Nested Concept Graphs with Cuts: Mathematical Foundations

Frithjof Dau, Joachim Hereth Correia

Technische Universität Darmstadt, Fachbereich Mathematik
Schloßgartenstr. 7, D-64289 Darmstadt, {dau,hereth}@mathematik.tu-darmstadt.de

Abstract. Conceptual graphs (CGs) have been developed as a graphic representation system for logic with the very goal to be humanly readable and understandable. They are designed to serve in different fields, one of them is information extraction. The best example for this is querying (relational) databases. There are different symbolic query languages, but these languages got a reputation of either being to restricted or to difficult to learn. Using CGs for querying relational databases, we can cope with this problem: As argued in [DH03]), nested CGs cover the whole expressiveness of SQL2, including negation and set functions, while being still comprehensible.
In this paper, we provide the first steps towards a mathematical foundation for a system of CGs which are suited to query databases. First we define their syntax. Next we define contextual models which extend the notion of power context families such that nested relations, which will be used to model the set functions of SQL, can be implemented. Finally, we define the semantics by showing how nested concept graphs are evaluated in the models.

1 Introduction

Connecting conceptual graphs to query relational databases is a self-suggesting idea (e. g., see [EGSW00, EG01]). In [DH03], we provide a couple of examples how queries can be formulated by graphs (which are provided as diagrams) instead of SQL-statements. In this article, we start to work out an appropriate mathematical foundation for these graphs.

The graphs of [DH03] are in two respects an extension of simple concept graphs with cuts of [Da02]. The first main difference is that the graphs herein considered do not form propositions (which can be evaluated in models) as the graphs of [Da02] do, but they describe relations which are built up from 'atomar' or predefined relations. This is done by a fairly simple syntactical and semantical extension of concept graphs with cuts: The vertices of the graphs in [DH03] can have query-marks $?i$, $i \in \mathbb{N}$ as references. If a graph contains query vertices with the references $?i$, $1 \leq i \leq n$, then it describes the relation of all n-tuples $(g_1, \ldots, g_n)$ of objects g_i which can replace the appropriate query markers $?i$ such that the graph evaluates to true in a given model. Simple concept graphs with cuts and query vertices can directly be compared with the relation graphs of Peirce, which are extensively studied by Burch ([Bu91]), and, inspired by the work of Burch, by Wille ([Wi01]) and Pollandt ([Po02]).

The next difference between the graphs in this article and the simple concept graphs with cuts of [Da02] is driven by the set functions (like `COUNT`, `SUM` or `AVG`) of SQL. Set functions can be understood as a kind of iterated relations, i. e. of relations such that the places of a relation can stand again for relations (and not only for objects of the domain we consider). In accordance to the terminology of existential or conceptual graphs, we will call relations like these *nested relations*. So we have to extend the syntax and semantics of simple concept graphs with cuts to the possibility

to describe nested relations. As said in [DH03], the main approach for this is the idea of *hypostatic abstractions* (HAs) of Peirce (see [Pe35]) and their syntactical implementation in existential graphs by contexts. Semantically and very briefly, a hypostatic abstraction occurs if a whole set or class of objects of our domain (our universe of discourse) is considered to be a new, singular object of our consideration on its own. So, in our case, when a relation can occur as an object in a tuple of another relation, the first relation is understood as an object on its own, so it should be described by a HA. Syntactically, we need in concept graphs with cuts a syntactical device which groups the elements (vertices, edges, cuts) of the graph to classes which describe different levels of hypostatic abstractions. In order to do this, we have two possibilities to implement HAs in graphs: Either we introduce a new syntactical device for them (similar to cuts), or we extend the vertices to allow nestings, i. e. we allow that subgraphs may be written *inside* vertices. We decided to use the second approach for a couple of reasons:

- It is better to avoid a new device to keep the syntactical overhead low.
- When using vertices as HAs, i. e. nested relations, we can use the references of the vertices to describe the nested relations (e.g. we can say 'there *exists* a nested relation with ...' or '`AVG` *is* a HA with ...')
- This approach is used in conceptual graphs as well.

So we extend the definition of simple concept graphs with cuts in order to allow nestings of vertices. That is:

> We can scribe graphs or part of graphs *into* vertices. A vertex is a hypostatic abstraction if and only if it contains another graph. (∗)

This main idea, although it seems to be natural, will imply a couple of conclusions, particularly the non-trivial direction '$\Longrightarrow$'.

The resulting graphs will be called *nested concept graphs with cuts* or, in this article, *nested concept graphs* for short (but this is only a short-cut: The graphs of this paper should not be mistaken with the nested concept graphs of [Pr98] which do not allow negations and which have a different semantical background). In these graphs, simple vertices (i. e. vertices which are no HAs) will be used to denote 'ground' objects of our domain and their types, i. e. concepts. A HA h will be used to denote a relation, which is described by the subgraph enclosed by h.[1]

In the next sections, we describe the mathematical foundations for nested concept graphs with cuts. In the first two sections, we start with the syntactical implementation. In the third section, we describe the appropriate models and the evaluation of nested concept graphs with cuts in these models. Finally, we provide a definition of the Φ-operator which maps nested concept graphs to set descriptions based on first order predicate logic.

2 Nested Relational Graphs

In this section, we define the underlying structure of nested concept graphs with cuts (ncgwc). It is important to note that –according to (∗)– we can already see from the *structure*, not from the (later introduced) labelling of vertices and edges, whether a vertex is a simple vertex or a HA.

[1] So relations occur twice: As edges, which can be understood as *pre*-defined relations, and as HAs, which can be understood as *post*-defined relations, built from objects, types and predefined relations.

Definition 1. *A structure* $(V, E, \nu, \top, Cut, area)$ *is called a* nested relational graph with cuts *if*

- V, E *and* Cut *are pairwise disjoint, finite sets whose elements are called* vertices, edges *and* cuts, *respectively,*
- $\nu : E \to \bigcup_{k\in\mathbb{N}} V^k$ *is a mapping,*
- $\top$ *is a single element, called the* sheet of assertion, *and*
- $area : V \dot\cup Cut \dot\cup \{\top\} \to \mathfrak{P}(V \dot\cup E \dot\cup Cut)$ *is a mapping such that*
  a) $area(k_1) \cap area(k_2) \neq \emptyset \Rightarrow k_1 = k_2$ *for* $k_1, k_2 \in V \dot\cup Cut \dot\cup \{\top\}$
  b) $V \dot\cup E \dot\cup Cut = \bigcup\{area(k) \,|\, k \in V \dot\cup Cut \dot\cup \{\top\}\}$,
  c) $x \notin area^n(x)$ *for each* $x \in V \dot\cup \{\top\} \dot\cup Cut$ *and* $n \in \mathbb{N}$
  (with $area^0(x) := \{x\}$ *and* $area^{n+1}(x) := \bigcup\{area(y) \,|\, y \in area^n(x)\}$*).*

For an edge $e \in E$ *with* $\nu(e) = (v_1, \ldots, v_k)$ *we define* $|e| := k$ *and* $\nu(e)\big|_i := v_i$. *Sometimes, we will write* $e\big|_i$ *instead of* $\nu(e)\big|_i$, *and we will write* $e = (v_1, \ldots, v_k)$ *instead of* $\nu(e) = (v_1, \ldots, v_k)$. *We set* $E^{(k)} := \{e \in E \,|\, |e| = k\}$.

For $v \in V$ *we set* $E_v := \{e \in E \mid \exists i.\nu(e)\big|_i = v\}$. *Analogously, for* $e \in E$ *we set* $V_e := \{v \in V \mid \exists i.\nu(e)\big|_i = v\}$.

We set $HA := \{v \in V \,|\, area(v) \neq \emptyset\}$. *The elements of* HA *are called* hypostatic abstractions.[2] *The elements of* $V \backslash HA$ *are called* simple vertices.

The elements of $Cut \dot\cup HA \dot\cup \{\top\}$ *are called* contexts. *As every* $x \in V \dot\cup E \dot\cup Cut$ *is directly enclosed by exactly one context* $k \in Cut \dot\cup HA \dot\cup \{\top\}$, *we can write* $k = area^{-1}(x)$ *for every* $x \in area(c)$, *or more simple and suggestive:* $k = ctx(x)$.

A first, trivial property of the mapping *area* is provided by the next lemma.

Lemma 1. *Let* $(V, E, \nu, \top, Cut, area)$ *be a nested relational graph with cuts, let* $c \in Cut \dot\cup HA \dot\cup \{\top\}$ *be a context, and let* $m \neq n$ *two different natural numbers. Then we have* $area^m(c) \cap area^n(c) = \emptyset$.

Proof: Assume the lemma does not hold. Let $n, m \in \mathbb{N}$ with $area^m(c) \cap area^n(c) \neq \emptyset$ such that $n + m$ is minimal. If $m = 0$, we have an $x \in area^0(c) \cap area^n(c) = \{c\} \cap area^n(c)$ in contradiction to clause a) for *area* in Definition 1. So we have $m > 0$. Analogously, we get $n > 0$.

Now we conclude that there are contexts d, e with $x \in area(d)$, $d \in area^{m-1}(c)$ and $x \in area(e)$, $d \in area^{n-1}(c)$. Using clause a) for *area* in Definition 1, we get $d = e$. So we have $d \in area^{m-1}(c) \cap area^{n-1}(c)$, a contradiction to the minimality of $m + n$. □

NCGwCs will be constructed from nested relational graph with cuts by additionally labelling the vertices and edges with names. There is a crucial difference between concept graphs and most other languages of logic: Usually, the well-formed formulas of a language are built up inductively. In contrast to that, nested concept graphs with cuts are defined in one step. The structure of a formula in a inductively defined language is given by its inductive construction. Nested concept graphs bear a structure as well: Similar to simple relational graphs with cuts, a context k of a relational graph with cuts and nestings may contain other contexts l in its area (i. e. $l \in area(k)$), which in turn may contain further contexts, etc. Is has to be expected that this idea induces an order $\leq$ on the contexts which should be a tree, having the sheet of assertion $\top$ as greatest element. This order reflects the structure of the graph. The naive understanding of $\leq$ is to set $l < k$ iff l is 'deeper nested'

[2] In [Pr98], the are called *complex vertices.*

than k. The next definition is the mathematical implementation of this naive idea. Furthermore the definition extends this idea to the set of vertices and edges. This order will be important in the evaluation of contexts.

Definition 2. *Let $(V, E, \nu, \top, Cut, area)$ be a relational graph with cuts. Define $\beta : V \dot\cup E \dot\cup Cut \dot\cup \{\top\} \to Cut \dot\cup HA \dot\cup \{\top\}$ by*

$$\beta(x) := \begin{cases} x \text{ for } x \in Cut \dot\cup HA \dot\cup \{\top\} \\ ctx(x) \text{ for } x \in (V \backslash HA) \dot\cup E \end{cases}.$$

Now we set

$$x_1 \leq x_2 \ :\Longleftrightarrow\ \exists n \in \mathbb{N}_0 . \beta(x_1) \in area^n(\beta(x_2))$$

for $x_1, x_2 \in V \dot\cup E \dot\cup Cut \dot\cup \{\top\}$.

To avoid misunderstandings, let

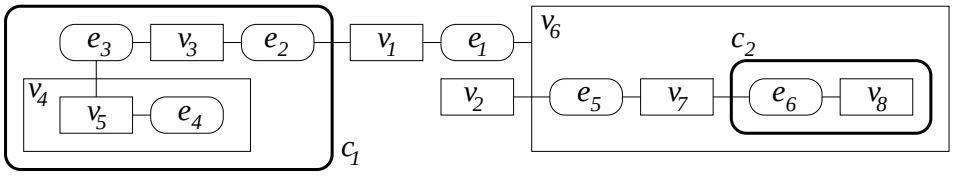
$$x < y :\Longleftrightarrow x \leq y \wedge y \not\leq x$$
$$x \lneq y :\Longleftrightarrow x \leq y \wedge y \neq x$$
$$x \sim y :\Longleftrightarrow x \leq y \wedge y \leq x$$

For $c \in Cut \dot\cup \{\top\}$, we furthermore set

$$\leq[c] := \{x \in V \dot\cup E \dot\cup Cut \dot\cup \{\top\} \mid x \leq c\}$$
$$\lneq[c] := \{x \in V \dot\cup E \dot\cup Cut \dot\cup \{\top\} \mid x \lneq c\}$$

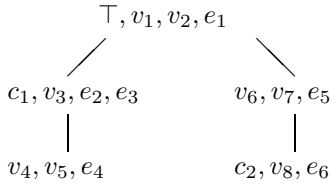
The following is an example of a well-formed nested relational graph with cuts, its diagram (we have written the names for the vertices end edges inside them), its mapping β and the induced relation $\leq$. We see that $\leq$ is a quasiorder and that each equivalence class of the induced equivalence relation $\sim$ contains exactly one context. This will be elaborated after the example.

$$\begin{aligned} \mathfrak{G} := (&\{v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8\}, \{e_1, e_2, e_3, e_4, e_5, e_6\}, \\ &\{(e_1, (v_1, v_6)), (e_2, (v_3, v_1)), (e_3, (v_5, v_3)), (e_4, (v_5)), (e_5, (v_2, v_7)), (e_6, (v_7, v_8))\}, \\ &\top, \{c_1, c_2\}, \{(\top, \{v_1, v_2, v_6, e_1, c_1\}), (c_1, \{v_3, v_4, e_2, e_3\}), \\ &(v_4, \{v_5, e_4\}), (v_6, \{v_7, e_5, c_2\}), (c_2, \{v_8, e_6\})\}) \end{aligned}$$



(Remark: In [Da02] one of the authors provides rules how concept graphs with cuts are drawn. In the graphical representation of nested concept graphs with cuts, we use the rules of [Da02], and the representation of HAs is a merger of the representation of vertices and cuts: A HA is drawn as a box, and this box contains in its inner space all the concept boxes, ovals, and curves of the vertices, edges, and other cuts, resp., which the HA encloses (not necessarily directly). We think that the graphical representation of nested concept graphs is quite natural, and due to limited space, a further discussion of it is omitted here.)

x	$\top$	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	e_1	e_2	e_3	e_4	e_5	e_6	c_1	c_2
$\beta(x)$	$\top$	$\top$	$\top$	c_1	v_4	v_4	v_6	v_6	c_2	$\top$	c_1	c_1	v_4	v_6	c_2	c_1	c_2



As the example shows, it is conjecturable that $\leq$ is a quasiorder. This is not immediately clear from the definitions, but it can be proven. Analogously to [Da02], we get the following lemma:

Lemma 2. *Let $(V, E, \nu, \top, Cut, area)$ be a nested relational graph with cuts. Then we have for $k \in Cut \dot\cup HA \dot\cup \{\top\}$:*

$$\leq[k] = \bigcup\{area^n(k) \,|\, n \in \mathbb{N}_0\} \quad \text{and} \quad \lneq[k] = \bigcup\{area^n(k) \,|\, n \in \mathbb{N}\} \ .$$

Moreover, $\leq$ is a quasiorder such that the restritction $\leq|_{Cut \dot\cup HA \dot\cup \{\top\}}$ is an order on $Cut \dot\cup HA \dot\cup \{\top\}$ which is a tree with the sheet of assertion $\top$ as greatest element.

Lemma 2 yields that we can not have two different contexts k_1, k_2 with $k_1 \sim k_2$. On the other hand we have $v \sim ctx(v)$ and $e \sim ctx(e)$ for each vertex $v \notin HA$ and each edge e. So we see that each class of $\sim$ contains exactly one context. In other words: Each class contains a context k and all simple vertices $v \notin HA$ and all edges e which are placed on the area of k. Thus k is a uniquely given representative of the class, the mapping β from Definition 2 is the mapping which assigns to each element the representing element of the equivalence class of the element.

Now it is easy to describe the mapping ctx in a purely order-theoretic way. We have the following lemma:

Lemma 3. *Let $(V, E, \nu, \top, Cut, area)$ be a nested relational graph with cuts. For each element $x \in V \dot\cup E \dot\cup Cut \dot\cup \{\top\}$, we have:*

$$ctx(x) = min\{k \in Cut \dot\cup HA \dot\cup \{\top\} \,|\, x \lneq k\} \,.$$

Proof: Let $x \in V \dot\cup E \dot\cup Cut \dot\cup \{\top\}$. We set $H := \{h \in HA \dot\cup \{\top\} \,|\, x \lneq h\}$. We have $x \in area(ctx(x))$. Thus Lemma 2 yields $ctx(x) \in H$.

According to Lemma 2, we know that H is totally ordered. Assume we have $l \in H$ with $l < ctx(x)$. That is we have $l \in area^m(ctx(x))$ for an $m \geq 1$. Furthermore we have $x \in area^n(l)$ for an $n \geq 1$. So we have $x \in area^{n+m}(ctx(x))$. From Lemma 1 we conclude $n + m = 1$, a contradiction to $n, m \geq 1$. Thus $ctx(x)$ is the minimal element of H. □

Sometimes we have to consider all elements which are enclosed by an hypostatic abstraction, even if they are deeper nested in some cuts, but *not* if they are contained by a deeper nested hypostatic abstraction. These are the elements x with $ha(x) = h$. In order to to this, it is evident to define a mapping ha as follows:

Definition 3. *Let $(V, E, \nu, \top, Cut, area)$ be a nested relational graph with cuts. We set*

$$ha : \begin{cases} V \dot\cup E \dot\cup Cut \to HA \dot\cup \{\top\} \\ \qquad\qquad x \mapsto min\{h \in HA \dot\cup \{\top\} \,|\, x \lneq h\} \end{cases}$$

Furthermore we define a mapping $encl : HA \to V \dot\cup E \dot\cup Cut$ by:

- $encl^1(h) := area(h)$,

– $encl^{n+1}(h) := \bigcup\{area(c) \,|\, c \in encl^n(h) \cap Cut\}$, *and*
– $encl(h) := \bigcup\{encl^n(h) \,|\, n \in \mathbb{N}\}$.

Lemma 4. *Let* $\mathfrak{G} := (V, E, \nu, \top, Cut, area)$ *be a* relational graph with cuts and nestings *and let* $h \in HA$. *Then we have* $encl(h) = \{x \in V \,\dot\cup\, E \,\dot\cup\, Cut \,|\, ha(x) = h\}$

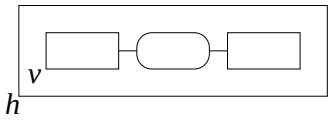
Proof: We have

$$\begin{aligned} k = ha(x) &\iff k = \min\{h \in HA \,\dot\cup\, \{\top\} \,|\, x \lneq h\} \\ &\iff x \in area^n(h) \text{ with } n \in \mathbb{N}, \text{ and there is no } l \in HA \,\dot\cup\, \{\top\} \\ &\qquad \text{with } x \in area^{m_1}(k) \text{ and } k \in area^{m_2}(l), \text{ with } m_1, m_2 \in \mathbb{N} \\ &\iff x \in encl(k) \end{aligned}$$ □

It has to be investigated how the mappings ctx and ha are related to each other. But before we do this, we first want to mention that neither ctx nor ha is order-preserving, i. e. we do not have

$$x_1 \leq x_2 \Longrightarrow ctx(x_1) \leq ctx(x_2) \quad \text{or} \quad x_1 \leq x_2 \Longrightarrow ha(x_1) \leq ha(x_2)\,.$$

To see this, consider the following nested graph:



In this graph, we have $ctx(v) = h$, thus we have $\beta(h) = h = \beta(v)$, so we have $h \leq v$. But we have $ctx(h) = ha(h) = \top \not\leq ctx(v) = ha(v) = h$.

The following lemma shows some very simple connections between the mappings ctx and ha.

Lemma 5. *Let* $x, x_1, x_2 \in V \,\dot\cup\, E \,\dot\cup\, Cut \,\dot\cup\, \{\top\}$. *Then we have:*

$$x \leq ctx(x) \leq ha(x) \;\text{ , and }\; ctx(x_1) \leq ctx(x_2) \Longrightarrow ha(x_1) \leq ha(x_2) \quad .$$

Proof: $x \leq ctx(x) \leq ha(x)$ follows immediately from Lemma 3 and Definition 3.

From $ctx(x_1) \leq ctx(x_2)$ we conclude

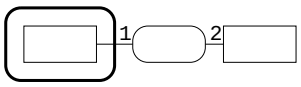
$$\{h \in Cut \,\dot\cup\, HA \,\dot\cup\, \{\top\} \,|\, x_2 \lneq h\} \subseteq \{h \in Cut \,\dot\cup\, HA \,\dot\cup\, \{\top\} \,|\, x_1 \lneq h\} \qquad (1)$$

Now let $h \in HA \,\dot\cup\, \{\top\}$ with $x_2 \lneq h$. From (1) we conclude $x_1 \lneq h$. So we have $\{h \in HA \,\dot\cup\, \{\top\} \,|\, x_2 \lneq h\} \subseteq \{h \in HA \,\dot\cup\, \{\top\} \,|\, x_1 \lneq h\}$, thus $ha(x_1) \leq ha(x_2)$. □

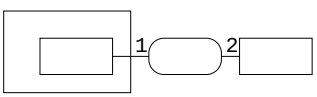
We will only consider graphs in which vertices must not be deeper nested than any relation they are incident with. This applies to all contexts, that is to cuts as well as to hypostatic abstractions.

Definition 4. *Let* $\mathfrak{G} := (V, E, \nu, \top, Cut, area)$ *be a nested relational graph with cuts. If* $ctx(e) \leq ctx(v)$ *for every* $e \in E$ *and* $v \in V_e$, *then* $\mathfrak{G}$ *is said to have* dominating nodes.

The following graphs do not have dominating nodes:



and

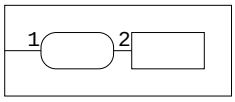


.

On the other hand: If an edge e is incident with an hypostatic abstraction h, it seems to be natural that e has to be placed outside of h. This could be captured by the following condition:

$$ha(e) > h \text{ for every } e \in E \text{ and } h \in V_e \cap HA$$

An example for a graph which violates this condition is the following:



Graphs like this will not arise in the applications we have in mind, but they cause no troubles: The semantics etc. work for graphs like this as well. For this reason, we have not included the condition above in our definition of nested relational graphs.

If a graph fulfills the condition above, we get the following lemma.

Lemma 6. *Let $\mathfrak{G} := (V, E, \nu, \top, Cut, area)$ be a nested relational graph with cuts which satisfies*

$$ha(e) > h \textit{ for every } e \in E \textit{ and } h \in V_e \cap HA$$

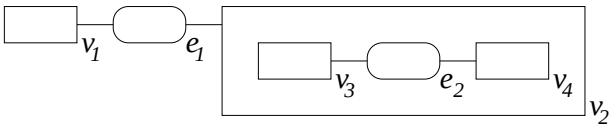
Then we have: If an edge e is incident with an hypostatic abstraction h, then e and h are placed in the same hypostatic abstraction, i. e. we have $ha(e) = ha(h)$.

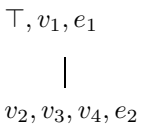
Proof: On the one hand, we have $ctx(e) \leq ctx(h)$. From this we conclude, using Lemma 5, that we have $ha(e) \leq ha(h)$. On the other hand, we have $ha(e) > h$. The definition of ha yields $ha(e) \geq ha(h)$. So we have $ha(h) \leq ha(e) \leq ha(h)$, thus $ha(h) = ha(e)$. □

Next we want to stress the following: In contrast to simple concept graphs, the condition of Definition 4 can not be replaced by

$$e \leq v \text{ for every } e \in E \text{ and } v \in V_e \tag{2}$$

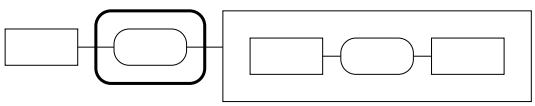
To see this, consider the following graph and its induced quasiorder $\leq$:





We have $ctx(e_1) = ctx(v_1) = ctx(v_2) = \top$ and $ctx(e_2) = ctx(v_3) = ctx(v_3) = v_2$, hence the condition of of Definition 4 is satisfied. On the other hand we do not have $e_1 \leq v_2$, hence (2) is not fulfilled for e_1.

Although e and h are placed in the same hypostatic abstraction, they do not have to be placed in the same context, as the following example shows:



3 Syntax

3.1 Types and Names

With simple concept graphs, we can denote relations between the objects of a given set of objects with relation names. Relations always have a fixed arity. For this reason, we assign in the syntax of simple concept graphs (and other logic languages like first order logic as well) to each relation name an arity.

It is reasonable to use concept graphs as a language to query relational databases, as it is argued in [DH03]. Tables can be understood as relations, and queries as operations on the algebra of relations. But as the discussion in [DH03] shows, the expressive power of SQL goes beyond the possibility to describe relations on the set of objects. A powerful feature of SQL are so-called set functions like `COUNT` or `SUM`. Our understanding of `COUNT` is that `COUNT` is a dyadic relation between natural numbers and relations: A number n stands in relation `COUNT` to a relation R if and only if R contains exactly n tuples. So `COUNT` is not a classical relation on the set of objects, but a relation between objects and relations. For this reason, we have to extend to the notion of relations such that the tuples of a relation may contain not only objects of our ground set, but complete relations again. In other words: We will iterate relations.

To describe the structure of such relations, we will use *signatures* instead of arities. A signature will not only give the arity of a relation, but they will furthermore describe for each place of a tuple the structure of the entries in that place, e.g. whether the entry is an object of our ground set or a relation itself. We will use the sign $\star$ to denote an arbitrary object of our ground set. A 'classical' k-ary relation on our ground set will be therefore described by the signature $(\star, \ldots, \star)$. In particular, signatures are a generalization of arities.

It will turn out that it is reasonable to understand even the objects of our ground set as relations. These considerations yield the following definition of signature:

Definition 5 (Signatures).

Let '$\star$' be a sign. We set that $\star$ is a signature, and if $S_1, \ldots, S_n$, $n \in \mathbb{N}_0$ are signatures, then $(S_1, \ldots, S_n)$ is a signature. The set of all signatures is denoted by Sig. We partition Sig into the following types of signatures:

- *The sign $\star$ is called* object signature.
- *The signature $()$ is called* boolean signature.
- *Every signature $(\star, \ldots, \star)$ is called* simple relation signature.
- *All remaining signatures are called* nested relation signatures.

We set furthermore $Sig^0 := Sig \backslash \{\star\}$.

Next we have to define an alphabet for nested concept graphs. In the case of simple concept graphs, we had relation names and assigned to each relation name its arity. Thus, for nested concept graphs, the approach which suggest itself is to assign signatures to relation names. But this approach is problematic, which can be again explained with the relation `COUNT` in SQL. We have in fact an infinite number of `COUNT`-relations, namely a relation `COUNT` between natural numbers and unary relations, a relation `COUNT` between natural numbers and dyadic relations and so on. But when `COUNT` is applied in a SQL-statement, it is applied to a relation where its arity is known, so we know which of the different `COUNT`-relations should be used. This is a situation similar to object-oriented languages where it is possible to

overload names, i. e. in C++ one function name can be implemented differently for different input types or classes. We adopt this approach and define names as pairs of so-called plain names and their signatures, i. e. a plain name can be used together with different signatures. This yields the following definition.

Definition 6 (Alphabet).

An alphabet *is a pair* $(\mathcal{C}, \mathcal{N})$*, where*

- $(\mathcal{C}, \leq_{\mathcal{C}})$ *is a finite ordered set whose elements are called* concept names*, and*
- $(\mathcal{N}, \leq_{\mathcal{N}})$ *is a finite ordered set which consists of pairs* (N, S) *with* $S \in Sig$*. Each element* $(N, S) \in \mathcal{N}$ *is called* signed name*. For the signed name* (N, S)*, we call* N *the* (plain) name of (N, S) *and* S *the* signature of (N, S)*. We demand that only names with the same signature can be compared, that is we demand* $(N_1, S_1) \leq (N_2, S_2) \Longrightarrow S_1 = S_2$ *.*

Let $(N, S) \in \mathcal{N}$ *be a signed name.*

- *If* $S = \star$*, then* (N, S) *is called* object name*. The set of all object names is denoted by* $\mathcal{N}_{\mathcal{G}}$*.*
- *If* $S = ()$*, then* (N, S) *is called* boolean name*. The set of all boolean names is denoted by* $\mathcal{N}_{\mathcal{B}}$*.*
- *If* $S = (\star, \ldots, \star)$*, then* (N, S) *is called* simple relation name*. The set of all simple relation names is denoted by* $\mathcal{N}_{\mathcal{R},s}$*.*
- *All remaining signed names* (N, S) *are called* nested relation name*. The set of all nested relation names is denoted by* $\mathcal{N}_{\mathcal{R},n}$*.*

Simple relation names and nested relation names are both called relation names*. The set of all relation names (i. e.* $\mathcal{N}_{\mathcal{R},s} \dot{\cup} \mathcal{N}_{\mathcal{R},n}$*) is denoted by* $\mathcal{N}_{\mathcal{R}}$*.*

We demand that we have a plain name $\doteq$ *with* $(\doteq, (S, S)) \in \mathcal{N}$ *for each* $S \in Sig$*.*

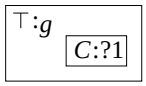
3.2 Nested concept graphs

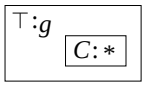
Nested concept graphs with cuts are built from nested relational graphs with cuts by additionally labelling the vertices and edges with names. There are two important points we have to meet:

First of all, when writing the diagram of a nested concept graph, it is desirable that it is sufficient to use plain names. This is again a situation similar to object oriented languages: When we have in C++ a function call of a function which is implemented differently for differents input types, is has to be clear from the context (that is, from the types of the parameter of the function) which implementation is meant. This paradigma should apply to nested concept graphs as well, i. e.

> If we use a plain name in the diagram of a concept graph, it has to be clear from the context which *signed* name is denoted by the plain name.

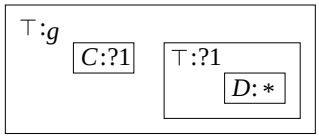
Second, it is important that the labelling is (syntactically) reasonable, i. e. that the graph can be interpreted in a model. To get an impression of this reflections, consider the following two diagrams (where C is the plain part of a concept name and g is a plain name):

$\mathfrak{G}_1 :=$ 

$\mathfrak{G}_2 :=$ 

The hypostatic abstraction (that is the outer vertex) in $\mathfrak{G}_1$ shall describe a (unary) simple relation, which can be seen from the inner vertex, which is labelled with a query marker $?i$. So $\mathfrak{G}_1$ can only be the diagram of a nested concept graph with cuts if g is the plain part of the simple relation name $(g, (*))$. Analogously, $\mathfrak{G}_2$ can only be the diagram of a nested concept graph with cuts if g is the plain part of the boolean name $(g, ())$. We have similar considerations for simple concept graphs: An edge e shall only be labelled with a reasonable relation name, that is, if we have $|e| = k$, then e shall be labelled with a k-ary relation name. But there is a crucial difference between simple and nested concept graphs: In simple concept graphs, it can be derived from their underlying structure (i. e. , simple relational graphs) how vertices and edges can be reasonable labelled. In nested concept graphs, this is not possible. In the examples above, it depends on the *name* which is assigned to the inner vertex how the hypostatic abstraction can be reasonable labelled.

Furthermore, we have another problem which may occur. Consider

$\mathfrak{G}_3 :=$ 

The question arises what kind of relation is described by the outer hypostatic abstraction. More specific: Which signature should g have? We see that g is a unary relation name. From the first vertex in the outer hypostatic abstraction we conclude furthermore that g is a unary *simple* relation name, but from the second vertex in the outer hypostatic abstraction we conclude that g is a unary *nested* relation name. This is contradictory, so $\mathfrak{G}_3$ should not be the diagram of a well-formed nested concept graph with cuts. And again, the names the vertices and edges are labelled with are not chosen in contradiction to the underlying structure to the graph, but they are chosen in contradiction to other names in the graph.

For this reason, the definition of nested concept graphs with cuts is done in two steps. In the first step, we label the vertices and edges of a relational graph with cuts with signed names. This will yield *quasi nested concept graphs*. From the labelling, we can assign to each hypostatic abstraction h (and to other vertices and to the edges) a set of signatures which is derived from the subgraph enclosed by h ($\mathfrak{G}_3$ shows that we sometimes can not derive a unary signature for a given hypostatic abstraction). After that, we only consider quasi nested concept graphs where each vertex and edge has a uniquely determined signature and where all vertices and edges are labelled accordingly to their signatures.

Definition 7 (Quasi Nested Concept Graph with Cuts).

A quasi nested concept graph with cuts over the alphabet $(\mathcal{C}, \mathcal{N})$ *is a structure* $\mathfrak{G} := (V, E, \nu, \top, Cut, area, \kappa, \rho)$ *where*

- $(V, E, \nu, \top, Cut, area)$ *is a nested relational graph with cuts that has dominating nodes,*
- $\kappa : V \dot\cup E \to \mathcal{C} \dot\cup \{\top\} \dot\cup \mathcal{N}$ *is a mapping such that* $\kappa(V \backslash HA) \subseteq \mathcal{C} \dot\cup \{\top\}$, $\kappa(HA) = \{\top\}$ *and* $\kappa(E) \subseteq \mathcal{N}^{\mathcal{R}}$, *and*
- $\rho : V \to \mathcal{N} \dot\cup \{*\} \dot\cup \{?_i \,|\, i \in \mathbb{N}\}$ *is a mapping such that for all* $h \in HA$ *exists a natural number* $ar(h) \in \mathbb{N}_0$ *with*

$$\{i \,|\, \exists v \in encl(h) \text{ with } \rho(v) = ?i\} = \{1, \ldots, ar(h)\}\,.$$

Moreover we set $V^* := \{v \in V \mid \rho(v) = *\}$, $V^? := \{v \in V \mid \rho(v) = ?i, i \in \mathbb{N}\}$, *and* $V^{\mathcal{G}} := \{v \in V \mid \rho(v) \in \mathcal{N}\}$. *The nodes in* V^* *are called* generic nodes, *the nodes in* $V^?$ *are called* query nodes, *and the nodes in* $V^{\mathcal{G}}$ *are called* object nodes.

If a quasi nested concept graph with cuts is given, we can assign to all vertices and edges a set of signatures.

Definition 8 (Signatures of Vertices, Edges, Hypostatic Abstractions).

Let $\mathfrak{G} := (V, E, \nu, \top, Cut, area, \kappa, \rho)$ *be a quasi nested concept graph with cuts over the alphabet* $(\mathcal{C}, \mathcal{N})$. *We assign sets of signatures to vertices, edges and hypostatic abstractions as follows:*

1. *For* $v \in V \backslash HA$, *we set* $sig(v) := \{\star\}$.
2. *Let* $h \in HA$. *We set* $sig(h)|_i := \{sig(v) \mid v \in V$ *with* $ha(v) = h$ *and* $\rho(v) = ?i\}$ *for* $1 \leq i \leq ar(h)$, *and* $sig(h) := sig(h)|_1 \times \ldots \times sig(h)|_{ar(h)}$.
3. *For* $e \in E$ *with* $|e| = k$, *we set* $sig(e) = sig(e|_1) \times \ldots \times sig(e|_k)$.

We will only consider graphs where the signatures of hypostatic abstractions (and hence for edges, too) are uniquely determined. Furthermore, we demand that the signed names we assign to vertices and edges match the signatures of the vertices and edges, respectively. This yields the following definition:

Definition 9 (Nested Concept Graphs with Cuts).

A nested concept graph with cuts over the alphabet $(\mathcal{C}, \mathcal{N})$ *is a quasi nested concept graph with cuts* $\mathfrak{G} := (V, E, \nu, \top, Cut, area, \kappa, \rho)$ *over the alphabet* $(\mathcal{C}, \mathcal{N})$, *where:*

- *For each* $h \in HA$, *we have* $|sig(h)| = 1$. *For* $x \in V \dot\cup E$, *we write* $sig(x) = S$ *instead of* $sig(x) = \{S\}$. *Moreover, for* $1 \leq i \leq ar(h)$, *we have* $sig(h)|_i = \{S\}$ *for an* $S \in Sig$, *and we write* $sig(h)|_i = S$ *instead of* $sig(h)|_i = \{S\}$.
- *If* $v \in V^{\mathcal{G}} \backslash HA$ *and* $\rho(v) = (N, S)$, *then* $S = \star$ *(i. e.* (N, S) *is an object name).*
- *If* $h \in V^{\mathcal{G}} \cap HA$ *and* $\rho(h) = (N, S)$, *then* $S = sig(h)$.
- *If* $e \in E$ *and* $\kappa(e) = (N, S)$, *then* $S = sig(e)$.

If we have $HA = \emptyset$, *then* $\mathfrak{G}$ *is called* simple concept graph with cuts. *If furthermore* $\rho : V \to \mathcal{N} \dot\cup \{*\}$ *holds, then* $\mathfrak{G}$ *is called* simple concept graph with cuts.

For the set E *of edges, let* $E^{id} := \{e \in E \mid \kappa(e) = \doteq\}$ *and* $E^{nonid} := \{e \in E \mid \kappa(e) \neq \doteq\}$. *The elements of* E^{id} *are called* identity-links.

For each vertex and each edge, we can read the signature of the vertex and edge from the graph, thus we can reconstruct the signature of the names which are assigned to the vertices and edges. Furthermore we know that all vertices and edges are labelled with names which match their signatures. For this reason, in the diagrams of the graphs it is sufficient to label edges and vertices with plain names instead of signed names.

4 Semantics

4.1 Models

Before we define the models for nested concept graphs with cuts, we first have to define elements and relations of a signature over a given set of objects.

Definition 10 (Elements and Relations of a Signature).

Let G be a set of objects. A relation of signature $\star$ *is an element of* G.[3] *The set of all relations of signature $\star$ is denoted by $\mathfrak{R}_\star$ resp. by $\mathfrak{R}_\star(G)$ (i. e. we have $\mathfrak{R}_\star = \mathfrak{R}_\star(G) = G$).*

Let $S = (S_1, \ldots, S_n)$ be a signature. An element of signature S *is an element of $\mathfrak{R}_{S_1} \times \ldots \times \mathfrak{R}_{S_n}$ (particularly for $n = 1$, the elements of signature (S_1) are the one-tuples (g) with $g \in \mathfrak{R}_{S_1}$). The set of all elements of signature S is denoted by $\mathfrak{O}_S$ resp. by $\mathfrak{O}_S(G)$. A* relation of signature S *is a subset of $\mathfrak{O}_S$. The set of all relations of signature S is denoted by $\mathfrak{R}_S$ resp. by $\mathfrak{R}_S(G)$. Finally, let $\mathfrak{R}(G) := \bigcup\{\mathfrak{R}_S(G) \,|\, S \in Sig\}$.*

Example: Let $G := \mathbb{N}$. We have the following examples of some signatures:

	$\star$	$()$	$(\star)$	$(\star,\star)$	$(\star,(\star))$
object		$\emptyset$	(1)	$(1,1)$	$(1,\{(3)\})$
			(2)	$(1,2)$	$(4,\{(5),(6)\})$
			(3)	$(3,4)$	$(3,\{(4),(5),(6)\})$
			$\ldots$	$\ldots$	$\ldots$
relation	1	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
	2	$\{\emptyset\}$	$\{(1)\}$	$\{(1,1)\}$	$\{(3,\{(4)\})\}$
	3		$\{(3),(4)\}$	$\{(1,1),(1,2)\}$	$\{(1,\{(3)\}),(3,\{(5),(6)\})\}$
	$\ldots$		$\ldots$	$\ldots$	$\ldots$

The two relations of signature $()$ are independent from the ground set G. It will turn out that they can be interpreted as truth-values, namely as false for $\emptyset$ and true for $\{\emptyset\}$.

For simple concept graphs, the models for these graphs are based on so-called power context families $\vec{\mathbb{K}}$ (see for example [Wi97, Pr98, Da02]). In a power context family, we have a formal context[4] $\mathbb{K}_0$ which encodes the concepts, and for each arity $i \geq 1$, we have a formal context $\mathbb{K}_i$ which encodes the relations of arity i. This approach is transferred to signatures of relations: In nested power context families, we will have a formal context $\mathbb{K}_C$ which encodes the concepts and for each signature S, we will have a formal context $\mathbb{K}_S$ which encodes the relations of signature S.

Definition 11 (Nested Power Context Families).

A nested power context family *is a family of contexts $\vec{\mathbb{K}} := \mathbb{K}_\mathcal{C} \,\dot\cup\, \{\mathbb{K}_S \,|\, S \in Sig^0\}$ with $\mathbb{K}_\mathcal{C} := (G_\mathcal{C}, M_\mathcal{C}, I_\mathcal{C})$ and $\mathbb{K}_S := (G_S, M_S, I_S)$ for each $S \in Sig^0$ such that we have $G_S \subseteq \mathfrak{O}_S(G_\mathcal{C})$ for each signature $S \in Sig^0$. We will abbreviate the term 'nested power context family' by npcf.*

The elements of $G_\mathcal{C}$ are the (plain or ground) objects *of $\vec{\mathbb{K}}$.*

The elements of $\mathfrak{B}(\mathbb{K}_\mathcal{C})$ are called type-concepts.

We set furthermore $\mathfrak{R}^0_{\vec{\mathbb{K}}} := \bigcup_{S \in Sig^0} \mathfrak{B}(\mathbb{K}_S)$ and $\mathfrak{R}_{\vec{\mathbb{K}}} := \mathfrak{R}^0_{\vec{\mathbb{K}}} \,\dot\cup\, G_C$. The elements of $\mathfrak{R}_{\vec{\mathbb{K}}}$ are called relation-concepts.

If we have $G_S = \mathfrak{O}_S(G_\mathcal{C})$ for each signature $S \in Sig^0$, the npcf is called object-complete. *If an object-complete npcf satisfies $\mathfrak{B}(\mathbb{K}_S) = \mathfrak{R}_S(G_\mathcal{C})$ for each $S \in Sig^0$, it is called* relation-complete.

[3] It may seem odd to denote the elements of G by *relations* of signature $\star$, but this terminology unifies the handling of simple vertices and HAs.

[4] Do note mistake the (semantical) formal contexts of formal concept analysis with the (syntactical) contexts of nested relational graphs.

Finally, we get the models for nested concept graphs by assigning the object names to the objects of $\mathbb{K}_C$, the concept names to formal concepts in $\mathbb{K}_C$, and the relation names of signature S to formal concepts in $\mathbb{K}_S$.

Definition 12 (Contextual Models for Alphabets).

Let $(\mathcal{C}, \mathcal{N})$ be an alphabet and let $\vec{\mathbb{K}} := \mathbb{K}_\mathcal{C} \dot\cup \{\mathbb{K}_S \mid S \in Sig^0\}$ be a ncpf.

*Let $\lambda := \lambda_\mathcal{G} \dot\cup \lambda_\mathcal{C} \dot\cup \lambda_\mathcal{R}$ be the disjoint union of the mappings $\lambda_\mathcal{G}: \mathcal{N}_G \to G_\mathcal{C}$, $\lambda_\mathcal{C}: \mathcal{C} \to \underline{\mathfrak{B}}(\mathbb{K}_\mathcal{C})$ and $\lambda_\mathcal{R}: \mathcal{N}_R \dot\cup \mathcal{N}_B \to \mathfrak{R}_{\vec{\mathbb{K}}}$. We call λ a $\vec{\mathbb{K}}$-*interpretation *of $(\mathcal{C}, \mathcal{N})$ iff:*

- *$\lambda_\mathcal{R}((N, S)) \in \underline{\mathfrak{B}}(\mathbb{K}_S)$ for all $S \in Sig^0$,*
- *$\lambda_\mathcal{C}$ and $\lambda_\mathcal{R}$ are order-preserving, (i. e. for two concept-names C_1, C_2 we have $C_1 \leq C_2 \implies \lambda_\mathcal{C}(C_1) \leq \lambda_\mathcal{C}(C_2)$ and for two boolean- or relation-names $(N_1, S), (N_2, S)$ we have $(N_1, S) \leq (N_2, S) \implies \lambda_\mathcal{R}((N, S_1) \leq \lambda_\mathcal{R}((N, S_2))$, and*
- *for each $S \in Sig^0$ and for all elements $g_1, g_2 \in \mathbb{K}_S$, we have $g_1 = g_2 \iff (g_1, g_2) \in Ext(\lambda_\mathcal{R}((\doteq, (S, S))))$.*

The pair $(\vec{\mathbb{K}}, \lambda)$ is called (nested) context-interpretation *of $(\mathcal{C}, \mathcal{N})$ or* (nested) contextual structure over $(\mathcal{C}, \mathcal{N})$.

According to our terminology, $\lambda_\mathcal{G}$ and $\lambda_\mathcal{R}$ assign to a name (N, S) a relation of signature S. Therefore, these two mappings could have been 'aggregated' under one name. We have kept the distinction between $\lambda_\mathcal{G}$ and $\lambda_\mathcal{R}$ to keep the terminology of simple power context families and contextual models, as they are defined in the works of Wille, Prediger, Dau et. al.

4.2 Evaluation

When a concept graph is evaluated in a model, we have to assign relations of our universe of discourse G to them such that the signatures of the vertices are respected. That is: To a vertex v with $sig(v) = S$, we have to assign a relation with signature S. In particular, we assign objects of G to simple vertices. These assignments are done by valuations. Valuations do not need to be total, i. e. we will consider valuations where we assign values only to a subset of V. Nevertheless, there are some conditions which have to be satisfied:

1. It is natural to assign to each vertex $v \in V^\mathcal{N}$ the relation $\lambda(\rho(v))$.
2. For a fixed $i \in \mathbb{N}$ and a fixed hypostatic abstraction $h \in HA \dot\cup \{\top\}$, all vertices v with $ha(v) = h$ and $\rho(v) = ?i$ shall denote the same object.

Furthermore, we want to define specific partial valuations for contexts. Assume we want know which relations can be assigned to the query vertices in a context such that the context evaluates to true or false in a given model. Then we need to know which objects are assigned to generic vertices above the context, but, on the other hand, we should not have assigned objects yet to generic vertices which are enclosed by the context, or to query vertices which are enclosed by any deeper nested hypostatic abstraction. This will be done by *open partial valuations*. The extensions of the open partial valuations to the query vertices in the hypostatic abstraction are the partial valuations which allow us to evaluate the graph which is enclosed by a hypostatic abstraction (and all enclosed cuts) in a given model. These extensions will be called *closed partial valuations* for that context. These considerations yield the following definition:

Definition 13 (Partial and Total Valuations).

Let $\mathfrak{G} := (V, E, \nu, \top, Cut, area, \kappa, \rho)$ *be a nested concept graph with cuts and let* $\mathcal{M} := (\vec{\mathbb{K}}, \lambda)$ *be a model. A mapping* $ref : V' \to \mathfrak{R}_{\vec{\mathbb{K}}}$ *is called* partial valuation of $\mathfrak{G}$*, iff*

1. $V^{\mathcal{G}} \subseteq V' \subseteq V$ *and* $ref(v) = \lambda(\rho(v))$ *for all* $v \in V^{\mathcal{G}}$*, and*
2. *if* $v_1, v_2 \in V$ *with* $ha(v_1) = ha(v_2)$ *and* $\rho(v_1) = \rho(v_2) = ?i$ *for an* $i \in \mathbb{N}$*, and if* $v_1 \in V'$*, then* $v_2 \in V'$ *and* $ref(v_1) = ref(v_2)$*.*
3. $ref(v) \in G_{\mathcal{C}}$ *for* $v \in V' \backslash HA$*, and* $ref(v) \in \mathfrak{B}(\vec{\mathbb{K}}_S)$ *for* $v \in V' \cap HA$ *and* $sig(v) = S$*.*

Next we have to define open and closed partial valuations for a context k*. So let* $ref : V' \to \mathfrak{R}_{\vec{\mathbb{K}}}$ *be a partial valuation and let* $k \in Cut \dot\cup HA \dot\cup \{\top\}$ *be a context.*

If $V' \supseteq \{v \in V \mid ctx(v) > k\}$ *and if* $V' \cap \{v \in V \mid ctx(v) \leq k\} = \emptyset$*, then* ref *is called* open partial valuation for the context k*.*

Now let h *be a hypostatic abstraction, let* ref *be an open partial valuation for* h*, and for* $1 \leq i \leq ar(h)$*, let* x_i *be a relation of signature* $sig(h)|_i$*, i. e. ,* $x_i \in \mathfrak{B}(\vec{\mathbb{K}}_{S_i})$ *with* $S_i := sig(h)|_i$*. Then we set*

$$ref[x_1, \ldots, x_{ar(h)}/h] : \begin{cases} dom(ref) \cup (encl(h) \cap V^?) \to \mathfrak{R}_{\vec{\mathbb{K}}} \\ v \mapsto \begin{cases} ref(v) \text{ , if } v \in dom(ref) \backslash (encl(h) \cap V^?) \\ x_i \text{ , if } v \in encl(h) \cap V^? \text{ and } \rho(v) = ?i \end{cases} \end{cases},$$

Each such mapping $ref[x_1, \ldots, x_{ar(h)}/h]$ *is called* closed partial valuation for k*.*

A partial valuation $ref : V \to \mathfrak{R}_{\vec{\mathbb{K}}}$ *is called* (total) valuation of $\mathfrak{G}$*.*

Please note that a partial valuation for an hypostatic abstraction h assigns a relation to h itself (even for $\rho(h) = *$ or $\rho(h) = ?i$).

Partial valuations for a context k can be extended to partial valuations for contexts which are directly enclosed by k. We have the following lemma:

Lemma 7 (Continuations of Part. Val. are Part. Val. for Contexts).

Let k *be a context and let* ref *be a partial valuation for* k*. Let* $\widetilde{ref}$ *: be a mapping with* $\widetilde{ref} \supseteq ref$ *and* $dom(\widetilde{ref} = dom(ref) \cup (V \cap area(k))$*. Then we have:*

1. *If* $c \in area(k)$ *is a cut, then* $\widetilde{ref}$ *is a partial valuation for* c*.*
2. *If* $c \in area(k)$ *is a hypostatic abstraction and if* $x_1, \ldots, x_{ar(h)} \in \mathfrak{R}_{\vec{\mathbb{K}}}$*, then* $ref[x_1, \ldots, x_{ar(h)}/h]$ *is partial valuation for* h*.*

Now we can define how a nested concept graph $\mathfrak{G}$ is valuated in a model, i. e. in a nested contextual structure $\mathcal{M} := (\vec{\mathbb{K}}, \lambda)$.

For a given hypostatic abstraction h, h can only be evaluated if we have assigned relations to all query vertices which are enclosed by h (which is done by closed partial valuations for h). Now h shall describe the relation of all tuples of objects, which can replace the query vertices such that the evaluation yields true. For this reason, we define in the next definition two different notions:

1. When a hypostatic abstraction (or an enclosed cut) k evaluates for a closed partial valuation ref in $\mathcal{M}$ to true, we will write $\mathcal{M} \models \mathfrak{G}[k, ref]$.
2. a hypostatic abstraction h and an open partial valuation ref for h describes the relation of all tuples of objects which can replace the the query vertices such that h evaluates to true. This relation will be denoted by $\mathfrak{R}_{\mathcal{M},\mathfrak{G},h,ref}$.

Definition 14 (Evaluation).

Let $\mathfrak{G} := (V, E, \nu, \top, Cut, area, \kappa, \rho)$ be an nested relation graph with cuts and let $\mathcal{M} := (\mathfrak{R}_{\vec{\mathbb{K}}}, \lambda)$ be a model. Inductively over the tree $Cut \dot\cup HA \dot\cup \{\top\}$, we define

1. *$\mathcal{M} \models \mathfrak{G}[k, ref]$ for each context $k \in Cut \dot\cup HA \dot\cup \{\top\}$ and every closed partial valuation $ref : V' \subseteq V \to \mathfrak{R}_{\vec{\mathbb{K}}}$ for k:*
   $\mathcal{M} \models \mathfrak{G}[k, ref] :\Longleftrightarrow$
   ref can be extended to a partial valuation $\widetilde{ref} : V' \cup (V \cap area(k)) \to \mathfrak{R}_{\vec{\mathbb{K}}}$ (i. e. , $\widetilde{ref}(v) = ref(v)$ for all $v \in V'$), such that the following conditions hold:
   - *$\widetilde{ref}(v) \in \lambda(\kappa(v))$ for each vertex$v \in V \cap area(k)$ with $\kappa(v) \neq \top$ (vertex condition)*
   - *$\widetilde{ref}(e) \in \lambda(\kappa(e))$ for each edge $e \in E \cap area(k)$ (edge condition)*
   - *$\widetilde{ref}(h) = \mathfrak{R}_{\mathcal{M},\mathfrak{G},h,ref}$ for each hyp. abstraction $h \in HA \cap area(k)$ (hyp. abst. condition and iteration over $Cut \dot\cup HA \dot\cup \{\top\}$)*
   - *$\mathcal{M} \not\models \mathfrak{G}[c, \widetilde{ref}]$ for each cut $c \in Cut \cap area(k)$ (cut condition and iteration over $Cut \dot\cup HA \dot\cup \{\top\}$)*
2. *Let h be a hypostatic abstraction, let ref be an open partial valuation for h, and for $1 \leq i \leq ar(h)$, let x_i be a relation of signature $sig(h)|_i$. We set:*

$$\mathfrak{R}_{\mathcal{M},\mathfrak{G},h,ref} := \{(x_1, \ldots, x_{ar(h)}) \mid \mathcal{M} \models \mathfrak{G}[h, ref[x_1, \ldots, x_{ar(h)}/h]]$$

Finally we define

$$\mathfrak{R}_{\mathcal{M},\mathfrak{G}} := \mathfrak{R}_{\mathcal{M},\mathfrak{G},\top,\emptyset}$$

It is not immediately clear that this definition yields a well-defined entailment relation $\models$ and well-defined relations $\mathfrak{R}_{\mathcal{M},\mathfrak{G},h,ref}$, but this can be seen inductively over the tree of contexts. In order to do this, please note that the vertex condition only applies to simple vertices. Moreover, the edge-condition an only be evaluated because we consider graphs with dominating nodes: To see this, consider a context k, a partial valuation $ref : V' \to \mathfrak{R}_{\vec{\mathbb{K}}}$ for k and an extension $\widetilde{ref}$ of ref to $area(k)$. Let $e \in area(k)$ be an edge which shall be evaluated. As we have $V' \supseteq \{v \in V \mid ctx(v) > k\}$ by Definition 13, we know that $\widetilde{ref}$ assigns objects to all vertices v with $ctx(v) \geq k = ctx(e)$. Now condition $ctx(e) \leq ctx(v)$ for every $v \in V_e$ of Definition 4 makes sure that we have already assigned objects to all vertices which are incident with e.

5 The Φ-Operator

In this section, we give a definition of the Φ-operator for nested concept graphs. In [Da02], one of the authors already provided a definition of the Φ-operator which translates a concept graph with cut to an closed FOPL-formula with the same meaning. Now we consider concept graphs which describe relations, so it seems natural that the Φ-operator should yield an FOPL-formula with free variables. This is true for *simple* concept graphs which describe relations, and this idea is extended to nested concept graphs with cuts. The main idea if the following: A simple concept graph which describes an n-ary relation is mapped to a set definition $\{(x_1, \ldots, x_n) \mid f(x_1, \ldots, x_n)\}$, where f is a FOPL-formula with the free variables $x_1, \ldots, x_n$ (and f is the result of the Φ-operator of [Da02] for simple concept graphs with cuts). If we consider nested concept graphs with cuts, we allow nested set

definitions as well, i. e. we allow a set definition to occur in the formula of another set definition.

Let $\mathfrak{G} := (V, E, \nu, \top, Cut, area, \kappa, \rho)$ be a concept graph with cuts, nestings, and dominating nodes. First, we assign to each $v \in V$ a term $\Phi_t(v)$ as follows: If $\rho(v) = g, g \in \mathcal{N}$, we set $\Phi_t(v) := g$. To each $v \in V$ with $\rho(v) = *$ or $\rho(v) = ?i, i \in \mathbb{N}$ we assign a variable such that the same variable is assigned to different vertices v_1, v_2 iff $\rho(v_1) = \rho(v_2) = ?i, i \in \mathbb{N}$ and $ha(v_1) = ha(v_2)$.[5] We need a further variable $\alpha_{empty} \notin \{\alpha_v \mid v \in V^* \cup V^?\}$.

Now we assign to each context $k \in Cut \dot\cup HA \dot\cup \{\top\}$ a set definition $\Phi(\mathfrak{G}, k)$, if k is a hypostatic abstraction, and a formula $\Phi(\mathfrak{G}, k)$, if k is a cut. This is done inductively over the tree $(Cut \dot\cup HA \dot\cup \{\top\}, \leq)$. So let k be a context such that $\Phi(\mathfrak{G}, l)$ is already defined for each context $l < k$.

First, we define a formula f which encodes all edges, vertices, cuts and hypostatic abstractions which are directly enclosed by k. Set $f := \exists \alpha_{empty}.\top(\alpha_{empty})$, if $area(k) = \emptyset$. Otherwise, let f be the conjunction of the following formulas:[6]

$$\begin{aligned}
\kappa(w)\underline{(}\Phi_t(w)\underline{)} &\text{ with } w \in area(k) \cap V\,,\\
\kappa(e)\underline{(}\Phi_t(w_1)\underline{,}\ldots\underline{,}\Phi_t(w_j)\underline{)} &\text{ with } e \in area(k) \cap E^{nonid} \text{ and } \nu(e) = (w_1, \ldots, w_j)\,,\\
\Phi_t(w_1)\underline{=}\Phi_t(w_2) &\text{ with } k \in area(k) \cap E^{id} \text{ und } \nu(k) = (w_1, w_2)\,,\\
\underline{\neg}\Phi(\mathfrak{G}, d) &\text{ with } d \in area(k) \cap Cut\,, \text{ and}\\
\Phi_t(h)\underline{=}\Phi(\mathfrak{G}, h) &\text{ with } h \in area(k) \cap HA\,.
\end{aligned}$$

It is sufficient to use the plain names instead of the signed names.

Let $v_1, \ldots, v_n$ be the vertices of $\mathfrak{G}$ which are enclosed by c and which fulfill $\rho(v_i) = *$. If $k \in Cut$, we set

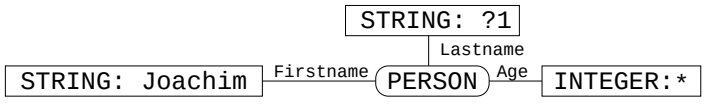
$$\Phi(\mathfrak{G}, k) := \underline{\exists}\alpha_{v_1}\underline{.}\ldots\underline{\exists}\alpha_{v_n}\underline{.(}f\underline{)}\,.$$

For $k \in HA$ and $1 \leq i \leq ar(h)$, let β_i be the variable which Φ_t assigns to each $v \in area(h)$ with $\rho(v) = ?i$. Now we set

$$\Phi(\mathfrak{G}, k) := \underline{\{\,(}\beta_1\underline{,}\ldots\underline{,}\beta_{ar(h)}\underline{)\mid\exists}\alpha_{v_1}\underline{.}\ldots\underline{\exists}\alpha_{v_n}\underline{.(}f\underline{)\,\}}\,.$$

Finally set $\Phi(\mathfrak{G}) := \Phi(\mathfrak{G}, \top)$, and the definition of Φ is finished.

In the following, we provide some examples for the Φ-operator. We start with a very simple graph, which is the graph of Figure 2 in [DH03].



This graph is translated to[7]

[5] i. e. we assign variables such that we have

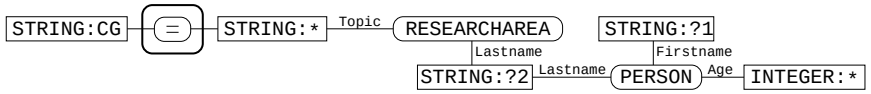
$$\begin{aligned}\Phi_t(v_1) = \Phi_t(v_2) \Longleftrightarrow (v_1 = v_2 \vee (\exists g \in \mathcal{N}\,.\,\rho(v_1) = \rho(v_2) = g)\\ \vee (\exists i \in \mathbb{N}\,.\,\rho(v_1) = \rho(v_2) = ?i \wedge ha(v_1) = ha(v_2)))\quad .\end{aligned}$$

[6] The signs which have to be understood literally are underlined.

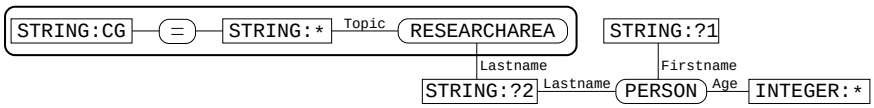
[7] Due to space limitations, we abbreviated STRING by STR, INTEGER by INT, PERSON by PER and RESEARCHAREA by RA.

$$\{(x_1) \,|\, \exists x_2 \,.\, STR(Joachim) \wedge STR(x_1) \wedge INT(x_2) \wedge PER(x_1, Joachim, x_2)\}$$

The next graphs are the graphs of Figure 7 and Figure 8 in [DH03]. They involve cuts, thus we will have negated subformulas in their translations.



and



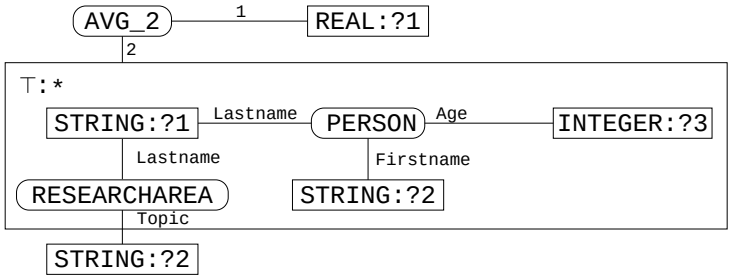
These graphs are translated to

$$\{(x_1, x_2) \,|\, \exists x_3 . \exists x_4 \,.\, STR(x_1) \wedge STR(x_2) \wedge INT(x_3) \wedge STR(x_4) \wedge STR(CG) \\ \wedge PER(x_1, x_2, x_2) \wedge RA(x_4, x_2) \wedge \neg(CG = x_4)\}$$

resp.

$$\{(x_1, x_2) \,|\, \exists x_3 \,.\, STR(x_1) \wedge STR(x_2) \wedge INT(x_3) \wedge PER(x_1, x_2, x_3) \\ \wedge \neg \exists x_4 . (STR(x_4) \wedge STR(CG) \wedge RA(x_4, x_2) \wedge CG = x_4)\}$$

Finally we provide a graph with a hypostatic abstraction. It is taken from Figure 11 in [DH03]. The HA will be translated to a nested set definition.



First we translate the outer context. This yields:

$$\{(x_1, x_2) \,|\, \exists x_3 . REAL(x_1) \wedge STR(x_2) \wedge \top(x_3) \wedge AVG(x_1, x_3) \wedge x_4 = \{\ldots\}\}$$

The inner context yields:

$$\{(y_1, y_2, y_3) \,|\, STR(y_1) \wedge STR(y_2) \wedge INT(y_3) \wedge RA(x_2, y_1) \wedge PER(y_1, y_2, y_3)\}$$

So, we translate the graph above to

$$\{(x_1, x_2) \,|\, \exists x_3 . REAL(x_1) \wedge STR(x_2) \wedge \top(x_3) \wedge AVG(x_1, x_3) \\ \wedge x_4 = \{(y_1, y_2, y_3) \,|\, STR(y_1) \wedge STR(y_2) \wedge INT(y_3) \\ \wedge RA(x_2, y_1) \wedge PER(y_1, y_2, y_3)\}\}$$

6 Outlook

As said in the introduction, this is only the very beginning of a mathematical foundation for query graphs. There is a lot more work to do: The ground domain is typed, so it is reasonable to develop a theory with typed signatures (but first attempts have shown that this approach leads to unexpected difficulties). Moreover, it has to be investigated whether the expressiveness of the graphs is restricted enough such that a sound and complete calculus is possible, and if this is the case, a calculus should be found. Finally, it is desirable that there will be implementations of these graphs to query existing databases: Only human interaction can show whether the theoretical considerations of [DH03] and this paper yield working and comprehensible query graphs for databases.

References

[Bu91] R. W. Burch: *A Peircean Reduction Thesis: The Foundations of Topological Logic.* Texas Tech University Press, 1991.

[Da02] F. Dau: *The Logic System of Concept Graphs with Negations and its Relationship to Predicate Logic* PhD-Thesis, Darmstadt University of Technology, 2002. To appear.
`www.mathematik.tu-darmstadt.de/∼dau/dissertation.pdf`

[DH03] F. Dau, J. Hereth: *Nested Concept Graphs: Applications for Databases.* Submitted for ICCS 2003.

[EGSW00] P. Eklund, B. Groh, G. Stumme, R. Wille: *A contextual-logic extension of toscana.* In: B. Ganter, G. W. Mineau (Eds.): Conceptual Structures: Logical, Linguistic, and Computational Issues. LNAI 1867, Springer Verlag, Berlin–New York 2000, 453-467.

[EG01] P. Eklund, B. Groh: *A cg-query engine based on relational power context families.* In: G. Mineau (Ed.): Conceptual Structures: Extracting and Representing Semantics. ICCS 2001. Dept. of Computer Science, Quebec, Canada, 33-46.

[GW99a] B. Ganter, R. Wille: *Formal Concept Analysis: Mathematical Foundations.* Springer, Berlin–Heidelberg–New York 1999.

[Pe35] *C. S. Peirce. Collected Papers.* Harvard University Press, Cambrigde, Massachusetts, 1931-1935.

[Pr98] S. Prediger: *Kontextuelle Urteilslogik mit Begriffsgraphen. Ein Beitrag zur Restrukturierung der mathematischen Logik.* Aachen, Shaker Verlag 1998.

[Po02] S. Pollandt: *Relation Graphs: A Structure for Representing Relations in Contextual Logic of Relations.* In: U. Priss,, D. Corbett, G. Angelova (Eds.): Conceptual Structures: Integration and Interfaces. LNAI 2393, Springer Verlag, Berlin–New York 2002, 34–48.

[So84] J. F. Sowa: *Conceptual Structures: Information Processing in Mind and Machine.* Addison Wesley Publishing Company Reading, 1984.

[So00] J. F. Sowa: *Knowledge Representation: Logical, Philosophical, and Computational Foundations.* Brooks Cole Publishing Co., Pacific Grove, CA, 2000.

[SoWWW] J. F. Sowa: *Semantic Foundations of Contexts.*
`http://www.jfsowa.com/ontology/contexts.htm`

[Wi97] R. Wille: *Conceptual Graphs and Formal Concept Analysis.* In: D. Lukose et al. (Hrsg.): Conceptual Structures: Fulfilling Peirce's Dream. LNAI 1257, Springer Verlag, Berlin–New York 1997, 290–303.

[Wi01] R. Wille: *Lecture Notes on Contextual Logic of Relations.* FB4-Preprint, Darmstadt University of Technology, 2000.