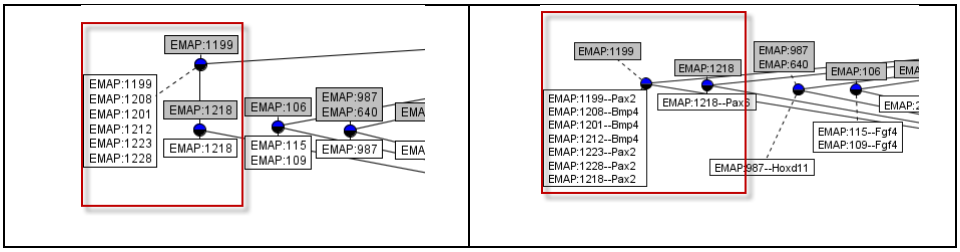
- There are two contradicting tissue pairs with EMAP:106 as upper (undetected) tissue, as we have the two lower tissues EMAP:109 and EMAP:115. We don't know anyhow how many pairs of textual annotations cause these contradictions: it must be at least two pairs, but for example, we might have a contradiction for many genes, detected in EMAP:109 and not detected in EMAP:106. The same concern applies to any node in the lattice: We never know how many contradicting pairs of textual annotations we have for one node.
- Similarly, we have two contradicting tissue pairs with EMAP:987 as lower (detected) tissue.
- The left hand side is most interesting: There is a number of contradicting tissue pairs with EMAP:1199 as upper tissue, namely 7 (6+1). Maybe the experiment(s) investigating EMAP:1199 deserve a closer look? We have moreover a dependency: Whenever a lower (detected) tissue contradicts with EMAP:1218 as upper (nondetected) tissue, it contradicts with EMAP:1199 as well (but not vice versa).

Now, for this query, we lose the information which genes cause the contradictions. Knowing about the involved genes would allow to partly cope with the questions raised above. We do only a slight change in

the query by adding the variable `?o1` to the list of variables in the select-clause, i.e. we reuse the last query and change it to query 2b by starting with "`select distinct ?o0 ?a0 ?o1 where {`". Below, the corresponding lattice is provided.



In the last discussion, we speculated that "we might have a contradiction for many genes, detected in EMAP:109 and not detected in EMAP:106." But now we see this contradiction is only causedwe by one gene (Fgf4), and same holds true for all nodes: all contradicting tissue pairs are caused by **exactly one** gene. Moreover, the lattices are not isomorphic: The difference is highlighted in the next screenshots:



Note that the contradiction between EMAP:1218 and EMAP:1119 as upper tissues and EMAP:1218 as lower tissue are caused by *different*, thus the attribute dependency between EMAP:1218 and EMAP:1119 on the left hand side is lost on the right hand side. It seems even more that EMAP:1119 deserves a closer observation.

3.3 Attributes of different types

So far, in the queries we have provided we have as objects or attributes either entities of **one** RDF-type, or combinations (via string-concatenation) of different types into one object or attribute. In some cases, though, it can be desirable to have objects or attributes of different types. This shall be exemplified with query 2 where we have analyzed contradiction pairs of tissues. These tissues in turn are assigned to Theiler stages. In the last query, we used

```
?x1 rdf:type :Tissue ; rdfs:label ?a0 .
```

to query the tissues. If we replace this line by

```
?x1 rdf:type :Tissue         .
?x1 :has_theiler_stage ?ts1 .
?ts1 rdfs:label ?a0          .
```

We obtain the Theiler stages instead. Now, utilizing the SPARQL-"UNION"-operator, it is possible to "combine" these slightly different queries, resulting in a formal context where the attributes are either tissues or Theiler stages.

The result lattice extends the lattice for query 2 by adding Theiler stages. It looks as follows:

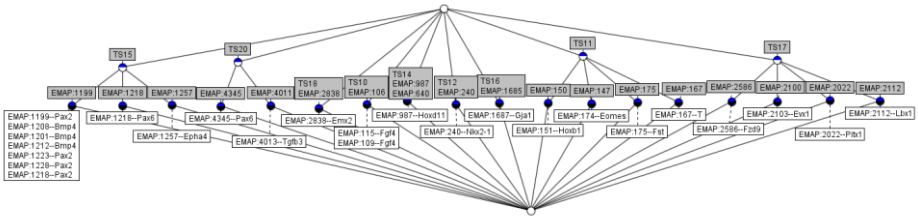


Fig. 5. Concept lattice for contradicting pairs of tissues

The nice lattice structure is unsurprising: It is caused by the fact that each tissue is assigned exactly one Theiler stage (and the part_of-relation only applies to tissues in the same stage).

The approach hereby exemplified it particularly helpful if attributes (here: tissues) are in turn classified by other attributes (here: Theiler stages). For this reason, it can easily be transferred to RDFS-instances and their corresponding types, thus utilizing the type-hierarchy in an RDFS-ontology. We have not exemplified this approach in this paper as the underlying ontology does not provide interesting type hierarchies.

4 SUMMARY AND NEXT STEPS

In the previous section, we have discussed how object properties between individuals in a triple store can be utilized for generating formal contexts. We have seen that only utilizing plain object properties between individuals is not sufficient: Instead, one should consider chains of object properties with constraints for intermediate nodes, and one should consider different means for adding formal attributes generated from *different* RDFS-types to the formal context. This is anyhow only a

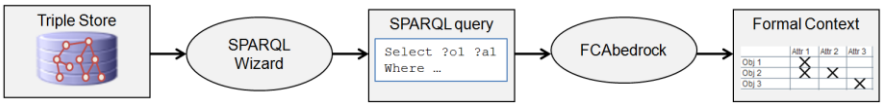
first step for generating formal contexts in CUBIST. There are essentially two tasks to be conducted:

1. First, it cannot be expected from a CUBIST user to write SPARQL-queries on her own. For this reason, we have to find common patterns for the generation of formal contexts of SPARQL-queries. Such patterns can be used in a wizard which guides the user in the creation of formal contexts without showing any SPARQL-queries.
2. Second, as already mentioned, the ideas presented in this paper complement the approach of FcaBedrock, thus it has to be investigated how these approaches can be combined.

A closer look at our approach reveals that there is no need that the SPARQL-queries, as presented in the paper, only return (the labels of) RDFS-individuals: Of course, one can extend the approach to queries where some of the variables return literals, e.g. strings or numbers. In our approach, for a given query-variable, each literal would be taken as it is for generating a formal attribute, which would be in most cases not desirable. Instead, for such variables, the process of conceptual scaling –as carried out by FcaBedrock- should apply. Indeed, as any SPARQL-query returns a table, it is straight-forward to feed such a table into FcaBedrock. Having said this, a possible workflow for the generation of contexts is as follows:

1. A user selects the type of individuals she wants to investigate.
2. A wizard guides the user in the creation of a SPARQL-query. For example, the wizard could provide properties or chains of properties (starting with the given RDFS-type), and the user can select which chain(s) should be used in the generation of the context. Moreover, it could be possible that in this step, the user adds additional constraints on intermediate nodes in the chain.
3. For each (chain of) properties selected by the user, the range consists either of individuals or of literals of a given type. In the former case, the labels of the individuals are used for the generation of attributes, whereas in the latter case, the values are transformed into formal contexts with the help of FcaBedrock.

That is, generally speaking, the meta-information which generated a formal context out of the triple store consists of a SPARQL-query and, for each query variable, instructions on how the results of the variable are used to generate objects and attributes. The whole process can be depicted as follows:



In the further course of CUBIST, the consortium will investigate and implement a unified approach for the generation of formal context which takes individuals, object properties and datatype properties into account, leading to a fully-fledged generation of context out of the triple store on the fly.

Acknowledgement

This work is part of the CUBIST project ("Combining and Uniting Business Intelligence with Semantic Technologies"), funded by the European Commission's 7th Framework Programme of ICT, under topic 4.3:Intelligent Information Management.

The herein used data orginates data from the Edinburgh Mouse Atlas of Gene Expression (EMAGE) and is released under a creative commons license.

5 REFERENCES

1. Ganter, B., Wille, R. (1989). Conceptual Scaling. In: Roberts, F. (ed.) *Applications of Combinatorics and Graph Theory to the Biological and Social Sciences*. IMA, vol. 17, 139–168. Springer, Heidelberg
2. Becker, P., Correia, J.H. (2005). The ToscanaJ Suite for Implementing Conceptual Information Systems. In: Ganter, B., Stumme, G., Wille, R. (eds.) *Formal Concept Analysis*. LNCS (LNAI), vol. 3626, 324–348. Springer, Heidelberg
3. Andrews, S. (2009). Data Conversion and Interoperability for FCA. In: CS-TIW 2009, 42–49, http://www.kde.cs.uni-kassel.de/ws/cs-tiw2009/proceedings_final_15July.pdf
4. Andrews, S., Orphanides, C. (2010). FcaBedrock, a Formal Context Creator. In: Croitoru,M., Ferre, S. and Lukose, D. (eds.): *Proceedings of ICCS 2010*, Kuching, Malaysia. LNAI 6208, Springer-Verlag
5. Dau, F., Sertkaya, B. (2011). Formal Concept Analysis for Qualitative Data Analysis over Triple Stores. In: Olga De Troyer, Claudia

Bauzer Medeiros, Roland Billen, Pierre Hallot, Alkis Simitsis and Hans Van Mingroot (eds): *Advances in Conceptual Modeling. Recent Developments and New Directions - ER 2011 Workshops FP-UML, MoRE-BI, Onto-CoM, SeCoGIS, Variability@ER, WISM*. Springer, LNCS, vol 6999.

6. Dau, F., Sertkaya, B. (2011). An Extension of ToscanaJ for FCA-based Data Analysis over Triple Stores. In F. Dau (ed): *Proceedings of the 1st CUBIST (Combining and Uniting Business Intelligence with Semantic Technologies) Workshop 2011*, CEUR proceedings, vol 753.
7. Andrews, S., M^{c}Leod, K. (2011). Gene Co-Expression in Mouse Embryo Tissues. In F. Dau (ed): *Proceedings of the 1st CUBIST (Combining and Uniting Business Intelligence with Semantic Technologies) Workshop 2011*, CEUR proceedings, vol 753.
8. M^{c}Leod, K., Ferguson, G., Burger, A. (2019). Argudas: arguing with gene expression information. In: Paschke, A., Burger, A., Splendiani, A., Marshall, M.S., Romano, P. (eds.) *Proceedings of the 3rd International Workshop on Semantic Web Applications and Tools for the Life Sciences* (December 2010)
9. Richardson, L., Venkataraman, S., Stevenson, P., Yang, Y., Burton, N., Rao, J., Fisher, M., Baldock, R.A., Davidson, D.R., Christiansen, J.H. (2010). EMAGE mouse embryo spatial gene expression database: 2010 update. Nucleic Acids Research 38, Database issue, D703–D709
10. Sowa, J. F. (1984). Conceptual Structures: Information Processing in Mind and Machine. Addison Wesley Publishing Company Reading, 1984.
11. F. Dau, F (2004). Query Graphs with Cuts: Mathematical Foundations. In A. Blackwell, K. Marriott, A. Shimojima (Eds): *Diagrammatic Representation and Inference*. LNAI 2980, Springer Verlag, Berlin–New York 2004, 32-50.