A Basic Mathematical Framework for Conceptual Graphs

Philip H.P. Nguyen[1] and Dan Corbett[2]

Abstract - Based on the original idea of Sowa on conceptual graph and a recent formalism by Corbett on ontology, this paper presents a rigorous mathematization of basic concepts encountered in the Conceptual Structure Theory, including canon, ontology, conceptual graph, projection, and canonical formation operations, with the aim of deriving their mathematical properties and applying them to future research and development on knowledge representation. Our proposed formalism enhances the Conceptual Structure Theory and enables it to compare favorably with other alternative methods such as the Formal Concept Analysis theory.

Index Terms - Graph algorithms, knowledge representation formalisms and methods, ontology design.

1. INTRODUCTION

J. Sowa, the main author of the conceptual graph theory, which forms the basis for the *Conceptual Structure Theory* (CST), remarked: “A conceptual graph has no meaning in isolation. Only through the semantic network are its concepts and relations linked to context, language, emotion, and perception” [21]. An attempt to implement Sowa’s idea was introduced in [3], with the notion of *support* for conceptual graphs. Such a graph is called an *S-graph*. S-graph’s support adds the semantic background to the piece of knowledge that the graph represents and therefore enables knowledge engineers to better represent the world that they wish to model. S-graph is later enhanced with the introduction of *Simple Conceptual Graph* [4][1][2][7][8], which assists reasoning through graph-theoretic operations, such as graph projection. A similar definition is *Simple Concept Graph* [9], which, through the Formal Concept Analysis (FCA) theory (see later), leads to the definition of *standard models* that enable complex reasoning over models of

1 Philip H.P. Nguyen is with Justice Technology Services, Attorney-General’s Department, Government of South Australia, 30, Wakefield Street, Adelaide, SA 5000, Australia. Email: nguyen.philip@saugov.sa.gov.au - http://www.justice.sa.gov.au
2 Dan Corbett is with Science Applications International Corporation, Artificial Intelligence, Washington, DC, USA. Email: corbettd@saic.com - http://www.saic.com

concept graphs. There have also been numerous undertakings to formalize conceptual graphs and to extrapolate their mathematical properties [18] [20] [29], including an attempt to standardize them [23]. In [5] and [6], D. Corbett proposes a general method to achieve interoperability of knowledge bases created by different knowledge engineers, through the use of canonical ontologies. *Canon* and *ontology*, as defined in those papers, are semantically similar to *support* in S-graphs and offer a way of formalizing and categorize *concepts*. FCA is another important theory that applies lattice theory to knowledge discovery by providing a mathematization of *concept* and *concept hierarchy*, that "activates mathematical methods for conceptual data analysis and knowledge processing" [10][11][32]. FCA's *formal context* [28][26] is similar to D. Corbett's canon and ontology. High-level comparison between CST and FCA is explored in [17]. And finally, although CST could be considered a branch of Formal Logic and other Formal Logic formalisms and operations exist (e.g. Semantic Web models based on Description Logic) and could be similar to the ones suggested in this paper, we have not attempted to explore this aspect within the limitation of this paper. Our scope is limited to introducing an enhanced mathematical formalization of canon, ontology, conceptual graph, and to exploring their manipulation through projection and canonical formation operations. Our comparison with other formalisms is mainly limited to that with FCA.

Our paper is broadly organized as follows. Section 3 introduces new and enhanced definitions of canon, ontology and conceptual graph, serving as the basis for the manipulation and comparison of knowledge in Section 4, which involves a formal re-mathematization of the operation of projection of concept, relation and conceptual graph. This leads to Theorem 1, which demonstrates that projection could be used as a method to "condense" knowledge representation. Reasoning on conceptual graphs is achieved through canonical graphs, formally introduced in Section 5, in which some of their intrinsic properties are presented. The Section concludes with Theorem 2, stating that reasoning on conceptual graphs can be preserved through their projections. Comparison between CST and FCA is undertaken in Section 6, which includes details on their main similarities and differences.

2. NOTATIONS

Throughout this paper, we adopt the usual terminology and symbology found in mathematical and conceptual graph theories, in particular:

- A partially ordered set (or “poset” for short) is a set on which exists a binary relation, which is reflexive, anti-symmetric and transitive. A lattice is a poset in which every pair of elements has a supremum (or least upper bound) and an infimum (or greatest lower bound). Within CST, we define a lattice of concept or relation types as a poset in which every pair of types has a *unique* minimal common supertype and a *unique* maximal common subtype. Since we only deal with finite sets of concepts in this paper, each type lattice necessarily has a top node and a bottom node. Without loss of generality, we also automatically add the *universal* node (represented by the symbol “$\top$”) on top of the lattice and the *absurd* node (represented by the symbol “$\perp$”) at the bottom of the lattice (unless they are already in the lattice). The universal and absurd nodes serve as the common points (among others) for all lattices, where our reasoning process on ontology comparison and merging could start. The assumption of uniqueness of the minimal common supertype and the maximal common subtype for any pair of types ensures “soundness of mathematical reasoning” over the ontology or canon represented by the lattice. Note that this assumption is also made FCA [30][27].
- $\varphi(S)$ denotes the set of all subsets of the set S.

- Given a function f: X -> Y and a set $S \subseteq X$, we define the function $f\angle_S$ as the restriction of f on the domain set S, i.e. $f\angle_S$: S -> Y with $\forall s \in S$ $f\angle_S(s)=f(s)$.

Other notations and terminologies will be introduced when their contexts are discussed.

3. CANON, ONTOLOGY AND CONCEPTUAL GRAPH

In simple terms, a canon is a framework for knowledge organization (e.g. an English dictionary) and an ontology is a subset of a canon dealing with a particular subject domain (e.g. a specialist dictionary on health), *a specification of a conceptualization*, as per [13]. One may argue that a specialist dictionary may omit many words in an English dictionary but may probably provide other additional terms that are only relevant to the specialist domain. In this case, in our definitions, a canon could be considered as a *universal ontology* encompassing other ontologies, which are *domain ontologies*. A conceptual graph is a representation of a part of knowledge under an ontology or a canon (e.g. a statement on any subject or on a subject within a particular domain, using words of a general or specialist dictionary). It could thus be considered that conceptual graphs represent knowledge itself while canons and ontologies are mainly frameworks for the organization of knowledge that conceptual graphs represent. Therefore canons and ontologies may be more appropriately classified as “meta-knowledge”.

This section further enhances definitions specified in [5] and [6], which in turn build on the concepts initially defined in [21]. Our definitions also generalize other attempts, such as the formalization of conceptual graph proposed in [20] and the *Universal Data Structure* suggested in [16]. A comparison between our formalism and FCA is made in Section 6.

3.1. Canon

The idea of a canon is to capture the overall structure of concepts and the environment in which they are utilized. A canon has three roles: firstly, it defines hierarchies of relationships between concept types and between relation types; secondly, it defines the relationship between each relation type and its associated concept types; and thirdly, it defines the relationships between concept types and their instances, if exist, in the real world. Those concepts are formalized as follows:

3.1.1. Definition

A canon ***K*** is a quintuple (or 5-tuple) ***K*** = (T, I, $\leq$, *conf*, B) whose elements are defined by statements (1) to (6) below:

(1) T is the set of concept and relation types, i.e. $T = T_C \cup T_R$ in which:

- T_C is the set of concept types.
- T_R is the set of relation types.
- In addition, we assume:
  - T_C and T_R are disjunctive.
  - T_C and T_R are finite sets.
  - T_C and T_R each has a lattice structure (which includes $\top$ and $\bot$).
  - When there is no confusion, we also call T_C (resp. T_R) the "concept type lattice" (resp. "relation type lattice") of ***K*** (or simply "concept lattice", resp. "relation lattice", of ***K***). We refer to a vertex of a concept lattice (resp. relation lattice) as a "concept node" (resp. "relation node") and an edge of the lattice as the "subsumption relation" linking the two nodes.

  We also assume in the rest of this paper that any sets of concept or relation types, in particular, T, T_C and T_R , do not contain duplicate elements. This is similar to a dictionary in which each entry is unique (unless the word has a meaning different from that of other identical words, but in this case each word and its meaning constitute a unique entry in the dictionary).

(2) I is the set of instances of T_C , called “individual markers”. An element of I is an instance or a realization of a concept type in the real world. A concept type may have any arbitrary number of instances. In addition, we assume that I includes the generic marker “*”, which is a fictitious instance that can be associated with any concept or relation type, and I does not contain duplicate elements.

(3) $\leq$ is the subsumption (or subtype) relation of T, i.e. $\leq$ is a function defined as:

$$\leq:\ (T_C \text{ x } T_C) \cup (T_R \text{ x } T_R) \text{ -> } \{\text{true, false}\}$$

- “$\leq$(c,d)=true” (commonly written as “c$\leq$d”) means: “c is a subtype of d”.
- We define the strict (or proper) subsumption relation “<” as:

$$\forall c,d \in T_C \text{ (or } T_R) \ \ c < d \Leftrightarrow c \leq d \text{ and } c \neq d$$

- “$\leq$(c,d)=false” (commonly written as “c $\nleq$ d”) means: either “d<c” or “c and d are not comparable, or more precisely, not *compatible*”.

(4) *conf* is the “conformity” relation, that relates each individual marker in I (except “*”) to one, and one only, concept type in T_C , i.e. *conf* is a function defined as:

$$conf:\ I\backslash\{*\} \text{ -> } T_C$$

Notes:

- In [4], it is suggested that a marker (representing an *entity*, as per terminology used in [4]) could be associated with multiple concept types. The entity is then said to be represented by multiple *co-referent concept nodes*. Since we assume that our concept types form a lattice structure, there must be a *unique* greatest lower bound for all those co-referent concept nodes. We assume that that unique greatest lower bound should also be a co-referent concept node, and in this case the function *conf* is equivalent to assigning each marker with that unique concept type representing the greatest lower bound of all co-referent concept nodes that could be used with that marker.

- So the *conf* relation does not mean that a marker can only be used with a concept type. It can be used with any concept types that subsumes the type defined through *conf* (see an example in Section 3.3.(6)).
- Our assumption (and restriction) that each individual marker (except the generic marker “*”) can only relate to one, and one only, concept type, ensures that the *typing* of the concepts in the canon makes sense and the whole canon is semantically consistent (or “well formed”). This also qualifies *conf* as a mathematical *function*.

(5) *B* is the Canonical Basis function, that associates each relation type with the concept types that may be used in that relation, i.e. *B* is a function defined as:

B: T_R -> $\varphi(T_C)$

where:

$\forall r \in T_R \qquad B(r) = \{c_1,..,c_n\}$ with:

- n is a variable associated with r.
- The set $\{c_1,..,c_n\}$ is an ordered set and could contain duplicate elements.
- Each c_i is called an *argument* of r and the number of elements of the set is called the *arity* (or *valence*) of the relation.

(6) The functions ≤ and *B* are linked through the following relationship:

$\forall r, r' \in T_R$ with $B(r) = \{c_1,..,c_n\}$ and $B(r') = \{c'_1,..,c'_m\}$, we have:

$r \le r' \implies (m = n \text{ and } \forall i \le n \;\; c_i \le c'_i)$

This means that if a relation type is subsumed into another relation type then they must have the same concept arguments and the concept arguments of the first relation are subsumed into the respective concept arguments of the second relation.

An example is the two relations: “Poss(Animate,Entity)” and “Has(Entity,Entity)” (as per Appendix B.3 of [22]). “Poss(Animate,Entity)” is a relation type between an Animate and an Entity, and means “an animate being possesses (or owns) an entity”. “Has(Entity,Entity)” is another relation type which has a more general meaning of “an entity owns another entity (or has another entity as one of its parts)”. We have: Poss(Animate,Entity) ≤ Has(Entity,Entity), since the semantics of the first relation is included in that of the second relation and the concept arguments of the first relation are subsumed in the concept arguments of the second relation, in their respective order (i.e. “Animate ≤ Entity” and “Entity ≤ Entity”).

Another example is given in [7] in which the relations “PersonCharacteristic (Person, PersonAttribute)”, “TransportCharacteristic (Transport, TransportAttribute)” and “LocationCharacteristic (Location, LocationAttribute)” are specializations of the more generic relation “Characteristic(Entity,Attribute)”:

Characteristic(Entity,Attribute) > PersonCharacteristic(Person,PersonAttribute)

Characteristic(Entity,Attribute) > TransportCharacteristic(Transport,TransportAttribute)

Characteristic(Entity,Attribute) > LocationCharacteristic(Location,LocationAttribute)

3.1.2. Synonyms and terms of multiple senses

Concepts and relations are abstract ideas and words are just a way to represent them. An abstract idea may have multiple representations and conversely a representation may be used for multiple abstract ideas. This is the issue of synonyms and terms of multiple senses (a term is a word or group of words with a particular meaning).

When multiple terms are synonymous, we consider that they are just different representations of the same concept. Since we assume that the sets of concept and relation types do not contain duplicate elements, when we represent an ontology through a lattice, only one such term could be used for a particular concept type. However, all of the synonyms of that term should be available when we compare or merge the ontology with another. In implementations, each concept type should be represented by a term and a set of its synonyms, if exist.

Conversely, when a term has multiple senses, we consider that it represents different concepts and we should distinguish those concepts in the ontology. In implementations, such a term should be accompanied by another qualifier to distinguish between the many concepts that it represents. For example, in Wordnet, a large on-line semantic English dictionary [33], (which could be used as a canon in our formalism), the word “bird” has 5 different meanings, called *senses,* representing 5 different concepts. Consequently, each concept is represented by the word "bird", plus a “sense number”.

3.2. Ontology

We define an ontology $\boldsymbol{O}$ on a domain $\boldsymbol{M}$ with respect to a canon $\boldsymbol{K}=(T, I, \leq, conf, B)$ as a triple $\boldsymbol{O} = (T_{CO}, T_{RO}, I_O)$ whose elements are defined by statements (1) to (3) below:

(1) T_{CO} (resp. T_{RO}) is a subset of T_C (resp. T_R) and contains concept types (resp. relation types) relating to the domain $\boldsymbol{M}$. As mentioned earlier, without loss of functionality, we assume that T_{CO} (resp. T_{RO}) includes $\top$ and $\perp$.

(2) I_O is a subset of individual markers in I, that semantically relate to the domain $\boldsymbol{M}$.

(3) All other relations in $\boldsymbol{K}$ are carried over to $\boldsymbol{O}$, i.e.

- The subsumption relation: $\leq$: $(T_{CO} \text{ x } T_{CO}) \cup (T_{RO} \text{ x } T_{RO})$ -> {true, false}

- The conformity relation: *conf*: $I_O \backslash \{*\}$ -> T_{CO}
- The Canonical Basis function: *B*: T_{RO} -> $\varphi(T_{CO})$

Notes:

- The roles of a canon, vis-à-vis an ontology, are to ensure "semantic completeness" of types, i.e. for any pair of types, all possible relationships (i.e. subsumption relations) between them are defined in the canon, and to ensure "reasoning soundness" as its structure conforms to a lattice structure and thus facilitates mathematical reasoning over all objects defined under it.
- ***M*** is the domain name of the ontology. It is just a label to identify the nature of knowledge represented by the ontology and the label ***M*** by itself is not mathematically significant.
- Canon and ontology, as defined in this paper, also extend the definitions of *Node Hierarchy* and *Relation Hierarchy* of *Universal Data Structure* [16].

3.3. Conceptual Graph with respect to a canon

We define a Conceptual Graph ***G*** with respect to a canon ***K***=(T, I, $\leq$, *conf*, B) as a quintuple (or 5-tuple) ***G*** = (***C***, ***R***, *type, referent, arg*) whose elements are defined by statements (1) to (7) below:

(1) ***C*** is the set of concept nodes (or C-vertices) of ***G***. The set of concept types used in ***C*** is denoted T_{CG}. T_{CG} is a subset of T_C.

(2) ***R*** is the set of relation nodes (or R-vertices) of ***G***. The set of relation types used in ***R*** is denoted T_{RG}. T_{RG} is a subset of T_R.

Note that ***C*** and ***R*** may contain duplicate elements, while T_{CG} and T_{RG} do not. This is due to the different purposes of these objects. A conceptual graph could be used to represent a statement in which the same word (i.e. a concept node) may be repeated many times although the same word may figure only once in the dictionary (i.e. the canon).

(3) *type* is a function that associates a concept or a relation node with its type, i.e.:

$$type\text{:}\ \boldsymbol{C}\cup\boldsymbol{R}\ \textbf{->}\ T_{CG}\cup T_{RG}$$

with:

$$\forall \mathrm{C}\in\boldsymbol{C} \qquad type(\mathrm{C})\in T_{CG} \qquad \text{and} \qquad T_{CG} = type(\boldsymbol{C})$$

$$\forall \mathrm{R}\in\boldsymbol{R} \qquad type(\mathrm{R})\in T_{RG} \qquad \text{and} \qquad T_{RG} = type(\boldsymbol{R})$$

This means that *type* is a *surjective* function with respect to the set of concept and relation nodes and the set of concept and relation types.

(4) *referent* is a function that associates a concept with its referent (or individual marker), i.e.: $referent\text{:}\ \boldsymbol{C}\ \textbf{->}\ I_G$

where I_G is the set of distinct markers used in association with concept nodes in C. $\mathrm{I_G}$ is a subset of I.

Notes:

- A blank referent is the generic "*" marker.
- *referent* is also a *surjective* function with respect to the set of concept nodes and the set of individual markers.

(5) *arg* is a partial function that associates a relation node with its i-th concept argument, i.e.

$$arg\text{:}\quad \mathbf{N+}\ \mathrm{x}\ \boldsymbol{R}\ \textbf{->}\ \boldsymbol{C}$$

where:

- **N+** is the set of strictly positive natural numbers (or integers).
- $\forall \mathrm{i}\in\mathbf{N+}\ \ \forall \mathrm{R}\in\boldsymbol{R}$ *arg*(i,R) is the i-th (concept node) argument, if exists, in the *signature* of R (the signature of a relation node is the ordered list of concept nodes that are directly connected to that relation node). *arg*(i,R) is not defined when the i-th argument of the relation node does not exist

(hence, *arg* is a *partial* function).

(6) *referent* and *type* must satisfy the conformance relation *conf* of the canon $\boldsymbol{K}$, i.e.

$\forall C \in \boldsymbol{C}\ \forall i \in I_G\quad referent(C)=i \ \Rightarrow\ conf(i) \leq type(C)$

or $\forall C \in \boldsymbol{C}\quad conf(referent(C)) \leq type(C)$

This means that an individual marker can be used as referent in a number of concept nodes of different types but all those types are compatible and subsume the type assigned to that marker through the *conf* function. For example, "John" is an individual marker that can be used in a number of concept nodes: [Man: John], [Person: John], [LivingBeing: John], etc. But *conf*("John") = [Man] since the concept "Man" is subsumed in the other concepts: "Person", "Living Being", etc.

(7) *arg* and *type* must satisfy the Canonical Basis function (*B*) of the canon K, i.e.

$\forall i \in \mathbf{N+}\ \ \forall R \in \boldsymbol{R}\ \ \forall C \in \boldsymbol{C}\quad arg(i,R)=C \iff type(C)$ is the i-th element of $B(type(R))$

Notes:

- For a given relation node, the set of all of its concept arguments is called its *signature*. *signature* is therefore a function defined as:

  $signature:\quad \boldsymbol{R} \ \rightarrow\ \wp(\boldsymbol{C})$

  $\forall R \in \boldsymbol{R}\ \ signature(R) = \{arg(i,R)\ \forall i \in \mathbf{N+}$ where arg is defined$\}$

  Notes:

  - *signature*(R) is an ordered set and could contain duplicate elements.

  - The function *signature* is similar to the Canonical Basis function *B*, except that *signature* applies to relation nodes while *B* applies to relation types. *signature*(R) is like an "instance" of *B*(*type*(R)), just like R is an instance of *type*(R) (an instance could be defined as a realization in the real world of an abstract idea).

- *arity* (also called *valence*) is another function often used with a relation type or relation node. It associates a relation with the number of concepts linked by the relation, i.e. $arity$: $\boldsymbol{R}$ -> **N+**

3.4. Conceptual Graph with respect to an ontology

Given an ontology $\boldsymbol{O}$=(T_{CO}, T_{RO}, I_O) defined on a given domain $\boldsymbol{M}$ with respect to a canon $\boldsymbol{K}$=(T, I, ≤, *conf*, B), we define a Conceptual Graph $\boldsymbol{G}$ with respect to the ontology $\boldsymbol{O}$ as a Conceptual Graph $\boldsymbol{G}$ =($\boldsymbol{C}$, $\boldsymbol{R}$, *type, referent, arg*) defined with respect to the canon $\boldsymbol{K}$, with in addition:

(1) $T_{CG} \subseteq T_{CO}$

(2) $T_{RG} \subseteq T_{RO}$

(3) $I_G \subseteq I_O$

3.5. Other Definitions

- We define *nodes* as the function that associates a concept type or a relation type used in a conceptual graph, with its set of nodes in that conceptual graph. *nodes* is therefore the inverse of the function *type*, i.e. *nodes* = $type^{-1}$ or:

nodes: $T_{CG} \cup T_{RG}$ -> $\varphi(\boldsymbol{C}) \cup \varphi(\boldsymbol{R})$

with: $\forall c \in T_{CG}\ \forall C \in \boldsymbol{C}$ $C \in nodes(c) \Leftrightarrow type(C)=c$

$\forall r \in T_{RG}\ \forall R \in \boldsymbol{R}$ $R \in nodes(r) \Leftrightarrow type(R)=r$

Notes:

- $\forall c \in T_{CG}$, we can also define the value of *nodes*(c) recursively, as follows:

  nodes(c) = $\{C_1,..,C_n\}$

  $\forall i \in [1..n]$, C_i is defined as $C_i = C$ if $\exists C \in \boldsymbol{C}$ such that type(C) = c

  and $\forall i>1\ C \notin \{C_1,..,C_{i-1}\}$

- Similarly, we can define *nodes*(r) for each $r \in T_{RG}$.

- The set of concept nodes in *nodes*(c) and the set of relation nodes in *nodes*(r) are not ordered (since in the above recursive definitions, the conditions ∃C∈***C*** and ∃R∈***R*** do not impose an order on the nodes that existent).
- *nodes*(c) and *nodes*(r) cannot be empty sets, as per definitions of a concept node and a relation node.
- A "concept node" in a conceptual graph is different from a "concept node" in a lattice of concept types (e.g. as used in a canon or an ontology) as the latter does not contain a referent and the link between any two nodes is the subsumption relation between them while a direct link between any two concept nodes in a conceptual graph is not possible as they can only be linked via a relation node (the conceptual graph is therefore called *bipartite*). Similar differences hold for relation nodes in a conceptual graph and relation nodes in a lattice of relation types.

4. PROJECTIONS

In order to compare knowledge represented by different canons, different ontologies or different conceptual graphs, the notion of *projection* is introduced. Initially projection was applicable to conceptual graphs only [21]. Since then there have been many other formalizations of projection, in particular in [5], [6], [7], [8], etc. We further extend the concept into ontology with a view of applying our formalism to the particular issue of ontology merging.

To simplify the notations in the rest of this paper:

- Whenever we mention *a conceptual graph* ***G*** (without further qualification), we mean *a conceptual graph* ***G*** *= (**C**, **R**, type, referent, arg) defined with respect to a canon* ***K*** = (*T*, *I*, ≤, *conf*, *B*).

- When there is no confusion, we also write *type* for the functions *type'*, *type"*, etc., and the same applies to the functions *referent* and *arg*, i.e. we write ***G'*** *=* *(**C'**, **R'**, type, referent, arg)* instead of ***G'*** *=* *(**C'**, **R'**, type', referent', arg'),* etc.
- We denote:
  - by ***C*** (upper case, bold, italic), the set of all concept nodes of a conceptual graph.
  - by C (upper case, not bold, not italic), an element of that set.
  - by c (lower case, not bold, not italic), the concept type of C.

Unless explicitly stated, whenever we simply write: C, we mean: C∈***C*** (e.g. ∀C means ∀C∈***C***). The same convention applies to ***R*** (upper case - bold - italic), R (upper case - not bold - not italic), r (lower case, not bold, not italic) and their variants, such as ***C'***, C', c', etc.

4.1. Concept Projection

Concept projection is used to find whether two concepts are semantically compatible or similar.

Concept type projection is a relation between 2 concept types, belonging to two ontologies defined with respect to the same canon. *Concept node projection* is a relation between 2 concept nodes, belonging to two conceptual graphs defined with respect to the same canon.

4.1.1. Concept projection in ontology

Given two concept types belonging to two ontologies, ***O*** and ***O'***, defined with respect to the same canon, c∈T_{CO} and c'∈$T_{CO'}$, c is said to have a projection into c', and we write c' = *proj*(c), if c'≤c (this makes sense as both types belong to T_C and could be associated through the global relation ≤ of ***K***).

4.1.2. Concept projection in conceptual graph

Given 2 conceptual graphs, ***G*** and ***G'***, both defined with respect to the same canon ***K***, and 2 concept nodes C and C' belonging to those conceptual graphs, C is said to have a projection into

C’ or C’ is a projection of C, and we write C’ = *proj*(C), if the following 2 conditions are satisfied:

(1) *type*(C’) ≤ *type*(C)

and (2) *referent*(C)=*referent*(C’) or *referent*(C)=*

Note:

- ***G*** and ***G’*** may belong to the same or different ontologies but all must be defined with respect to the same canon ***K*** (so that the above use of relation ≤ between two types belonging to two different conceptual graphs makes sense).

4.2. Relation Projection

Relation projection is used to find whether two relations are semantically compatible or similar. Relation projection is defined similarly to concept projection.

4.2.1. Relation projection in ontology

$\forall r \in T_{RO} \quad \forall r' \in T_{RO'} \qquad r' = proj(r) \quad \Leftrightarrow \quad r' \leq r$

4.2.2. Relation projection in conceptual graph

$\forall R\ \forall R'$ R’ = *proj*(R) if the following 2 conditions are satisfied:

(1) *type*(R’) ≤ *type*(R) and

(2) $\forall i \in \mathbf{N+}\ \forall C_i \in signature(R)\ \forall C'_i \in signature(R')\ \ C'_i = proj(C_i)$

The first condition implies that R and R’ have the same arity and the concept types associated with R’ are subsumed in the corresponding concept types associated with R (as per (6) of Section 3.1). The second condition implies that in addition for any concept node argument of R’, the corresponding concept node argument of R must have the same referent or the referent of the latter must be generic.

Note that the above second condition can also be equivalently expressed as:

$\forall i \in \mathbf{N+}\ \forall C_i \in signature(R)\ \forall C'_i \in signature(R')\ \ C'_i = proj(C_i)$

$\Leftrightarrow \quad \forall i \in \mathbf{N+}\ \ \forall C\ \ arg(i,R)=C \Rightarrow arg(i,R')=proj(C)$

$\Leftrightarrow \quad \forall i \in \mathbf{N+}\ \ arg(i,R')=proj(arg(i,R))$

$\Leftrightarrow \quad (\forall i \in \mathbf{N+}\ \ type(arg(i,R')) \leq type(arg(i,R))\ and$

$(referent(arg(i,R'))=referent(arg(i,R))\ or\ referent(arg(i,R))=*)\ )$

4.3. Conceptual Graph Projection

Conceptual graph projection is used to find whether two conceptual graphs are semantically compatible or similar.

Given 2 conceptual graphs, ***G*** and ***G'***, defined with respect to the same canon, ***G*** is said to have a projection into ***G'***, and we write ***G'***=*proj*(***G***), if every concept node and every relation node in ***G*** has a projection in ***G***'.

The above definition also extends the *Conceptual Graph Hierarchy* of *Universal Data Structure* [16].

4.4. Ontology Projection

Ontology projection is used to find whether two ontologies are semantically compatible or similar. This is a new definition first proposed in this paper.

Given 2 ontologies, ***O*** and ***O'***, defined with respect to the same canon ***K***, ***O*** is said to have a projection into ***O'***, and we write ***O'***=*proj*(***O***), if every concept type and every relation type in ***O*** has a projection in ***O'*** and in addition the set of individual markers of ***O'*** contains that of ***O*** (i.e. $I_O \subseteq I_{O'}$).

Theorem 1. *If an ontology **O** has a projection into an ontology **O'**, then for any conceptual graph defined with respect to **O**, there exists a conceptual graph **G'** defined with respect to **O'** and which is a projection of **G**. In addition, we can select **G'** so that it is condensed (see*

Section 5.7 for the formal definition of condensed graph).

Proof. Please refer to Appendix in Supplemental Material.

Interpretation. In simple terms, Theorem 1 means that if the semantics of one ontology is included in that of another ontology, then for any statement made from the first ontology, one can build a more condensed statement in the second ontology that includes the semantics of the initial assertion. In other words, ontology projection could be used to "condense" knowledge representation through conceptual graphs.

5. CANONICAL GRAPH

A *Canonical Graph* is a conceptual graph, derived from other conceptual graphs (defined with respect to the same canon ***K***), through the use of one or many of the following operations (called *Canonical Formation Rules*): External Join, Internal Join, Type Restriction, Referent Restriction, Copy, Simplification and Condensation. The Canonical Graph is then said to be the "closure" of the starting conceptual graphs under the canonical formation rules. Canonical graphs are usually the results of interoperability operations on conceptual graphs, such as merging, simplification, restriction, etc., which rely on canonical formation rules. Interoperability between ontologies is not discussed in detail in this paper, but could be found in a large number of disciplines, for example, [5], [6] for CST and [25], [26], [27] for FCA, and [19] for Description Logic, just to name a few. This section is an enhanced formalization of canonical formation rules mainly based on [21] and [5].

Note that canonical formation rules only apply to conceptual graphs and not to ontologies and canons.

5.1. Notations and Definitions

- Given two different concept nodes C and C' belonging to the same conceptual graph or to two different conceptual graphs but defined with respect to the same canon, we say that C' is a *duplicate concept node* (or simply *duplicate*) of C, or C is a *duplicate concept node* of C', and we write C ≡ C', if they have the same type and same referent, i.e.

$\forall C\ \ \forall C'\ \ C \equiv C' \iff$ *type*(C)=*type*(C') and *referent*(C)=*referent*(C')

- Given a conceptual graph ***G***, we define *dup*(***C***) as the set of all concept nodes in ***C***, each of which has at least a duplicate in ***C***, i.e.:

  dup(***C***) = $\{D \in \boldsymbol{C} \mid \exists C\ \ C \neq D \text{ and } C \equiv D\}$

- Given two different conceptual graphs ***G*** and ***G'*** defined with respect to the same canon, we define ***C∩C'*** as the set of all concept nodes in ***C***, each of which has at least a duplicate in ***C'***.

  $\boldsymbol{C} \cap \boldsymbol{C'} = \{D \in \boldsymbol{C} \mid \exists C' \in \boldsymbol{C'} \text{ with } C' \equiv D\}$

  Note that it follows from the definition that ***C∩C'*** and ***C'∩C*** are 2 different sets (i.e. $\boldsymbol{C} \cap \boldsymbol{C'} \subseteq \boldsymbol{C}$ while $\boldsymbol{C'} \cap \boldsymbol{C} \subseteq \boldsymbol{C'}$).

- Similarly, we can define duplicate relation nodes as follows:

  (1) $\forall R\ \ \forall R'$ such that $R \neq R'$, we define:

  $R \equiv R' \iff$ *type*(R)=*type*(R')

  and *arity*(R)=*arity*(R')

  and $\forall i \in [1..$*arity*(R)]

  arg(i,R)=*arg*(i,R') (when R and R' are in the same conceptual graph and their i-th arguments are linked to the same concept node)

  or *arg*(i,R) ≡ *arg*(i,R') (when R and R' are in 2 different conceptual graphs or when they are in same graph but with their i-th arguments are linked to duplicate concept nodes)

  (2) *dup*(***R***) = $\{S \in \boldsymbol{R} \mid \exists R\ \ R \neq S \text{ and } R \equiv S\}$

(3) $\boldsymbol{R}\cap\boldsymbol{R'} = \{S\in\boldsymbol{R} \mid \exists R' \; R' \equiv S\}$

Property 1. *If a conceptual graph contains a duplicate relation node pair then it contains at least a duplicate concept node pair.*

Proof. Please refer to Appendix in Supplemental Material.

Interpretation. There would be no relation node duplication without concept node duplication in the first instance.

5.2. External Join

External join is an operation to combine two conceptual graphs into a single graph. This is a formalization of the definition given in [5] and [6].

Given 2 conceptual graphs, ***G*** and ***G'***, both defined with respect to the same canon ***K***, the conceptual graph ***G"*** = (***C"***, ***R"***, *type", referent", arg"*) is said to be the *external join* of ***G*** and ***G'*** when the following conditions (1) to (5) hold:

(1) $\boldsymbol{C''} = \boldsymbol{C} \cup(\boldsymbol{C'}\backslash(\boldsymbol{C'}\cap\boldsymbol{C}))$

This simply means that the concept node set of ***G"*** is the union of the two starting sets of concept nodes, in which we remove all the nodes in ***C'*** that have a duplicate in ***C***.

(2) $\boldsymbol{R''} = \boldsymbol{R}\cup(\boldsymbol{R'}\backslash(\boldsymbol{R'}\cap\boldsymbol{R}))$

The above remark on ***C"*** also applies here for ***R"***.

(3) The partial function *arg"* is defined as follows:

$\forall i\in \boldsymbol{N}+ \quad \forall R \;\; arg''(i,R) = arg(i,R)$

$\forall R'\in\boldsymbol{R'}\backslash(\boldsymbol{R'}\cap\boldsymbol{R})$

- if $arg'(i,R')\notin\boldsymbol{C'}\cap\boldsymbol{C}$, we define $arg''(i,R') = arg'(i,R')$
- if $arg'(i,R')\in\boldsymbol{C'}\cap\boldsymbol{C}$, then by virtue of Property 1, $\exists C$ such that $arg'(i,R') \equiv C$ (multiple such C may exist but we only select one) and we define in this case

$arg''(\text{i,R'}) = \text{C}$

This definition means that *arg"=arg* when defined on the relation nodes of the first graph, *arg"=arg'* when defined on the relation nodes of the second graph except when their concept values are common to both starting concept node sets, in which case their values are replaced by the duplicate concept nodes in the first graph.

(4) $\forall \text{C}$ $\quad type''(\text{C}) = type(\text{C})$

$\forall \text{C'} \in \boldsymbol{C'} \backslash (\boldsymbol{C'} \cap \boldsymbol{C'})$ $\quad type''(\text{C'}) = type'(\text{C'})$

(5) $\forall \text{C}$ $\quad referent''(\text{C}) = referent(\text{C})$

$\forall \text{C'} \in \boldsymbol{C'} \backslash (\boldsymbol{C'} \cap \boldsymbol{C})$ $\quad referent''(\text{C'}) = referent'(\text{C'})$

Property 2. *If* ***G''*** *is the external join of* ***G*** *and* ***G'****, then we have:*

$$T_{CG''} = T_{CG} \cup (T_{CG'} \backslash (T_{CG'} \cap T_{CG}))$$

and $$T_{RG''} = T_{RG} \cup (T_{RG'} \backslash (T_{RG'} \cap T_{RG}))$$

Proof. Please refer to Appendix in Supplemental Material.

Interpretation. When two conceptual graphs are externally joined, the duplicate concept and relation types in the 2nd graph are removed.

5.3. Internal Join

Internal join is an operation to remove all concept node duplicates in a conceptual graph. This is a formalization of the definition given in [5] and [6].

Given a conceptual graph $\boldsymbol{G}$ and a concept node $\text{C} \in \boldsymbol{C}$ for which multiple duplicates may exist, a conceptual graph $\boldsymbol{G'}$ is said to be an *internal join of* ***G*** *by C* when the following conditions (1) to (5) hold:

(1) $\boldsymbol{C'} = \boldsymbol{C} \backslash dup(\text{C})$

(2) $\boldsymbol{R'} = \boldsymbol{R}$ with arguments of all relation nodes modified by the function *arg'* as follows.

(3) *arg'* is defined as:

$\forall$R' $\forall$i$\in$[1..*arity*(R')]

arg'(i,R') = *arg*(i,R') if *arg*(i,R')$\notin$*dup*(C)

arg'(i,R') = C if *arg*(i,R')$\in$*dup*(C)

(4) *type'* = *type*$\lfloor_{C'}$

(5) *referent'* = *referent*$\lfloor_{C'}$

Note that if in particular C has no duplicates then ***G'*** = ***G***.

Given a conceptual graph ***G***, a conceptual graph ***G'*** is said to be the *internal join of* ***G,*** if ***G'*** is the internal join of ***G*** by C for all C$\in$***C***.

5.4. Type Restriction

Type restriction is an operation to *specialize* a concept type in a conceptual graph, i.e. to replace a concept type with a more specialized one. This is a formalization of the definition given in [5] and [6].

Given a conceptual graph ***G*** and two concept types c$\in T_{CG}$ and d$\in T_{CG}$ with d≤c and d≠⊥, a conceptual graph ***G'*** is said to be a *type restriction of* ***G*** *from* c *to* d when the following conditions (1) to (4) hold:

(1) ***C'*** = ***C*** with the types of all concept nodes modified by the function *type'* as follows.

$\forall$C'$\in$***C'*** *type'*(C') = d if *type*(C')=c

type'(C')= *type*(C') otherwise

This means that we simply replace type c by type d in all concept nodes whose type is c.

(2) ***R'*** = ***R***

(3) *referent'* = *referent*

(4) *arg'*= *arg*

Note that the concept node arguments are also affected by Condition (1), i.e. if a concept node argument of a relation is of type c then it is changed to type d.

5.5. Referent Restriction

Referent restriction is an operation to *specialize* the referent of some concept nodes in a conceptual graph, i.e. to replace the generic referent by an individual marker, in all concept nodes whose referent is generic and whose type subsumes the type of the individual marker. This is a formalization of the definition given in [5] and [6].

Given a conceptual graph $\boldsymbol{G}$ and an individual marker $i \in I_G$, a conceptual graph $\boldsymbol{G'}$ is said to be a *referent restriction of* $\boldsymbol{G}$ *to i* when the following conditions (1) to (4) hold:

(1) $\boldsymbol{C'} = \boldsymbol{C}$ with the referents of all concept nodes modified by the function *referent'* as follows:

$$\forall C' \quad \begin{array}{ll} \mathit{referent'}(C') = i & \text{if } \mathit{conf}(i) \leq \mathit{type}(C') \text{ and } \mathit{referent'}(C') = * \\ \mathit{referent'}(C') = \mathit{referent}(C') & \text{otherwise} \end{array}$$

This means that we replace the generic referent with the given individual marker *i* in all concept nodes whose referent is generic and whose type subsumes the type associated with *i* through the conformity relation.

(2) $\boldsymbol{R'} = \boldsymbol{R}$

(3) $\mathit{type'} = \mathit{type}$

(4) $\mathit{arg'} = \mathit{arg}$

Note that the concept node arguments are also affected by Condition (1), i.e. if the type of a concept node argument subsumes the type *conf(i)* and its referent is generic, then we replace that concept node argument with the one whose referent is *i*.

5.6. Copy

Copy is an operation that duplicates a conceptual graph. This is a formalization of the definition given in [21].

Given a conceptual graph ***G***, a conceptual graph ***G'*** is said to be a *copy of **G*** when the following conditions (1) to (4) hold:

(1) $\forall C \ \ \exists C'$ (and only 1 such C') such that $C' \equiv C$

(2) $\forall C' \ \ \exists C$ (and only 1 such C) such that $C \equiv C'$

(3) $\forall R \ \ \exists R'$ (and only 1 such R') such that $R' \equiv R$

(4) $\forall R' \ \ \exists R$ (and only 1 such R) such that $R \equiv R'$

Note that the use of the equivalence relation ($\equiv$) in the above implies that *type'*, *referent'* and *arg'* are also defined through that relation.

The above conditions simply mean that each concept node and each relation node in the original conceptual graph are duplicated, once and only once, in the second conceptual graph.

5.7. Simplification

Simplification is an operation to remove duplicate relation nodes in a conceptual graph. This is a formalization of the definition given in [21].

Given a conceptual graph ***G*** and a relation node $R \in \boldsymbol{R}$ for which multiple duplicates may exist, a conceptual graph ***G'*** is said to be a *simplification of **G** with respect to the relation node* R when the following conditions (1) to (5) hold:

(1) $\boldsymbol{R'} = \boldsymbol{R} \setminus dup(R)$

(2) $\boldsymbol{C'} = \boldsymbol{C} \setminus \{ C \in \boldsymbol{C} \mid \exists R' \in dup(R) \ \exists i \in [1..arity(R')] \ C = arg(i,R') \text{ and } C \equiv arg(i,R) \text{ and}$

$C \neq arg(i,R) \text{ and } \forall R'' \in \boldsymbol{R'} \ \forall j \in \mathbf{N+} \ \ C \neq arg(j,R'') \}$

(3) $type' = type\lfloor_{C'}$

(3) $referent' = referent\lfloor_{\mathbf{C'}}$

(5) $arg' = arg\lfloor_{\mathbf{N+} \times \mathbf{R'}}$

Condition (2) means that we remove from ***C*** all the duplicate concept nodes which are arguments of duplicates of R and which are not linked to any other relation nodes.

A simplification with respect to R is obtained by removing relation nodes which are duplicates of R, and by removing duplicate concept nodes, which are arguments of those duplicate relation nodes and which are not used elsewhere (i.e. as arguments in other relation nodes). Simplification does not mean that all duplicate *concept* nodes are removed, as there may be duplicate concept nodes, which do not belong to (or are not arguments of) duplicate relation nodes. Simplification is similar to internal join, but applied to relation nodes.

Given a conceptual graph ***G***, a conceptual graph ***G'*** is said to be a *simplification of* ***G*** (or *simplified graph of* ***G***) if $\forall$R, ***G'*** is a simplification of ***G*** with respect to R.

Property 3. *If a conceptual graph could be simplified then it could be internally joined.*

Proof. Please refer to Appendix in Supplemental Material.

Interpretation. Removing a duplicate relation node in a conceptual graph implies removing at least a duplicate concept node in that graph.

5.7. Condensation

This terminology is borrowed from the terms: *thematic* or *textual condensation*, a concept mainly explored in semiotics [12][24] but it is first introduced in this paper in the context of conceptual graphs.

Given a conceptual graph ***G***, a conceptual graph ***G'*** is said to be a *condensation of* ***G*** (or *condensed graph of* ***G***) if ***G'*** is both a simplification of ***G*** and an internal join of ***G***.

Condensation is the operation that removes all duplicate concept nodes and duplicate relation nodes in a conceptual graph.

Theorem 2. *Each canonical formation rule (i.e. external join, internal join, type restriction, referent restriction, copy, simplification and condensation) is a homomorphism with regard to the conceptual graph projection operation, that is if* ***G*** *has a projection into* ***H*** *and* ***G'*** *has a projection into* ***H'*** *then the external join (or any other canonical formation rule) of* ***G*** *and* ***G'*** *has a projection into the external join (or any other same canonical formation rule) of* ***H*** *and* ***H'***.

Proof. Please refer to Appendix in Supplemental Material.

Interpretation. In simple terms, Theorem 2 means that if a set of statements in one ontology is semantically included in a set of statements of another ontology, then for any inference obtained from the first set, one could find a similar inference in the second set, that semantically contains the first inference. In other words, reasoning on conceptual graphs is preserved through their projection.

6. COMPARISON WITH FCA

A few similarities and differences between our formalism and other theories, in particular FCA, have been mentioned in previous sections. High-level comparison between CST and FCA at the philosophical, implementational, logical, epistemological, conceptual and linguistical levels has been explored in [17]. However, there are other points concerning FCA and CST, that should also be noted:

(1) Our definitions of canon and ontology are similar to the notion of *formal context* in FCA [10][31]. In FCA, a formal context as a relational structure K=(G,M,I) where G is a set of *formal objects*, M, a set of *formal attributes* and I, a binary relation between G and M indicating which formal object has which formal attribute.

(2) To assist with the development of an FCA-based technique to merge ontologies (called "FCA-Merge"), G. Stumme provides formal definitions of "core ontology" (which does not include individual markers) and of "knowledge base" (which includes individual markers), and defines linkages between them [25][26]. Our approach is similar, although we merge those two notions into the definition of "canon", with "ontology" defined as a subset of a canon. FCA does not have the exact equivalence of the definition of a canon which encompasses other ontologies (i.e. like a universal ontology) as given in this paper.

(3) In [31], it is suggested that FCA provide a means to complete the formalization of the three *Elements* of Immanuel Kant's *Logic*, or *Elementary Logic* [14]. Kant attributes three main functions to human thinking: *concept*, *judgment* and *inference*. According to Kant, a concept is a representation of what is common to several objects, a judgment is a synthesis of concepts and intuitions, or, of concepts and concepts and an inference or conclusion is the deduction of one judgment from another. In [31], it is contented that FCA could be a formalization of Kant's *concepts*, Sowa's conceptual graphs, a formalization of Kant's *judgments*, and Sowa's canonical graphs, a formalization of Kant's *inferences*. We believe that the formalism proposed in this paper is also an alternative to FCA in formalizing Kant's *concepts* (especially with our mathematization of canon and ontology), as well as enhances the formalization of the two other Kant's logic elements. Thus our formalism enables CST to completely represent the three Elements of Kant's Logic, without the need to use an external theory to complement it.

(4) Our definition of canonical formation rules guarantees that the derived conceptual graphs (the canonical graphs) also conform to the rules of the same canon to which the source conceptual graphs belong, i.e. the canonical graphs formed by our rules are also defined

with respect to the same canon. In other words, if the starting conceptual graphs are meaningful within a canon, then the resulting canonical graphs are also meaningful within the same canon. Note that "meaningful" (i.e. we can understand its meaning in a natural language) does not mean "true in real life", i.e. canonical graphs, produced from semantically true conceptual graphs, may not be semantically true. For example, the external join of the two conceptual graphs representing the two true sentences: "Some people are tall" and "Some people are short" is: "Some people are both tall and short", which of course is not true in real life, although meaningful. Canonical formation rules however guarantee that one can never end up with a non-sensical conclusion such as Chomsky's famous syntactically-correct but meaningless sentence: "Colorless green ideas sleep furiously". In brief, canonical formation rules are syntactical rules as well as inference rules, although not fully deductive inference rules (In philosophy, deductive inference guarantees the truth of conclusions from true premises while inductive inference does not always - see e.g. [15]). This converges with Elementary Logic in which it is agreed that true judgments and true concepts may not necessarily lead to true conclusions [14].

7. FUTURE WORK

Mathematical properties of the various objects presented in this paper could be further explored, in particular in relation to graph projection, canonical formation, and ontology merging. This would strengthen the mathematical foundation of the Conceptual Structure Theory and greatly assist representation and manipulation of human knowledge. It would also be useful to compare, link and integrate notions presented in this paper with similar work under other approaches, such as S-graph and Simple Conceptual Graph [1][2][3][4][18], FCA [10][25][31], Universal Data

Structure [16], Semantic Web and Description Logic [19], just to name only a few. And finally, it would be of interest to apply the results of this paper to real knowledge engineering applications such as ontology merging and semantic web development.

8. CONCLUSION

In this paper, canon, ontology and conceptual graph, as well as their manipulation through projection and canonical formation operations, are formally re-defined to broaden various previous definitions by other authors. Projection operations enable comparison of concepts, relations and their hierarchies while canonical formation operations help with inferences from existing knowledge and discovery of new knowledge. The derived mathematical properties should assist future work in research and development on knowledge representation, in particular, in the area of ontology interoperability, which has many real life applications. Our proposed formalism also enables CST to be compared favorably with FCA in formalizing the three main elements of human thinking, as described in Kant's Logic: the rigorous mathematization of basic objects encountered in the Conceptual Graph Theory, examined under the light of formal definitions of canon and ontology, offers a powerful formalization of Kant's doctrine of concepts; the enhanced mathematical definitions of conceptual graph and its projection improves the formalization of Kant's doctrine of judgments; and finally, the description of new mathematical properties for conceptual graph manipulation through canonical formation rules completes the formalization of Kant's doctrine of inferences.

ACKNOWLEDGMENT

The authors wish to thank the anonymous IEEE reviewers for their comments, which have contributed to the enhancement of this paper.

REFERENCES

[1] J.F. Baget and M.L. Mugnier, “The SG Family: Extensions of Simple Conceptual Graphs”, Proc. 17th International Joint Conference on Artificial Intelligence, 2001, pp. 205-210.

[2] J.F. Baget and M.L. Mugnier, “Extensions of Simple Conceptual Graphs: the Complexity of Rules and Constraints”, Journal of Artificial Intelligence Research, Vol. 16, 2002, pp. 425-465.

[3] M. Chein and M.L. Mugnier, “Conceptual Graphs: Fundamental Notions”, Revue d’Intelligence Artificielle, 1992.

[4] M. Chein and M.L. Mugnier, “Concept Types and Coreference in Simple Conceptual Graphs”, Proc. 12th International Conference on Conceptual Structures, 2004.

[5] D. Corbett, “Reasoning and Unification over Conceptual Graphs”, New York, Kluwer Academic Publishers, 2003.

[6] D. Corbett, “Filtering and Merging Knowledge Bases: a Formal Approach to Tailoring Ontologies”, Revue d'Intelligence Artificielle, Special Issue on Tailored Information Delivery, Cecile Paris and Nathalie Colineau (Eds.), Sept/Oct 2004, pp. 463 – 481.

[7] M. Croitoru and E. Compatangelo, “A Combinatorial Approach to Conceptual Graph Projection Checking”, to appear in Proc. 24th SGAI International Conference on Innovative Techniques and Applications of Artificial Intelligence, Dec. 2004.

[8] M. Croitoru and E. Compatangelo, “On Conceptual Graph Projection”, Technical Report AUCS/TR0403, University of Aberdeen, UK, 2004.

[9] F. Dau, “Concept Graphs without Negations: Standardmodels and Standardgraphs”, 11th International Conference on Conceptual Structures, 2003.

[10] B. Ganter and R. Wille, “Formal Concept Analysis: Mathematical Foundations”, Springer, Heidelberg, Germany, 1996 (German version - English translation, 1999).

[11] B. Ganter and R. Wille, “Applied Lattice Theory: Formal Concept Analysis”, Technical University of Dresden, Germany, 1997, available at:

http://www.math.tu-dresden.de/~ganter/psfiles/concept.ps.

[12] A.J. Greimas, “Semantique Structurale: Recherche de Methode”, Ed. Larousse, Paris, 1966 (reprinted Nouvelle Edition Paris, 1986) (English translation: “Structural Semantics: An Attempt at a Method”, Lincoln University of Nebraska Press, 1983).

[13] T. Gruber, “A Translation Approach to Portable Ontology Specifications”, in: “Knowledge Acquisition”, Vol. 5(2), pp.199-220, 1993.

[14] I. Kant, “Logic”, first published in German in 1800, English translation by R. Hartman and W. Schwarz, Bobbs-Merrill Company, 1974. See also:

http://www-philosophy.ucdavis.edu/kant/CONCEPT.HTM and

http://www-philosophy.ucdavis.edu/kant/JUDGMENT.HTM

[15] G. Kemerling, “Arguments and Inference”, The Philosophy Pages:

http://www.philosophypages.com/lg/e01.htm, 2001.

[16] R. Levinson, “UDS: A Universal Data Structure”, Research Report No. UCSC-CRL-94-15, University of California – Santa Cruz, USA, 1994, also available at:

http://citeseer.ist.psu.edu/levinson94uds.html.

[17] G. Mineau, G. Stumme and R. Wille, “Conceptual Structures Represented by Conceptual Graphs and Formal Concept Analysis ”, Proc. 7th International Conference on Conceptual Structures, 1999.

[18] M.L. Mugnier, “Knowledge Representation and Reasonings based on Graph Homomorphism”, Proc. 8th International Conference on Conceptual Structures, 2000.

[19] N. Noy and M. Musen, "The PROMPT Suite: Interactive Tools For Ontology Merging And Mapping", International Journal of Human-Computer Studies, 2003.

[20] A. Puder and A. Alireza, "Service Type Specification through Conceptual Graphs", Workshop on Visual Reasoning, 13th International Conference on Automated Deduction, Rutgers University, 1996.

[21] J. Sowa, "Conceptual Structures – Information Processing in Mind and Machine", Addison-Wesley, 1984.

[22] J. Sowa, "Knowledge Representation: Logical, Philosophical, and Computational Foundations", Brooks Cole Publishing Co., Pacific Grove, CA, 1999.

[23] J. Sowa, "Conceptual Graphs", Draft proposed ISO standard for Conceptual Graphs, 2001, available at: http://www.jfsowa.com/cg/cgstand.htm.

[24] P. Stockinger, "Conceptual Analysis and Knowledge Management", Report from Equippe Semiotique Cognitive et Nouveaux Medias, Maison des Sciences de l'Homme, Paris, 1993.

[25] G. Stumme, "Using Ontologies and Formal Concept Analysis for Organizing Business Knowledge", in "Wissensmanagement mit Referenzmodellen - Konzepte für die Anwendungssystem und Organisationsgestaltung", J. Becker, R. Knackstedt (Eds.), Physica, Heidelberg 2002, pp. 163-174.

[26] G. Stumme and A. Maedche, "FCA-Merge: Bottom-Up Merging of Ontologies", Proc. 7th International Conference on Artificial Intelligence, 2001.

[27] G. Stumme, R. Studer and Y. Sure, "Towards an Order-Theoretical Foundation for Maintaining and Merging Ontologies", University of Karlsruhe, Germany, 2000, available at: http://www.aifb.uni-karlsruhe.de/WBS/gst/papers/2000/REFMOD00.pdf

[28] G. Stumme, R. Wille, U. Wille, "Conceptual Knowledge Discovery in Databases Using

Formal Concept Analysis Methods", in "Proceedings of the Second European Symposium on Principles of Data Mining and Knowledge Discovery", Springer-Verlag, 1998, pp. 450 – 458.

[29] M. Wermelinger and J.G. Lopes, "Basic Conceptual Structures Theory", Proc. 2nd International Conference on Conceptual Structures, 1994.

[30] R. Wille, "Restructuring lattice theory: an approach based on hierarchies of concepts", in "Ordered Sets", I. Rival (ed.), Reidel, Dordrecht-Boston, 1982.

[31] R. Wille, "Conceptual Graphs and Formal Concept Analysis", Proc. 5th International Conference on Conceptual Structures, 1997.

[32] R. Wille, "Why Can Concept Lattices Support Knowledge Discovery in Databases?", Proc. 9th International Conference on Conceptual Structures, 2001.

[33] Wordnet, V.2.0, Cognitive Science Laboratory, Princeton University, USA, available at: http://wordnet.princeton.edu

Philip H.P. Nguyen received the MS degree in pure mathematics from the University of Grenoble, France, the postgraduate degree in economical mathematics, the postgraduate degree in industry management, and the PhD degree in information technology, respectively, from the University of Paris, France. He is currently a principal technical specialist for the Department of Justice of the Government of South Australia.

Dan Corbett, Ph.D., was recently the Director of the Intelligent Systems Laboratory, University of South Australia. He is currently the Principal Scientist, Artificial Intelligence, for Science Applications International Corporation, an engineering and IT Research and Development company in Washington, DC, USA.

Supplemental Material to "A Basic Mathematical Framework for Conceptual Graphs"

APPENDIX

This appendix contains proofs of all properties and theorems mentioned in the paper.

Property 1. *If a conceptual graph contains a duplicate relation node pair then it contains at least a duplicate concept node pair.*

Proof. Indeed, if R and R' are a duplicate pair of relation nodes, then by definition:

$$\forall i \in [1..arity(R)]\ arg(i,R) = arg(i,R') \text{ or } arg(i,R) \equiv arg(i,R')$$

Since R must be different from R' by definition of duplicate, it ensues that there must exist at least an argument of R which is different from, but is a duplicate concept node of, the equivalent argument of R', i.e. $\exists j \in [1..arity(R)]$ with $arg(j,R) \equiv arg(j,R')$ and $arg(j,R) \neq arg(j,R')$ □

Property 2. *If **G''** is the external join of **G** and **G'**, then we have:*

$$T_{CG''} = T_{CG} \cup (T_{CG'} \setminus (T_{CG'} \cap T_{CG}))$$

and $$T_{RG''} = T_{RG} \cup (T_{RG'} \setminus (T_{RG'} \cap T_{RG}))$$

Proof. Indeed, to prove $T_{CG''} = T_{CG} \cup (T_{CG'} \setminus (T_{CG'} \cap T_{CG}))$, we must prove:

$$(1)\ \forall c\ \ c \in T_{CG''} \Rightarrow c \in T_{CG} \cup (T_{CG'} \setminus (T_{CG'} \cap T_{CG}))$$

and $$(2)\ \forall c\ \ c \in T_{CG} \cup (T_{CG'} \setminus (T_{CG'} \cap T_{CG})) \Rightarrow c \in T_{CG''}$$

- To prove (1):

$c \in T_{CG''} \Rightarrow \exists C \in \boldsymbol{C} \cup (\boldsymbol{C'} \setminus (\boldsymbol{C'} \cap \boldsymbol{C}))$ with $type(C) = c$ or $type'(C) = c$ (by definition of $T_{CG''}$)

b) If $C \in \boldsymbol{C}$ then: $type(C) \in T_{CG}$. Since $T_{CG} \subseteq T_{CG} \cup (T_{CG'} \setminus (T_{CG'} \cap T_{CG}))$, we obtain:

$$c = type(C) \in T_{CG} \cup (T_{CG'} \setminus (T_{CG'} \cap T_{CG}))$$

c) If $C \in \boldsymbol{C'} \setminus (\boldsymbol{C'} \cap \boldsymbol{C})$ then we have: $type'(C) \in T_{CG'}$ (since $\boldsymbol{C'} \setminus (\boldsymbol{C'} \cap \boldsymbol{C}) \subseteq \boldsymbol{C'}$)

If *type'*(C)$\in T_{CG'}$ then we have either: *type'*(C)$\in T_{CG'}\backslash(T_{CG'}\cap T_{CG})$

or: *type'*(C)$\in T_{CG'}\cap T_{CG}$

- If *type'*(C)$\in T_{CG'}\backslash(T_{CG'}\cap T_{CG})$, since $T_{CG'}\backslash(T_{CG'}\cap T_{CG})\subseteq T_{CG}\cup(T_{CG'}\backslash(T_{CG'}\cap T_{CG}))$,

  it follows that c= *type'*(C)$\in T_{CG}\cup(T_{CG'}\backslash(T_{CG'}\cap T_{CG}))$

- If *type'*(C)$\in T_{CG'}\cap T_{CG}$, then $\exists$d$\in T_{CG}$ such that *type'*(C)=d.

  Since $T_{CG}\subseteq T_{CG}\cup(T_{CG'}\backslash(T_{CG'}\cap T_{CG}))$, we obtain

  d$\in T_{CG}\cup(T_{CG'}\backslash(T_{CG'}\cap T_{CG}))$

  or c$\in T_{CG}\cup(T_{CG'}\backslash(T_{CG'}\cap T_{CG}))$ (as c=d)

- To prove (2):

c$\in T_{CG}\cup(T_{CG'}\backslash(T_{CG'}\cap T_{CG}))$ means c$\in T_{CG}$ or c$\in T_{CG'}\backslash(T_{CG'}\cap T_{CG})$.

a) If c$\in T_{CG}$, then $\exists$C$\in\boldsymbol{C}$ with *type*(C)=c. Since $\boldsymbol{C}\subseteq\boldsymbol{C}\cup(\boldsymbol{C'}\backslash(\boldsymbol{C'}\cap\boldsymbol{C}))$, this implies: C$\in\boldsymbol{C}\cup(\boldsymbol{C'}\backslash(\boldsymbol{C'}\cap\boldsymbol{C}))$ or C$\in\boldsymbol{C''}$ or *type"*(C)$\in T_{CG''}$. Since *type"*(C)= *type*(C)=c, we obtain c$\in T_{CG''}$.

b) If c$\in T_{CG'}\backslash(T_{CG'}\cap T_{CG})$, then c$\in T_{CG'}$ and $\exists$C'$\in\boldsymbol{C'}$ with *type'*(C')=c.

We have either: C'$\in\boldsymbol{C'}\backslash(\boldsymbol{C'}\cap\boldsymbol{C})$ or C'$\in\boldsymbol{C'}\cap\boldsymbol{C}$

- If C'$\in\boldsymbol{C'}\backslash(\boldsymbol{C'}\cap\boldsymbol{C})$, then since $\boldsymbol{C'}\backslash(\boldsymbol{C'}\cap\boldsymbol{C})\subseteq\boldsymbol{C}\cup(\boldsymbol{C'}\backslash(\boldsymbol{C'}\cap\boldsymbol{C}))$, we obtain C'$\in\boldsymbol{C}\cup(\boldsymbol{C'}\backslash(\boldsymbol{C'}\cap\boldsymbol{C}))$ or C'$\in\boldsymbol{C''}$ and therefore *type"*(C')$\in T_{CG''}$ (by definition of $T_{CG''}$) or type"(C')=type'(C')=c$\in T_{CG''}$.
- If C'$\in\boldsymbol{C'}\cap\boldsymbol{C}$, then by definition of $\boldsymbol{C'}\cap\boldsymbol{C}$, $\exists$C$\in\boldsymbol{C}$ such that *type*(C)=*type'*(C')=c. Since $\boldsymbol{C}\subseteq\boldsymbol{C}\cup(\boldsymbol{C'}\backslash(\boldsymbol{C'}\cap\boldsymbol{C}))$, this implies C$\in\boldsymbol{C}\cup(\boldsymbol{C'}\backslash(\boldsymbol{C'}\cap\boldsymbol{C}))$ or C$\in\boldsymbol{C''}$ and therefore *type"*(C)$\in T_{CG''}$ or *type"*(C)= *type*(C)=c$\in T_{CG''}$.

Similarly, one can demonstrate that: $T_{RG''}=T_{RG}\cup(T_{RG'}\backslash(T_{RG'}\cap T_{RG}))$. □

Property 3. *If a conceptual graph could be simplified then it could be internally joined.*

Proof. "A graph could be simplified" means that there exists at least a relation node which is a duplicate of another relation node. As per definition of relation node duplicate, this means that there exists at least a pair of arguments of the same order in the two duplicate relation nodes, which are duplicate concept nodes (otherwise the 2 relation nodes are identical, not duplicates). Therefore, there exists at least a pair of concept nodes, which are duplicates of each other (but not identical). Thus, the graph could be internally joint by those duplicate concept nodes. □

Theorem 1. *If an ontology **O** has a projection into an ontology **O'**, then for any conceptual graph defined with respect to **O**, there exists a conceptual graph **G'** defined with respect to **O'** and which is a projection of **G**. In addition, we can select **G'** so that it is condensed.*

Proof. The conceptual graph ***G'***=(***C'***, ***R'***, *type', referent', arg'*) could be defined as follows:

(1) ***C'*** is built as the set of all elements C' defined as follows:

- $\forall$C let c = *type*(C). We have: $c \in T_{CO}$
- Since the ontology ***O*** has a projection into ***O'***, $\exists c' \in T_{CO'}$ such that $c' \leq c$. We call c" the common supertype of such c' (since $T_{CO'}$ has a lattice structure, as per assumption in Section 2, such c' must have a unique common supertype c" that also belongs to $T_{CO'}$), i.e. c" is the maximal proper subtype of c, that exists in $T_{CO'}$.
- C' is the concept node in $T_{CO'}$ whose type is c" and whose referent is the referent of C. This is allowed as $I_O \subseteq I_{O'}$.
- C' thus defined is a projection of C: C' = *proj*(C)

  Therefore $\forall$C $\exists$C' such that C' = *proj*(C)

Note that in defining C' as above, we can build an *internally joined* ***C'*** by considering that: if C' has already been created in ***C'*** by a previous operation, then we re-use the existing node in ***C'*** (i.e. we don't need to add a duplicate C' to ***C'***). In addition, we can keep ***C'*** *compact* by considering the following: if C' as defined above has a generic referent and if there is already in ***C'*** a node of the same type but with a non-generic referent, then we re-use the latter (i.e. we don't need to add the former node to ***C'***). Note that in this case, the latter node is a projection of the former node.

(2) ***R'*** is built similarly (without the need to consider referents).

(3) The above definitions of ***C'*** and ***R'*** also implicitly define the functions *type'* and *referent'*.

(4) As per (2) above, we have: ∀R' ∃R such that R' = *proj*(R)

We define the function *arg'* as: ∀i∈**N+** ∀R' *arg'*(i,R') = *proj*(arg(i,R)) with R retrieved from ***R*** as above.

Notes:

- The function *arg'* thus defined may be a surjective function with respect to the domain set (**N+,** ***R'***) and the value set ***C'*** (i.e. every element of ***C'*** may be linked to a relation node in ***R'***) while *arg* may not, i.e. there may be a concept node in ***C*** which is not part of any relation node and two concept nodes may have the same projection in ***G'***.
- Also, for the same reason, R' may have duplicate arcs (while R may not).
- As in the case of building ***C'***, we can build ***R'*** so that it is always *simplified* by not adding duplicates to ***R'*** and by re-using appropriate relation nodes, if they already exist in ***R'***.

We can then easily verify that the conceptual graph ***G'*** thus built is a projection of the conceptual graph ***G***. Furthermore, ***G'*** is *condensed.* □

Theorem 2. *Each canonical formation rule (i.e. external join, internal join, type restriction, referent restriction, copy, simplification and condensation) is a homomorphism with regard to the conceptual graph projection operation, that is if* ***G*** *has a projection into* ***H*** *and* ***G'*** *has* *a projection into **H'** then the external join (or any other canonical formation rule) of **G** and **G'** has a projection into the external join (or any other same canonical formation rule) of **H** and **H'**.*

Proof. We will perform the demonstration for the external join only. The demonstration is similar for other canonical formation rules.

Let **G**" = External Join of **G** and **G'**

H" = External Join of **H** and **H'**

C, **C'**, **C"** = the concept node sets of **G**, **G'** and **G"** respectively

D, **D'**, **D"** = the concept node sets of **H**, **H'** and **H"** respectively

R, **R'**, **R"** = the relation node sets of **G**, **G'** and **G"** respectively

U, **U'**, **U"** = the relation node sets of **H**, **H'** and **H"** respectively

Also, in order not to further overload the notations, we denote *type* as the type functions for all conceptual graphs (i.e. *type* is also *type'*, *type"*, etc. depending on its argument). We use the same convention for the other functions: *referent*, *arg* and *arity*.

We must demonstrate that **G**" has a projection into **H"**, i.e. that the following statements (1), (2) and (3) are true.

(1) We must prove that every concept node in **C"** has a projection in **D"**, or, as per definition of concept node projection:

∀S∈**C"** ∃T∈**D"** *type*(T)≤*type*(S) and (*referent*(S)=*referent*(T) or *referent*(S)=*).

Indeed, by definition of the external join,

S∈**C"** ⇔ S∈**C**∪(**C'**\(**C'**∩**C**))

- If S∈**C** then since **G** has a projection into **H**, there exists a concept node T∈**D** with *type*(T)≤ *type*(S) and (*referent*(S)=*referent*(T) or *referent*(S)=*).

T∈***D*** => T∈***D***∪(***D'***\(***D'***∩***D***)) or T∈***D"*** (as ***H"*** is the external join of ***H*** and ***H'***).

Statement (1) is therefore proven in this case.

- If S∈***C'***\(***C'***∩***C***) then S∈***C'*** and since ***G'*** has a projection into ***H'***, there exists a concept node T'∈***D'*** such that *type*(T')≤ *type*(S) and (*referent*(S)=*referent*(T') or *referent*(S)=*).

  T'∈***D'*** means either T'∈***D'***\(***D'***∩***D***) or T'∈***D'***∩***D***

  - If T'∈***D'***∩***D*** then there exists T"∈***D*** such that T" ≡ T' (by definition of ***D'***∩***D***), i.e. *type*(T")=*type*(T') and *referent*(T")=*referent*(T').

    Since ***D*** ⊆ ***D"*** and *type*(T')≤ *type*(S) and (*referent*(S)=*referent*(T') or *referent*(S)=*), we obtain:

    ∃T"∈***D"*** *type*(T")≤*type*(S) and (*referent*(S)=*referent*(T") or *referent*(S)=*).

    Statement (1) is therefore proven in this case.

  - If T'∈***D'***\(***D'***∩***D***) then T'∈***D"*** (as ***D"*** =***D***∪(***D'***\(***D'***∩***D***))) and we have: ∃T'∈***D"*** such that *type*(T')≤*type*(S) and (*referent*(S)=*referent*(T') or *referent*(S)=*).

    Statement (1) is therefore proven in this case.

(2) Similarly, we can demonstrate that all relation nodes in ***G"*** have a projection into ***H"***.

(3) The last statement to demonstrate is that every relation node in ***R"*** has a projection in ***U"***, i.e. ∀R∈***R"*** ∃U∈***U"*** such that U=*proj*(R)

or (*type*(U) ≤ *type*(R) and ∀i∈**N+** *arg*(i,U)=*proj*(*arg*(i,R))

Indeed, ∀R∈***R",*** since ***G"*** is the external join of ***G*** and ***G'***, and ***H",*** the external join of ***H*** and ***H'***, we have:

R" = ***R***∪(***R'***\(***R'***∩***R***))

and ***U"*** = ***U***∪(***U'***\(***U'***∩***U***))

There are 2 cases:

- If R∈***R*** then since ***G*** has a projection into ***H,*** there exists U∈***U*** (which implies U∈***U''***) such that *type*(U) ≤ *type*(R) and ∀i∈**N+** *arg*(i,U)=*proj*(*arg*(i,R)). Statement (3) is therefore proven in this case.
- If R∈***R'***\(***R'***∩***R***) then R∈***R'*** and since ***G'*** has a projection into ***H',*** there exists U'∈***U'*** such that *type*(U') ≤ *type*(R) and ∀i∈**N+** *arg*(i,U')=*proj*(*arg*(i,R)).

  U'∈***U'*** means either U'∈***U'***\(***U'***∩***U***) or U'∈***U'***∩***U***

  - if U'∈***U'***\(***U'***∩***U***) then U'∈***U***∪(***U'***\(***U'***∩***U***)) or U'∈***U''***. Statement (3) is therefore proven in this case.
  - if U'∈***U'***∩***U*** then ∃U∈***U*** such that U' ≡ U

    or (*type*(U')=*type*(U)

    and *arity*(U')= *arity*(U)

    and ∀i∈**N+** *arg*(i,U')=*arg*(i,U) or *arg*(i,U') ≡*arg*(i,U))

  a) Since ***U*** ⊆ ***U***∪(***U'***\(***U'***∩***U***)) or ***U*** ⊆ ***U''*** , we have U∈***U''***

  b) Since *type*(U')≤*type*(R), it ensues that *type*(U)≤*type*(R).

  c) Furthermore, ∀i∈**N+**

  arg(i,U')=*proj*(*arg*(i,R))

  and (*arg*(i,U')=*arg*(i,U) or *arg*(i,U') ≡*arg*(i,U))

  imply that *arg*(i,U)=*proj*(*arg*(i,R))

a), b) and c) prove Statement (3) in this case. □