A Database Browser based on Pattern Concepts

Jens Kötters and Heinz W. Schmidt†

†Computer Science & IT, RMIT University, Melbourne, Australia
(Heinz.Schmidt@rmit.edu.au)

Abstract. A Galois connection is stated between a knowledge base and queries over this knowledge. Queries are stated as conjunctions. Both the knowledge and queries are represented by certain graphs. This Galois connection gives rise to lattices of pattern concepts implicitly contained in the theory (all derivable facts) over the knowledge base.
The formal foundation for browsing such lattices and a realisation in terms of a prototype tool is outlined. Data types may be assigned to individual columns of tables in the database. A type assignment corresponds to an extension of the query language and incorporates additional knowledge into the process of concept creation. Type and derivation support in the tool may be provided by pluggable modules. In the examples in this paper, only the numeric type and concrete, stored relations are featured.

Keywords: Database Browsing, Pattern Concepts, Formal Concept Analysis, Knowledge Representation, Many-Sorted Logics

1 Introduction

The paper presents a prototype of a FCA browser for knowledge bases and its formal foundation. The browser allows interactive access to the content of a database. Via a command-line interface, concept lattices over relational data can be traversed. Each concept intent corresponds to a logical formula (or query) in one or more free variables, using relational expressions over function terms with variables where the functions range over primitive and user-defined data sorts.

Each extent is the corresponding table of results. There is one concept lattice for each set of free variables; the user can cross over into different lattices during navigation (thus changing variables in the result set). We will see that each concept lattice arises from a suitably defined pattern structure [5] (stretching the definition a bit), and pattern concepts have indeed been considered for the representation of logical formulas [5, p.129]. Further references and details of the approach can be found in [10], although many-sorted logic has not been considered there.

The formalisation of FCA navigation includes concrete (stored) and abstract (computed) relations derived by domain-specific conditional logical rules and/or relational algebra operations (SQL). Domain knowledge rules need to capture

– derived relations (e.g. computing relatives based on a network of parent-child relationships);

– domain-specific interpretations of object attributes, incl. common taxonomies and discretizations (e.g. age range of legal childhood, adulthood, retirement in social insurance databases etc.)
– representation invariants abstracting from syntactic and computational details of the representation incl. relational algebra, and independent of the specific database platform. While representations are typically realised imperatively, occassionally such invariants are required to manipulate and transform queries or tables for the purpose of navigation.

We call these rules *abstract*, in particular, because they are independent of specific sets of concrete relation tables, and hence remain invariant across different concrete databases for the given domain and also across updates of the same database. Although our browser prototype does not include an inference engine, such a module can be interfaced easily by storing the results of external reasoning steps as special tables accessible to the browser, on-the-fly. Here we focus on the connection with FCA lattices.

For the purpose of this paper, we interpret many-sorted logics in an algebraic-categorical framework – a view that has gained wide acceptance in the semantics of programming languages, abstract data types, knowledge representation and behaviour specification over several decades. It goes back to universal algebra [7] and work on formal specification and abstraction since the seventies (cf. e.g. [3, 4, 1]). In the interest of readability of the paper to a broad FCA audience, we limit ourselves to an overview and introduce notation only where necessary to be able to follow the core examples and algorithms presented in the paper. A complete formal exposition is beyond the scope of this paper and this conference.

The paper is organised as follows. In Section 2 we review relevant existing work on many-sorted structures and logics and summarise our notation; section 3 presents some technical advanced many-sorted structures that form the basis for our FCA-centric approach to patterns and queries; section 4 focuses on the browser, both in terms of the core algorithm and the user interface for navigation. Finally Section 5 provides some links to related work.

2 Many-Sorted Structures and Logics

In this section we briefly summarise basic notations and formalisation used in the rest of the paper. The algebraic-categorical view of *abstract data types* and data analysis has developed in line with model theory: syntax is captured in signatures limiting the construction of well-sorted terms and atomic formulae over algebras (data and functions) or structures (algebras plus relations) as models. Terms are sorted to represent data of primitive sorts abstractly, independent of a specific interpretation by a data domain. For example attributes, arithmetic or logical operators appearing in logical formulae or database queries may be sorted, as in the example below, where `Anne` is a constant of sort `person`, `age` is a numeric attribute of an `object` and `Parent` is a binary predicate on sort `person`. For flexible abstract many-sorted definitions we permit so-called order-sorted models, i.e., where sorts are partially ordered. **bool** and **int** are assumed to be built in primitive sorts. **object** is a built-in maximal sort.

```
sort int < number, person < object
Anne,Bob,Chris,Dora,Emily:  → person
_+_: number × number → number
_<_: number × number → bool
age: object → int
Parent: person × person
female,male: person
```

As we will see later, the sort order abstracts from a corresponding subset relationship between corresponding data domains. We also allow so-called 'mixfix' notation for function and predicate symbols as known from platforms realising algebraic-categorical forms of many-sorted type or logical specifications, such as OBJ3 [6], CASL [1], and ELAN [2, 9]. For instance, _+_ indicates the two argument positions for this binary infix operator '+'. **Signatures**. Formally,

Parent

c0	c1
Anne	Bob
Anne	Chris
Bob	Dora
Bob	Emily

age

c0	c1
Anne	59
Bob	31
Chris	27
Dora	7
Emily	3

META

table	column	type
Parent	c0	person
Parent	c1	person
male	c0	person
female	c0	person
age	c0	person
age	c1	number

female

c0
Anne
Dora
Emily

male

c0
Bob
Chris

Fig. 1. Database

a many-sorted signature is a triple $\Sigma = (S, F, P)$ where S is a finite partial order (of elements called *sort symbols* or sorts for short), $F = (F_{u,s})_{u \in S^*, s \in S}$ is a pairwise disjoint family of sets of symbols (called *function symbols*) and $P = (P_u)_{u \in S^*}$ is a family of pairwise disjoint sets of symbol (called *predicate symbols*). For $f \in F_{u,s}$ (or $p \in P_u$) we set $dom(f) = u$ (or $dom(p) = u$, respectively) and $cod(f) = s$ (read *'domain'* and *'codomain'* respectively). As usual for abstract types and many-sorted logics, signature morphism remap sorts, function and predicate symbols preserving domains, codomains and sort order. We use $T_{\Sigma,s}$ to denote the set of well-formed terms of sort s and A_Σ the set of well-formed atoms $p(t_1, \ldots, t_n)$ for $p \in P_u, u = s_1 \cdots s_n, t_i \in T_{\Sigma,s_i} (1 \leq i \leq n)$.

Elimination of junk. Many-sorted approaches are interesting to us as they reduce the search space for inferencing and navigation: ill-sorted terms and formulae can be recognised efficiently, and in fact eliminated syntactically. This reduces the search space significantly and eliminates massive amounts of so-called 'junk data' in terms of ill-sorted elements, in particular in queries and auxiliary formulae occuring in searches.

Example database. Before we formalise concrete models, let us look at a example database as a concrete model in terms of sets and tables.

A database table corresponds to (a) a relation interpreting a corresponding predicate symbol, or (b) a function from a set of columns (arguments) to a column (result), or (c) a set of attributes mapping the rows (object keys) to attributes, thus encoding functions similar not unlike (b). For example in Fig. 1, the table named `Parent` contains all pairs (p, p') such that $Parent(p, p')$ is valid, while the table `age` maps persons to their ages. Column are representing attributes including selectors of components in tuples. For example $c0$ in the `age` table selects the `person` component of the table rows etc. Queries supported by the database are constrained by the signature and the logical connectives permitted in the structure of formulae outside the algebraic structure captured by the signatures. For concrete databases and in our prototype implementation, we assume the existence of a META table (see Fig. 1) which represents the signature information relevant for the tables in the database. The remaining signature (outside the data base) represents operators on data types in the tables or relations that can be computed from the data base using queries.

Many-sorted structures. Given a many-sorted signature Σ, a Σ-structure $\mathbf{D} = \langle (D_s)_{s\in S}; \mathcal{F}, \mathcal{R}\rangle$ has a family of carrier sets D_s (aka domains) sorted and ordered by S and families of sets of functions and relations compatible with the prescribed domains and codomains of functions and predicate symbols in Σ. The partial order of sorts is interpreted as subsorting: $D_s \subseteq D_t$ if $s \leq t$. Functions are total on their domains.[1] For readability in concrete examples we also denote D_s by s_D (the interpretation of sort s in $\mathbf{D}$). Likewise, for $f \in F_{u,s}$ ($p \in P_u$) we denote the corresponding function in $\mathbf{D}$ by f_D (or p_D respectively). It is well-known that the term structure $\mathbf{T}_\Sigma := \langle (T_s)_{s\in S}; F, P\rangle$ with empty relations forms the free Σ-structure. Homomorphisms between Σ-structures are *weak*, i.e. preserve definedness but not necessarily undefinedness of relations. They are called *strong* if they also preserve undefinedness.

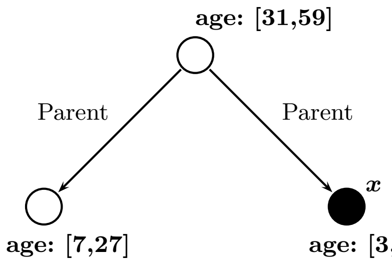
The formal structure underlying the database in Fig. 1 has for example:

$$\begin{aligned}
\texttt{person}_D &= \{\text{Anne}, \text{Bob}, \text{Chris}, \text{Dora}, \text{Emily}\},\\
\texttt{Anne}_D &= \text{Anne}, \ldots, \texttt{Emily}_D = \text{Emily},\\
\texttt{number}_D &= \{3, 7, 27, 31, 59, ...\},\\
\texttt{Parent}_D &= \{(\text{Anne}, \text{Bob}), (\text{Anne}, \text{Chris}), (\text{Bob}, \text{Dora}), (\text{Bob}, \text{Emily})\},\\
\texttt{age}_D &= \{\text{Anne} \mapsto 59, \text{Bob} \mapsto 31, \text{Chris} \mapsto 27, \text{Dora} \mapsto 7, \text{Emily} \mapsto 3\}
\end{aligned}$$

Many-sorted logics. For the rest of the paper, let Σ be a fixed signature and $\mathbf{D}$ a Σ-structure. We assume each sort in Σ includes a distinguished equality predicate $=_s$ with the obvious interpretation in $\mathbf{D}$. For sorted terms and formulae

[1] Note that the underlying algebra $\mathbf{D} = \langle D; \mathcal{F}\rangle$, with the unsorted carrier D the union of the D_s, is partial, as is the underlying unsorted structure.

with variables we use a many-sorted family $(V_s)_{s \in S}$ of at most countably infinite and pairwise disjoint sets of variable symbols that are disjoint from function symbols in $\mathcal{F}$. We denote by $\Sigma(V)$ the extended signature that adds variables as constant function symbols to Σ. The $\Sigma(V) - \mathit{term}$ structure now contains all well-sorted terms with variables. Σ-formulae are built using well-sorted atoms $p(t, ...)$ for predicate symbols p in Σ, conjunction, implication and existential quantifiers over sorted variables constrained to prenex normal form (i.e. not occurring under conjunction or implication but ranging over the entire formula at hand). We denote by $Free(\phi)$ the free variables of a formula ϕ, call ϕ *closed* iff $Free(\phi) = \emptyset$, denote by At_Σ the set of all Σ-atoms, and by Cl_Σ the set of Σ-formulae. Formulae are evaluated over a Σ-structure as usual, by recursively 'translating' function symbols into function application, predicate symbols into relations in $\mathbf{D}$ and interpreting conjunction, implication and existential quantification logically. We use I_D to denote the corresponding interpretation of closed terms and fomulae and write $\mathbf{D} \models \phi$ to denote that ϕ is valid in the model $\mathbf{D}$. For terms or formulae with variables ($Free(\phi) = \{x_1, \ldots, x_n\}$) we write $I_D(\phi[a_1/x_1, \ldots, a_n/x_n])$ to denote the corresponding evaluation under the assignment of the a_i to x_i.



$$\begin{aligned}&\exists p_0 \exists p_1 \exists p_2 \exists n_0 \exists n_1 \exists n_2 :\\ &(\mathit{age}(p_0, n_0) \wedge 31 \leq n_0 \leq 59\\ &\wedge \mathit{age}(p_1, n_1) \wedge 7 \leq n_1 \leq 27\\ &\wedge \mathit{age}(p_2, n_2) \wedge 3 \leq n_2 \leq 31\\ &\wedge \mathit{Parent}(p0, p1) \wedge \mathit{Parent}(p0, p2)\\ &\wedge x =_{\text{person}} p_2)\end{aligned}$$

Fig. 2. Query graph for pattern formula

3 Patterns

For queries in particular, we are interested in formulae ψ in the following prenex normal form,

$$\exists x_1 \ldots \exists x_m : \phi \tag{1}$$

where ϕ is a conjunction, $\phi \equiv (\phi_1 \wedge \cdots \wedge \phi_l)$. We interpret ψ as the request to compute all possible consistent assignments to $Free(\psi)$ such that $\mathbf{D} \models \psi[a_1/x_1, \ldots, a_n/x_n]$. Without loss of generality, we assume that each free variable $y \in Free(\psi)$ has a single occurrence on the left-hand side of an equation $y =_s \ldots$. In the above sense, Σ-formulae of the form ψ can be regarded as *patterns* that select matches in any Σ-structure. More precisely, the set of pattern matches $P_\phi := \{(a_1, \ldots, a_n) \in D_{s_1} \times \cdots \times D_{s_n} \mid \mathbf{D} \models \psi[a_i/x_1, \ldots, a_n/x_n]\}$ is well-defined. The patterns for a given set of free variables form a lattice by implication (set inclusion of their matches):

$$\psi \leq \psi' :\Leftrightarrow P_\psi \subseteq P_{\psi'} \tag{2}$$

where ψ and ψ' are two Σ-formulae of the form (1) above, s.t. $Free(\psi) = Free(\psi')$. We denote this lattice by $L_{\Sigma,u}$ (or L_u for short when Σ is fixed), where $x_i \in V_{s_i} (1 \leq i \leq n)$ and $u = s_1 \cdots s_n$. (Because formulae are equivalent under renaming of free variables, the sorts of variables only matter.) The top element $\top_{L_u}$ corresponds to the tautology $\exists y_1 \ldots \exists y_n : x_1 =_{s_1} y_1 \wedge \cdots \wedge x_n =_{s_n} y_n$. In particular single-sort patterns with $|Free(\psi)| = 1$ define subdomains of some D_s (with $\{x\} = Free(\psi)$). Elements of L_s represent *logical subdomains* of the given sort, i.e., subdomains expressible in logical formulae over Σ.

For the *structural* representation of (1) we use a tuple $(X, \nu, (\mathbf{G}, \kappa))$ determined as follows: For each sort $s \in S$, the free variables of sort s occuring in (1) are collected in the set X_s, and $X := (X_s)_{s \in S}$. Correspondingly, the bound variables of sort s are collected in the domain G_s of the many-sorted structure $\mathbf{G}$. For each relation symbol $p \in P_u$, we have $\mathbf{G} \models p(y_1, \ldots, y_k)$ iff $p(y_1, \ldots, y_k)$ is an atom in ϕ. For each $s \in S$, κ_s is a mapping on D_s. For each $x \in D_s$, $\kappa_s(x)$ is a formula equivalent to the conjunction of all domain-specific conditions ϕ_i on x. In particular, $\kappa_s(x) := \top$ if no conditions on x occur in (1). Finally, ν is a family of mappings $\nu_s : X_s \to D_s$, where $\nu_s(x)$ is the unique v such that $x =_s v$ occurs in ϕ. We call such a tuple a *windowed structure*, and the pair $(\mathbf{G}, \kappa)$ an *augmented structure*. For technical reasons, we allow arbitrary sets for the domains of $\mathbf{G}$. Figure 2 shows a formula and next to it the associated windowed structure, which may be drawn as a graph.

Entailment is formalized by homomorphisms. Their definition reflects the nestedness of structures. A homomorphism $f : (\mathbf{G_1}, \kappa_1) \to (\mathbf{G_2}, \kappa_2)$ of augmented structures is a homomorphism $f : \mathbf{G_1} \to \mathbf{G_2}$ of many-sorted structures such that $(\kappa_{\mathbf{D}})_s(f(v)) \Rightarrow \kappa_s(v)$ for all $s \in S$ and $v \in G_s$. A homomorphism $f : (X_1, \nu_1, \mathcal{G}_1) \to (X_2, \nu_2, \mathcal{G}_2)$ exists in the case $X_1 \subseteq X_2$ and is then a homomorphism $f : \mathcal{G}_1 \to \mathcal{G}_2$ which preserves free variables, that is $f \circ \nu_1 = \nu_2 \circ \iota_{X_2}^{X_1}$, where $\iota_{X_2}^{X_1}$ is the subset embedding.

Within the scope of this paper, we represent a knowledge base by an augmented structure $\Delta := (\mathbf{D}, \kappa_\Delta)$, where $\mathbf{D}$ is a Σ-structure representing a database and and $(\kappa_\Delta)_s(g)$ is the most specific equivalence class of formulae in L_s characterising the object g. The solution set of a conjunctive query over Δ, represented by a windowed structure $(X, \nu, \mathcal{G})$, is $\mathrm{Hom}(\mathcal{G}, \Delta) \circ \nu$. The solution set can be regarded a subset of $\mathrm{Hom}(X, \Delta)$, if we regard X as a trivial augmented structure (the details are omitted). More generally, we define a *table* to be a pair (X, Λ), where X is a many-sorted family of variables and $\Lambda \subseteq \mathrm{Hom}(X, \Delta)$. By $\mathrm{Tab}(\Delta)$ we denote the set of all tables over Δ. The order on $\mathrm{Tab}(\Delta)$ in which the infimum is the join is given by $(X_1, \Lambda_1) \leq (X_2, \Lambda_2) :\Longleftrightarrow X_2 \subseteq X_1 \wedge \Lambda_1 \circ \iota_{X_1}^{X_2} \subseteq \Lambda_2$.

Given many-sorted, augmented or windowed structures S_1 and S_2, we say that S_1 *generalizes* S_2 and denote this by $S_1 \lesssim S_2$, if a homomorphism $f : S_1 \to S_2$ exists. Generalization is a preorder, and we call S_1 and S_2 *hom-equivalent*, if $S_1 \lesssim S_2$ and $S_2 \lesssim S_1$. It is not difficult to see that the product $\prod_{i \in I} \mathbf{G_i}$ of a family $(\mathbf{G_i})_{i \in I}$ of many-sorted structures is an infimum in the generalization preorder(recall however that an infimum in a preorder is not unique). Infima of augmented or windowed structures are realized by products:

$$\prod_{i\in I}(\mathbf{G_i},\kappa_i) := (\prod_{i\in I}\mathbf{G_i},\kappa, \quad \text{where } \kappa_s(v) := \bigwedge_{i\in I}(\kappa_i)_s(v) \quad ,$$

$$\prod_{i\in I}(X_i,\nu_i,\mathcal{G}_i) := (\bigcap_{i\in I}X_i, \langle\nu_i\rangle, \prod_{i\in I}\mathcal{G}_i, \quad \text{where } \langle\nu_i\rangle(x) := (\nu_i(x))_{i\in I} \quad .$$

Galois Connection The following operations define a Galois connection between $(\mathcal{W},\lesssim)$ and $(\mathrm{Tab}(\varDelta),\leq)$:

$$(X,\nu,\mathcal{G})' := (X, \mathrm{Hom}(\mathcal{G},\varDelta)\circ\nu), \tag{3}$$

$$(X,\varLambda)' := (X, \langle(\lambda)_{\lambda\in\varLambda}\rangle, \varDelta^{\varLambda}) = \prod_{\lambda\in\varLambda}(X,\lambda,\varDelta). \tag{4}$$

Proof. We only show operations are order-reversing, extensivity is easier to see. If $(X_1,\nu_1,\mathcal{G}_1) \lesssim (X_2,\nu_2,\mathcal{G}_2)$, then there is by definition $\varphi \in \mathrm{Hom}(\mathcal{G}_1,\mathcal{G}_2)$ with $\nu_2\circ\iota^{X_1}_{X_2} = \varphi\circ\nu_1$. Thus $\mathrm{Hom}(\mathcal{G}_2,\varDelta)\circ\nu_2\circ\iota^{X_1}_{X_2} = \mathrm{Hom}(\mathcal{G}_2,\varDelta)\circ\varphi\circ\nu_1 \subseteq \mathrm{Hom}(\mathcal{G}_1,\varDelta)$. So $(\cdot)'$ in (3) is order-reversing.

Let $(X_1,\varLambda_1) \leq (X_2,\varLambda_2)$. Then for all $\lambda \in \varLambda_1$ we have $\lambda\circ\iota^{X_2}_{X_1} \in \varLambda_2$, and thus $\prod_{\lambda\in\varLambda_2}(X,\lambda,\varDelta) \lesssim (X_2,\lambda\circ\iota^{X_2}_{X_1},\varDelta) \lesssim (X,\lambda,\varDelta)$. So $\prod_{\lambda\in\varLambda_2}(X,\lambda,\varDelta) \lesssim \prod_{\lambda\in\varLambda_1}(X,\lambda,\varDelta)$, and $(\cdot)'$ in (4) is order-reversing. □

The Galois connection gives rise to the complete lattice $\mathfrak{L}_\varDelta$, or to $\mathfrak{L}_\varDelta[X]$ if restricted to queries ϕ with $Free(\phi) = \bigcup_{s\in S}X_s$:

$$\mathfrak{L}_\varDelta := \{(T,W) \mid T \in \mathrm{Tab}(\varDelta), W \in \mathcal{W}, T' = W, W' = T\} \tag{5}$$

$$\mathfrak{L}_\varDelta[X] := \{(T,W) \in \mathfrak{L}_\varDelta \mid \exists\varLambda : T = (X,\varLambda)\} \tag{6}$$

We hold that only formulas represented by connected patterns (as in Fig. 2) qualify as concept descriptions, and thus only components of powers of $\varDelta$ qualify as concept intents (cf. (4)). The implemented algorithm is still immature and will therefore only briefly be considered in the next section.

#(concepts)	DB relations	+constants	+numeric comparison
x:person	9	12	18
x,y:person	26	59	85

Table 1. Number of concepts, depending on free variables and signature

4 Pattern Browser

4.1 Algorithm

In the order $\varDelta^0, \varDelta^1, \varDelta^2, \ldots$, powers are computed and decomposed into their components, which are paired up with morphisms designating the subjects of the query (cf. $\langle(\lambda)_{\lambda\in\varLambda}\rangle$ in (4)), translated into SQL, paired up with result tables returned by a MySQL server, and then compared (using the order on Tab(Δ)) to eliminate equivalent patterns and build the concept lattice(s). The algorithm terminates when a power Δ^k does not produce new patterns. Table 1 shows the number of generated concepts for queries in one and two free variables of type person, for different settings of query expressiveness. The "naive" algorithm did not terminate in reasonable time even for some of the small examples. This is due to combinatorial explosion of patterns in higher powers of Δ and hardness of query optimization. More efficient algorithms are expected to make use of the fact that graph nodes are tuples over Δ.

```
00| DATABASE BROWSER (press 'h' for help)
01| Concept#0>top
02| Concept#0>intent
03| x : age(x,n0) AND 3<=n0<=59
04| Concept#0>specialize
05| Concept#1>intent
06| x : age(x,n0) AND female(x) AND 3<=n0<=59
07| Concept#1>extent
08|
09|   x
10| -----
11| Anne
12| Dora
13| Emily
14|
15| Concept#1>generalize
16| Concept#0>specialize
17| Concept#3>intent
18| x : age(p0,n0) AND age(p4,n4) AND age(x,n5) AND parent(p0,p4) AND parent(p0,x)
19| AND 31<=n0<=59 AND 7<=n4<=27 AND 3<=n5<=31
20|
21| Concept#3>specialize
22| Concept#14>intent
23| y,x : age(y,n0) AND age(p3,n3) AND age(x,n2) AND parent(y,p3) AND parent(y,x)
24| AND 31<=n0<=59 AND 7<=n3<=27 AND 3<=n2<=31
25|
26| Concept#14>extent
27|
28|  y  |   x
29| ---------
30| Anne| Bob
31| Anne| Chris
32| Bob | Dora
33| Bob | Emily
34|
35| Concept#14>specialize
36| ->Concept#46***
37| ->Concept#18
38|
39| ---------------------------------------------------------------------
40| y,x : age(y,n0) AND age(p3,n3) AND age(x,n4) AND parent(y,p3) AND
41| parent(y,x) AND 31<=n0<=59 AND 7<=n3<=27 AND 3<=n4<=27
```

Fig. 3. Browsing session

4.2 Interface

Figure 3 shows a browsing session, which starts in the top concept. Extent, intent, lower neighbors and upper neighbors of the current concept can each be shown by pressing a key. If a list of neighbors is shown, each concept in the list can be highlighted and examined in the subwindow below the dashed line before it is selected (see Fig.3). The intent is shown as a formula; free variables are listed before the colon, all other variables are existence quantified. The concepts computed are the 103 concepts listed in the right column of Table 1.

5 Related Work

The Galois connection between Σ-theories (sets of Σ-formulae closed under derivation) and categories of models (here Σ-structures) is folklore in model theory and celebrated in textbooks on algebraic and logical specification for data and behaviour. In this paper we use a more restricted connection for formal concept navigation on a knowledge base, focusing on queries (formulae) and their result sets (structures). The abstraction of the representation of data and knowledge bases in such theories renders access to a powerful mathematical tools. Details of a mapping from such data and knowledge bases to theories and structures can be found elsewhere.

The second author used a many-sorted theory construction [11] for expressively modeling typed formal concepts with a rich set of sorts and user-defined data types, including subsorts. However typed conceptual scaling was used for the relevant subsorts to associate a Galois connection with the resulting structures for each sort and to work directly on typed context tables. Patterns were not supported in that work.

If we extend the definition of pattern structure in [5] to categories of patterns (preordered by morphisms), then $(\bigcup_X \mathrm{Hom}(X, \varDelta), (\mathcal{W}, \bigsqcap), \delta)$ is a pattern structure. A morphism $\lambda \in \mathrm{Hom}(X, \varDelta)$, $X \subseteq V$, is essentially a partially defined variable assignment. Each such assignment is naturally identified with a windowed structure $\delta(\lambda) := (\mathrm{dom}(\lambda), \lambda, \mathrm{cod}(\lambda))$, where always $\mathrm{cod}(\lambda) = \varDelta$. The Galois connection stated in [5] becomes

$$A^{\square} := \textstyle\bigsqcap_{\lambda \in \varLambda} \delta(\lambda) \tag{7}$$

$$(X, \nu, \mathcal{G})^{\square} := \{\lambda \in \textstyle\bigcup_X \mathrm{Hom}(X, \varDelta) \mid \exists f : (X, \nu, \mathcal{G}) \to \delta(\lambda)\}. \tag{8}$$

The set A of partial assignments corresponds to the table $(X, \varLambda)$ with $X := \bigcap_{\lambda \in A} \mathrm{dom}(\lambda)$ and $\varLambda := \{\lambda|_X \mid \lambda \in A\}$; the windowed structures $\varLambda'$ and $A^{\square}$ are hom-equivalent. However, the empty tables $(X, \emptyset)$,$X \neq V$, have no representation in this approach; in this, the produced lattice may differ from $\mathfrak{L}_{\varDelta}$.

A relational context family can be defined as a pair $((\mathbb{K}_i)_{i \in I}, (R_j)_{j \in J})$, where each $\mathbb{K}_i =: (G_i, M_i, I_i)$ is a formal context and each R_j is a binary relation on $G_{i_1} \times G_{i_2}$ for some $i_1, i_2 \in I$. The concept lattice for $\mathbb{K}_i$ is denoted by $\mathfrak{B}(\mathbb{K}_i)$. The relational context family corresponds to a many-sorted relational structure with sort set I and family of relations $(R_j)_{j \in J}$. The concept lattices $\mathfrak{B}(\mathbb{K}_i)$, $i \in I$, correspond to the lattices of domain logical formulas. Relational Concept Analysis, as described in [8] and with existential scaling, produces for each sort i a $\bigvee$-sublattice of $\mathfrak{C}_{\varDelta}[X]$, where X contains one variable of sort i, which contains all concepts generated by finite, connected, acylic windowed graphs. This can be shown by induction over the steps of the algorithm given in [8].

6 Conclusion

This paper introduced a novel approach to model queries over relational data – both stored and computed – in terms of FCA lattices. In logical terms, abstract patterns are represented by certain many-sorted formulae with variables. The underlying implication lattices of the formulae and certain structures computed over the database form a Galois connection, as we showed, suitable for navigation of a solution space to the query. Changes in the query are translated into changes to the underlying lattice. Navigation thus includes intra-lattice and inter-lattice moves available to the user exploring domain knowledge over a database in terms of its concrete relation tables and tacit knowledge about these tables. A prototype browser was implemented to evaluate these concepts and was described in the paper.

References

1. Baumeister, H., Bert, D.: Algebraic specification in casl. In: Frappier, M., Habrias, H. (eds.) Software Specification Methods, pp. 209–224. Formal Approaches to Computing and Information Technology FACIT, Springer London (2001), `http://dx.doi.org/10.1007/978-1-4471-0701-9_12`
2. Borovansky, P., Castro, C.: Cooperation of constraint solvers: Using the new process control facilities of elan. In: Proceedings of The Second International Workshop on Rewriting Logic and its Applications, RWLW'98. pp. 379–398 (1998)
3. Broy, M., Wirsing, M.: Partial abstract types. Acta Informatica 18(1), 47–64 (1982)
4. Cohn, A.G.: A more expressive formulation of many sorted logic. Journal of automated reasoning 3(2), 113–200 (1987)
5. Ganter, B., Kuznetsov, S.O.: Pattern structures and their projections. In: Delugach, H.S., Stumme, G. (eds.) Proceedings of ICCS 2001. LNCS, vol. 2120, pp. 129–142. Springer (2001)
6. Goguen, J., Kirchner, C., Kirchner, H., Mgrelis, A., Meseguer, J., Winkler, T.: An introduction to obj 3. In: Kaplan, S., Jouannaud, J.P. (eds.) Conditional Term Rewriting Systems, Lecture Notes in Computer Science, vol. 308, pp. 258–263. Springer Berlin Heidelberg (1988), `http://dx.doi.org/10.1007/3-540-19242-5_22`
7. Grätzer, G.: Universal algebra. Springer (2008)
8. Huchard, M., Hacene, M.R., Roume, C., Valtchev, P.: Relational concept discovery in structured datasets. Annals of Mathematics and Artificial Intelligence 49(1-4), 39–76 (2007)
9. Kirchner, H., Moreau, P.E.: Non-deterministic computations in elan. In: Fiadeiro, J. (ed.) Recent Trends in Algebraic Development Techniques, Lecture Notes in Computer Science, vol. 1589, pp. 168–183. Springer Berlin Heidelberg (1999), `http://dx.doi.org/10.1007/3-540-48483-3_12`
10. Kötters, J.: Concept lattices of a relational structure. In: Pfeiffer, H.D., Ignatov, D.I., Poelmans, J., Gadiraju, N. (eds.) Proceedings of ICCS 2013. LNCS, vol. 7735, pp. 301–310. Springer (2013)
11. Peake, I.D., Thomas, I., Schmidt, H.: Typed formal concept analysis. In: 7th International Conference on Formal Concept Analysis (ICFCA09). pp. 35–51. Springer (2009)