Expressing canonical formation rules with graph grammars operations

Stéphane Lapalut *

Projet ACACIA, INRIA Sophia Antipolis, BP 93,
06 902 Sophia Antipolis Cédex, France
Stephane.Lapalut@sophia.inria.fr
phone: 33-9365-7645, fax: 33-9365-7783

Abstract. Since the first Sowa's proposal, two sets of rules provide reasoning capabilities on conceptual graphs. We think it is important to be able to deal with both sets in the same framework, hence to have a clear notion of how these rules interact together. We think also that the strength of the conceptual graph formalism is the possibility to use a graph theoric approach to deal with. A natural way to do that is the use of graph grammars. We propose in this paper an approach that goes deeper in the graph structure of conceptual graph, using a detailled algebraic model that we manipulate in the graph grammars theory framework. We provide a precise description of canonical derivation rules with graph grammars operations. When we discuss the issue to deal with inference rules in the same way.

1 Introduction

Since the first Sowa's proposal, two sets of rule provide reasoning capabilities on conceptual graphs. We think it is important to be able to deal with both sets in the same framework, hence to have a clear notion of how these rules interact together. Recent works attempt to mix the two sets of rules, providing some answers to the questions "are their complementary sets?", "is one set more general than the other?", "is there an overlap between the two sets?". But to our knowledge, a full answer to these questions remains to be worked out. The main differences between the two sets of rules is the kind of graphs they act on. The canonical formation rules are defined upon graphs without "context". Sowa's rules of inference inspired from Peirce's $\beta-$system deal with "contexts". But as stated in [3], this notion of "context" is not well defined in the conceptual graph framework.

The canonical formation rules have been said to be graph grammars rules [4]. From a high level of description, it is true, but it lacks details to use them as graph grammars operations. To precise them, we need a raffined algebraic description of conceptual graph, as proposed in [5], but in an unified framework, viewing also *support* as graphs related to *conceptual graph* graphs. With such a model, we are able to use graph grammars formalisms to precisely describe the canonical derivation rules. We hope to be able to formalize the inference rule in the same way, hence to give an algebraic description of the "context" notion. It will provide a way to related both sets on the common ground of graph grammars and then the way to use the two sets of rules together.

This paper is organized as follow. In the second section, we recall the two sets of rules, the canonical formation rules and the inference rules, analysing their fundations and showing their specifities. Then we propose in the third section a more raffined algebraic definition of conceptual graphs, which enable us to explicitly use graph grammars. In the fourth section, we define canonical formation rules based on our model, using graph grammars formalism. We relate in the fifth section two works and proposals, which try to combine semantic networks with the graph grammars formalism.Then we conclude in the sixth section with the usefullness of our approach in representing reasoning on conceptual graphs.

* Thanks to the anonymous referees for their usefull remarks and having pointing us to the two references [1, 2], and thanks to G. Ellis in providing a copy of his article [1].

2 The Two Sets of Rules

Since the first statements of canonical derivation rules and inference rules, we recall in this section the amendements that have been made, while the separation between the two sets remains. In the first set, the canonical derivation rules are a kind of *generic* graph-grammars that we will precise in the next section. The second set, the inference rules, is borrowed from the Peirce's $\beta-$system. The conceptual graph framework is different from the Peirce's existential graph framework, as we precise it, and the problem of "context" definition remains.

2.1 Canonical Formation Rules

Historically, canonical formation rules set have been the first and only set of rules to transform conceptual graphs. Canonical formation rules are used to derive canonical graphs from an initial set of canonical graphs. These graphs are simple graphs as defined in [5]. They act as a kind of "selectional constraints" [6] to preserve the canonicity upon derivation. One of the firstly proposed set of canonical derivation rules is {*copy*, *restrict*, "*join on a concept*", "*join on a star graph*"}, defined as follow (from [6]) :

- **Copy:** An exact copy of a canonical graph u is canonical;
- **Restrict:** Let a be a concept in a canonical graph u, and let t be a type label where $t \leq type(a)$, then the graph obtained by replacing a with a concept of type t is canonical[2] ;
- **Join on a concept:** Let a be a concept in a canonical graph u, and b a concept in a canonical graph v. If a and b are joignable[3], then the graph derived by the following steps is canonical: (1) delete b from v, (2) then attach to a all arcs of conceptual relation that had been attached to b;
- **Join on a star:** Let r be a conceptual relation in a canonical graph u, and s a conceptual relation in a canonical graph v. If the star graphs r and s are joignable[4] then the graph derived by the following steps is canonical: (1) delete s and its arcs from v, (2) for each i, join the concept attached to arc i of r to the concept that had been attached to arc i of s.

These earlier rules do not consider referent. They can be seen as a description of a kind of *node label controled* graph grammars rules [8]. In the "join on a star graph" rule, *gluing points* and *embedding conditions* (see 4.1) are specified. The "join on a star graph" rule itself is not a basic graph grammars rule, but an association of rules (parallel applications of rules). In the 1984 Sowa's book, this set of canonical rule have been refined and recently, a new proposal for a six rules set organized in three subsets have been made. We think this set is a definitive one and we will use it as a basis for expressing conceptual graph operations with graph grammars. In his forthcoming book, Sowa will state it as this[5]:

- **Equivalence rules:** The two rules *copy* and *simplify* create a graph that is logically equivalent to the original.

- **Specialization rules:** The two rules *restrict* and *join* create a graph v that is more specialized than the original v (in the sens of the implication: v implies u).

  Restrict modified a single node to make it more specialized. It either replaces the type by a subtype, or it replaces the referent by a more specific referent.

  Join combines two nodes to make a single node.

[2] In this earlier statement, the problem of referents seems to be left out, especially the case of compatible concept (see [7, 5]).
[3] In [6], two concepts a and b are said to be joignable if $type(a) = type(b)$.
[4] Two star graphs with conceptual relations r and s are said to be joignable if r and s have the same number of arcs, $type(r) = type(s)$, and for each i, the concept attached to arc i of r is joinable to the concept attached to arc i of s [6].
[5] Borrowed from a message to the cg-mailist, also in [9].

– **Generalization rules:** The two rules *unrestrict* and *detach* are are the inverse of restrict and join. They create a new graph v that is logically implied by the orignal graph u.

These rules give a clearer point of view upon the early set. We can define a kind of algebra with the three subsets (“any combination of equivalence rules is an equivalence rule”, “any combination of specialization rules and equivalence rules is another specialization rule”, “any combination of generalization rules and equivalence rules is a generalization rule”). But our interest is to precise these rules from a graph grammars point of view, which is squetched in the previous definitions. The specifity of this set of rules is the concept restriction, which is not basically a graph grammar operation. To handle type restriction in the graph grammars framework, we need a precise algebraic description of conceptual graphs, which we give in the section 3. Using this algebraic model, we propose graph grammars expressions to handle canonical derivation rules in the section 4.

2.2 Inference Rules

The logical fondation of conceptual graph are the Peirce's existential graphs [10]. In his conceptual graph framework, Sowa reuse the four rules of inference defined by Peirce in his $\beta-$system. These rules primary act on existential graphs containing areas closed by a *cuts*, “a self-returning finely drawn line”[6] that defined together with the enclosed area an *enclosure*. Without the cut, existential graphs are a kind of existential conjonctive positive logic notation, drawn on a *sheet of assertion*. The cut provide a kind of negation, “cutting” the enclosed graphs from the sheet of assertion, hence negating them. Under the closed world assumption, the area cutted from the area of a cut from the sheet of assertion, which is called a *double cut*, is the sheet of assertion itself. The existential graph notation with the cut provides the minimal features to enabling logical calculus (such as a primitive set of connectors, $\{\neg, \wedge\}$ in Peano-Russell notations).

From a graph structure point of view, existential graphs are different from conceptual graphs. The latter support an algebraic definition, as the one given in [5], but the former is rather a kind of graphical notation than graphs in the mathematical sense. Peirce's *lines of identity* become *coreferent links* in conceptual graphs, relating the occurences of the same explicite individual (a variable or an individual referent). To introduce the cut notion in the conceptual graph framework and reuse the Peirce's rules of inference, Sowa expect a notion of context. He understands the Peirce's cut as a “partitioning of a negative context from the surrounding sheet of assertion”. But he defines a “negative context” without a previous account of a context definition: NEG $\longrightarrow$ $\boxed{\text{PROPOSITION: u}}$ (Definition 4.2.2 in [4]). We can rather see this definition as the introduction of a kind of modal (unary) relation, leading to meta-level distinctions for concept type (such as PROPOSITION) and relation types (such as NEG). Sowa remarks at the end of his 1984 book that contexts, as intended in his 4.2.2 definition, lead to modalities. But as Nazarenko remarks it [3], there is two possible interpretations of context notion used in [4]. The syntactic interpretation of context, inspired by the Peirce's cut, is melt with another one related to the linguistic notion of point of view. Later works try to precise this notion of context [12, 13, 9], which are major contribution to the formalisation of the context notion. The χ expansion defined by Esch views context as relating concept definition to concept type occurence [12]. It leads to a kind of hierarchical graphs, close to Wermelinger proposal [14], but Esch does not provide a meta-level to deal with modalities or negation. Esch define black box (resp. white box) view of context as a special kind of concept (resp. referent). Context and concept are viewed as abstraction dual, which does not fit the previous intuition of context to represent the Peirce's cut. The negation is only handled in the coding of graph hierarchy, without account of its formal counterpart. We can the need of a meta-level description, involving higher order types, with an example provided in [15].

[6] Peirce cited in [11], p. 35.

$\boxed{\text{SITUATION : } \boxed{\text{CAT}} \to (\text{ON}) \to \boxed{\text{MAT}}} \equiv$

$\boxed{\text{SITUATION}} \to (\text{DESC}) \to \boxed{\text{PROPOSITION : } \boxed{\text{CAT}} \to (\text{ON}) \to \boxed{\text{MAT}}} \equiv$

$\boxed{\text{SITUATION}} \to (\text{DESC}) \to \boxed{\text{PROPOSITION}} \to (\text{STMT}) \to \boxed{\text{GRAPH : } \boxed{\text{CAT}} \to (\text{ON}) \to \boxed{\text{MAT}}}$

Esch shows us three successive levels of details, describing the context of the graph $\boxed{\text{CAT}} \to (\text{ON}) \to \boxed{\text{MAT}}$ always appears in a referent field, "typed" by a higher order type, such as SITUATION, PROPOSITION or GRAPH. More, if we try to extract this graph from its context, we have no means to related it to its context. We can thing of the use of a coreferent link, writing for the first graph $\boxed{\text{SITUATION}} \cdots \boxed{\text{CAT}} \to (\text{ON}) \to \boxed{\text{MAT}}$, but we cannot decide which concept this link will lies to. It make no sens to link SITUATION with the concepts CAT or MAP, nor with the relation ON. The same problem arises when doing the same with GRAPH. This fact illustrates the need for a disctinction between the domain concepts and the meta-concept, leading to higher order types distinctions. We will address at more this issue at the end of the fifth section.

To our knowldge, no other work deals with the handling of context and negation in the conceptual graph famework. We think these latter are closely interrelated in the conceptual graph theory, due to the use of Peirce inference rule that deal with the cut. At that time, there is no clear formal account of both negation and context in the CG theory. In relating Peirce like inference rules to graph grammars operations, we hope to provide a way to precise the cut notion with an algebraic definition.

2.3 Merging the two sets

From our point of view, we think that relating both sets of rules together is a key issue of the conceptual graph theory. We hyphotesize that this goal could be reach if we provide a formal definition of the context and negation, hence the conceptual graphs counterpart of the Peirce's cut. We have choose the algebraic approach to allow an easy manipulation of conceptual graphs and to exploit their graph structure properties. The next section details an algebraic model of conceptual graphs, extending and precising previous one, as in [5]. The obtained model is more suitable to deal with conceptual graph structure properties in the graph grammars framework. We provide in the fourth section the way to express canonical rules, especially the concept restriction operation, with the help of our model.

3 Algebraic Expression of Conceptual Graphs

In [5], Chein and Mugnier have proposed the interesting definitions relating conceptual graphs to a *support*, which encloses all we need to write simple conceptual graphs and to ensure the rightness of its syntax. Then they define conceptual graphs, but only with a functional relation between them and the support (for a concept node c, his label is defined as $lab(c) = (type(c), ref(c))$, with $type(c)$ the type of c and $ref(c)$ the referent of c). We propose a more precise definition, expliciting structuraly the link between graphs and their support, to allow type restriction handling by graph grammars operations, making them more suitable for the use of graph grammars rules.

The introduction of the support in the formalism is very important. It enable us to explictly refer to concept types, relation types and relation signatures through star graphs. But the definitions of Chein and Mugnier maintain the type/referent duality of the concept node labelling. The couples (type,referent) is more than a label. It complexifies the concept node label handling, when performing a concept type restriction through a graph grammars operation, for instance. Formally, it induces semantic considerations (changing a character string in the concept node label) in place of structural one (the subtype relation in the concept type hierarchy, which is a kind of graph). Thus, we propose in the following subsection a refinement of the conceptual graph algebraic definition to formalize the typing relation as a structural one. This is a cornerstone of our approach, which enabling a wide use of graph grammars to formalize conceptual graph rules.

3.1 Relating the support to the conceptual graphs

In order to relate the conceptual graphs with the relevant supports, we define a conceptual base B_{CG} as a tuple, $B_{CG} = (S_c, S_r, B_c, B_r, M, conf, G)$:

Definition 1. A conceptual graph base B_{CG} is a tuple, $B_{CG} = (S_c, S_r, B_c, B_r, M, conf, G)$, where:

1. S_c is a graph that supportes the concept types structure, defined as $S_c = (N_{T_c}, A_{T_c}, m_{T_c}, M_{T_c})$ where
   (a) N_{T_c} is a set of labelled nodes, called *concept type nodes*,
   (b) A_{T_c} is a set of directed edges between elements in N_{T_c} labelled as *subtype*,
   (c) M_{T_c} is a set of labels, called *concept types*,
   (d) m_{T_c} is a mapping between concept type nodes and concept types :
   $m_{T_c} : N_{T_c} \mapsto M_{T_c}$, hence the labelling of concept nodes by concept node types;
2. S_r is a graph that supportes the relation type structure, defined as
   $S_r = (N_{T_r}, A_{T_r}, m_{T_r}, M_{T_r})$ where
   (a) N_{T_r} is a set of labelled nodes, called *relation type nodes*,
   (b) A_{T_r} is a set of directed edges between elements in N_{T_r} labelled as *subtype*[7],
   (c) m_{T_r} is a set of labels, called *relation types*,
   (d) M_{T_r} is a mapping between relation type nodes and relation types :
   $m_{T_r} : N_{T_r} \mapsto M_{T_r}$;
3. B_r is a set of *star graphs*, as defined by Chein and Mugnier [5];
4. B_c is the set of concept type definition;
5. M is a set of node labels, called *markers*;
6. $conf$ is a predicate that checks the use of markers [mc92];
7. G is a finite set of graph G_i, called conceptual graph, defined as a tuple $G = (R, C, s, t, u_r, u_c, m)$, where
   (a) R is a set of *relation nodes*,
   (b) C is a set of *concept nodes*,
   (c) s is the set of edges from relation nodes in R to concept nodes in C,
   (d) t is the set of edges from concept nodes in C to relation nodes in R,
   (e) u_r is the set of edges labelled "type" from relation nodes in R to relation type nodes in N_{T_c}, which attach a relation type to each relation node,
   (f) u_c is the set of edges from concept nodes in C to concept type nodes in N_{T_c},
   (g) m is a mapping from the set of makers M and the set of concept nodes C, then the labelling of concept node.

3.2 Extension of the model

The above model only describe "simple graphs", as defined by Chein and Mugnier [5]. To handle the Peirce's cut notion, it is not sufficient. As we have previously remarked it, we need some meta-types to describe "context", then the Peirce's notion of cut. Introduction of such meta-types will change the basic conceptual graph formalism. This work is the matter to a full article. We will present the basic ideas at the end of the next section.

[7] This feature is not yet used in the conceptual graph theory (empty set), but we mention it as it can be used in some future, as described in [14].

4 Graph Grammars to Express Canonical Derivation Rules

In [4], the semantics of the canonical derivation rules are given. They ensure the well-formedness of derived graph. They are also a generic statement of graph grammars operations. We propose in this section to explicite the underlying graph grammars transformations of these rules, using the previous definitional framework. We first introduce the graph grammars framework and then we discuss the graph grammars expressions of the rules.

4.1 Graph Grammars Framework

We give a taste of the graph grammars field, with the (very) basic knowledges to understand the use graph grammars rules to formalize conceptual graphs rules.

Principles : graph productions and direct derivations. A graph grammars is a set of productions that generates a language of terminal graphs and produces non-terminal graphs as intermediate results. It can be viewed as a generalization of string grammars that handles complex transformation on complex data structures. The field of graph grammars as grown steady from the early 70's. It starded mainly with theorical grounding works [8, 16], to reach recently available integrated tools which support graph rewriting programming [17]. We refer the reader to [18, 19, 20] to get more materials about the state of the art of the graph grammars field.

The basic entity of a graph grammars is a *graph production*. Under certain conditions, namely the *gluing conditions*, a graph production can be used to rewrite some parts of a graph. This process is formalized in the algebraic categorial theory by a *pushout diagram*. Precisely, Ehrig [8] define graph productions and graph grammars as follow, to transform a graph G into H using a production p:

1. A graph production is a pair of graph morphisms $p = (B_1 \leftarrow K \rightarrow B_2)$, where the graphs B_1, B_2 and K are called *left side*, *right side* and *interface graph* of p respectively.
2. Given a production $p = (B_1 \leftarrow K \rightarrow B_2)$ and a graph D, called *context graph*, a graph morphism $K \rightarrow D$, a *direct derivation* consist of the following pushouts PO_1 and PO_2 :

$$\begin{tikzcd} B_1 \arrow[d] \arrow[dr, phantom, "PO_1"] & K \arrow[l] \arrow[r] \arrow[d] \arrow[dr, phantom, "PO_2"] & B_2 \arrow[d] \\ G & D \arrow[l] \arrow[r] & H \end{tikzcd}$$

   We write $G \overset{p}{\Rightarrow} H$ and $B_1 \rightarrow G$ (resp. $B_2 \rightarrow H$) is said to be the occurence of p in G (resp. H). $G \overset{p}{\Rightarrow} H$ is called a direct derivation by p based on $B_1 \rightarrow G$.
3. A derivation $G \overset{*}{\Rightarrow} H$ means we obtain H by a *sequence* of direct derivations from G.

We interpret the interface graph K of a production p as the set of gluing items which are mapped by $(K \leftarrow B_i)$ to B_i for $i \in 1, 2$. In the direct derivation $G \overset{p}{\Rightarrow} H$ given by the pushout PO_1 and PO_2, the production p and the context graph D are defining G (resp. H) as gluing of B_1 (resp. B_2) and D along K. In other words, we obtain H from G by deleting B_1 and adding B_2 (in both cases without gluing items). The gluing items remain in the context graph D, which is the result deleting only B_1 from G.

As a picture of the whole process, we can think of G as an egg, D as egg white with the membrane, B_1 as yolk and K as the membrane between the yolk and egg white. If you want an egg with two yolks, you will substitute B_1 (one yolk) with B_2 (two yolks) to build a new egg from G, say H, which as two yolks.

Definition 2. conceptual graph grammar.

We define a *conceptual graph grammar* as an uplet $CGG = (CG, CG_T, Pd, St)$ with CG a set of conceptual graphs, CG_T a set of terminal canonical conceptual graphs included in CG, a finite set of graph productions Pd, and a set of finite canonical conceptual graphs St. In some particular graph language, we will have $CG = CG_T$.

In the next section, we will precisely describe the Pd set for canonical formation rules using the previously defined algebraic model of conceptual graph.

Definition 3. Conceptual graph language.

The set of all terminal conceptual graphs derivable from graphs in St by graph productions in Pd is called the conceptual graph language, noted $L(CGG)$ with $L(CGG) = \{H \in CG_T | G \in St, G \stackrel{*}{\Rightarrow} H\}$.

The previous definitions provide us a graph grammars framework to deal with conceptual graphs. The point of interest of this papers are the graph productions that we use to express conceptual graph canonical formation rules. Before to precise this point, we need to overview the main expression of the graph production rules and their general features.

Formalization of graph grammars rules. Practically, graph productions can be used in different ways. Two main points of view have been raised: dealing with *nodes* or dealing with *edges*. The former is said to be Node Label Control (NLC) graph grammars, those rules transform nodes of a given graph, and the second is called Edge Label Control graph grammars, which included the hyperedge replacement technics [21]. The idea is to substitute a node or an edge, identified by its label, with a graph, as specified in a graph production. But we need additional prescriptions to deals with embedding of the "patched" graph in the existing graphs (with previous notations, the way to connect B_2 into D, through gluing points). These prescriptions are called "embedding rules". A very compact and precise notation of theses embedding rules have been provided by Nagl in [20], which are called *embedding expressions*. So, we can sum up the basic application of graph productions (together with embedding rules) in three steps[8] (with Del the prescriptions to delete nodes and edges, Dem the prescriptions to determine embedding areas and Gen the descriptions to achieve nodes and edges generation):

1. Del and Det identify possible starting area to apply graph productions (match the interface graph K, hence test the gluing conditions),
2. Del identifies the removing of node and edge according to a choosen production application,
3. Det and Gen allow the patching of the production result B_2 into the rest graph D,
4. Dem links the patched graph B_2 into the host graph D, which achieves the rule application.

This quick overview does not take into account powerful extensions, such as conditional embedding and replacement, but it allow us to deal with the graph grammars expressions of conceptual graph canonical formation rules in a fluent manner. We present such an application in the next section.

4.2 Graph Grammars Productions

From all technics used to describe graph productions, we can excerpt a small set of primitive operations. This set has been given in [22] and includes six elementary types of graph productions, namely the *addition*, the *deletion* and the *renaming* of a *node* or an *edge*[9]. We can easely figure out the expression

[8] Inspired from a message of Robert Mueller to the cg mailist.

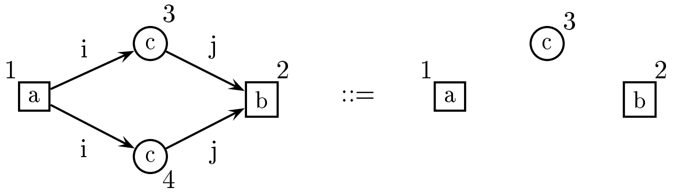
[9] Notice that the application of a graph grammars rule means the parallel application of some of the elementary operations. In the same way, one of the powerful use of graph grammars is the parallel applications of graph grammars rules. We have not yet address this topic, even if it is entails in the handling of some rule, such as the duplication of a graph (copy rule).

of canonical operation such as copy, simply and join with these primitive productions, but the concept restriction is not an immediate consequence of them. To expresse concept restriction, we need to use the specificity of our model, the explicite links between a conceptual graphs and its support, together with a graph embedding operation and deletion/addition operations.

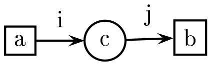
Copy. This rule does nothing but give the same graph.Both sets of gluing points and embedding expressions are empty: this production is always applicable to any graphs. We will just write the copie of a graph G:

$$G ::= G \; G$$

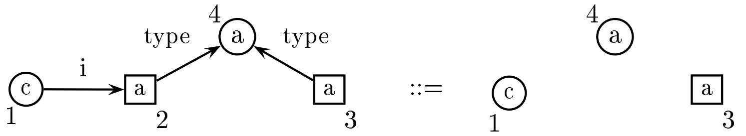
Symplify. Given redondante relational nodes, the production will allow only one of them. Given the three gluing points $\{1, 2, 3\}$, the rule delete the node 4 and all edges in the identified subgraph (left hand side of the rule). Then the embedding conditions draw the two remaining edges. This rule will be stated as :



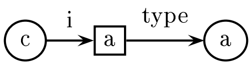
together with the two embedding rules $r_i = (1; R_i(1))$ and $l_j = (L_j(2); 2)$.

The result will be the subgraph  .

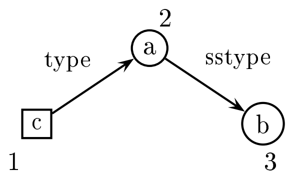
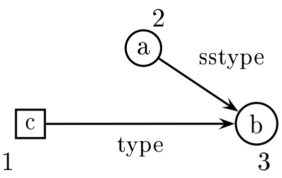
Join. This rule takes a connexe or a non-connexe graph as left hand side, and do a join (identifies gluing nodes, deletes one node of the two identique nodes and connect the edges to the remaining node). We give the rule for an elementary join, say the join of two concept nodes. For a maximal join, we applied in parallel several join rules. Gluing points are $\{1, 3, 4\}$ and we proceed the same way in deleting all nodes and connections but gluing poibts. Then embedding rules build needed edges :



with the embedding rules $r_{type} = (3; R_{type}(3))$ and $l_i = (L_i(3); 3)$.

which lead to the result  .

Concept restriction. For this operation, we distinguish between referent decreasing and subtyping. The former is just a label rewriting, according to our model. The latter is an embedding transformation, based of the node identity rule for the subtyped concept node :

from the subgraph  we want 

For that, we use the node identity rule :

$$\underset{1}{\boxed{\text{c}}} \qquad ::= \qquad \underset{1}{\boxed{\text{c}}}$$

together with the embedding transformation that proceed to *type* link change:

$$l_{type} = (\ 1;\ L_{sstype} L_{type}(1)\ )$$

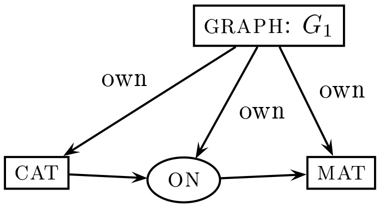
This embedding rule can be read "link the node 1 with a left edge labelled *type* with the node determined by the walk from node 1 along a left edge labelled *type* then along a left edge labelled *sstype*." The useless edge to be removed, labelled *type*, the edge from node 1 to node 2, will be specified by the expression $(\ 1\ ;\ L_{type}(1)\ )$.

Unrestrict and Detach. These rules will be inverse of the join and restrict rules.

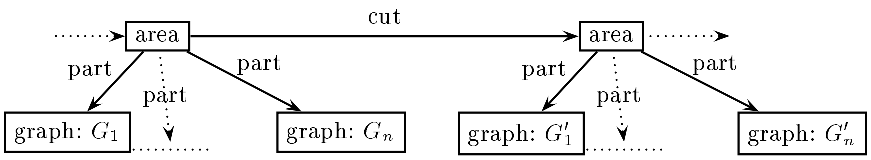
The main point that enable us to effectively specify canonical formation rule with graph production, especially the specific rule of concept subtyping, is the use of these embedding transformations on conceptual graphs describe with our model.

4.3 Inference rules

Canonical formation rule have a natural expression in the framework of graph grammars, using our algebraic model of conceptual graphs. As we have shown it, manipulating inference rule within the conceptual graph formalism requires to consider higher order types, hence to extend the model of simple graphs. The main idea is to grasp the Peirce's cut notion. We propose to name graphs and relate them to singles nodes. If the graph G_1 is the graphe [CAT] → (ON) → [MAT] , we will write :



When dealing with inference rule, we will able to manipulate entire graph, especillay through the area they appear to. We propose the following representation :



The formal account of such descriptions and the graph grammars expressions of inference rules are in preparation. The main problem of such an extension is how it does influence the simple graph kernel.

5 Related Work

Despite the genese of conceptual graphs and the claim that "canonical formation rules are a kind of context free graph grammars", the graph grammars field and the conceptual graphs field have grown independently from each other. Only recent and few works have drawn attention to the link between the both. One of this work have been conducted by Ellis and Willems [1], from the conceptual graph community, and the other by Ehrig [2], one of the father of the graph grammars community. To our knowledge, no one have published a paper dealing with a full connection between the two fields, even if these two works address relevant issues.

The work from Ehrig addresses the application of graph grammars to semantic networks. It draw attention to the Sowa's formalism together with the KL-ONE formalism. He remarks the "extra order" in the Sowa's formalism, for that he preferes to deals with the KL-ONE structure. Anyway, he shows that graph grammars rules can be use to preserve well-formedness of graph. Considering conceptual graphs, the idea is simply to express canonical formation rules, which have been proved to derive well-formed graphs, with graph grammars rules. We have provided in the present paper an algebraic model of conceptual graphs together with such expressions of canonical formation rules with graph grammars rules. The main advantage Ehrig find in using graph grammars rules to act on semantic network is the preservation of this well-formedness and the automated parsing it allows.

In their working note [1], Ellis and Willems focus their attention to the conceptual graphs formalism. As they remark it, the use of graph grammars rules induces a precise description of the targeted formalism. Their main interest is the comparison of different formalisms, applied in their paper to the SNePS formalism and the conceptual graph formalism. This way, they also give graph grammars rules to express the four Sowa's canonical formation rules. But it seems they limite them to the morphism aspect of graph grammars rules, using only graph productions, without the powerful embedding expressions. They also differ in the way they handle referent of concept node. They use an extra node to support the marker, which we have represented as the label of the concept node. We think our way is a more natural one, since we do not put interest in the complex referent problem. They also propose a representation of context, which also use an extra node, leading to a complicated description. We think they miss the point of the context fundation, say the Peirce's notion of cut, for which we propose a description. They do not notice the need of meta-type to handle the "context" description, even if their representation uses such types. And last, they do not give an algebraic model for conceptual graphs, as we propose in this paper.

Then, precising and extending these first works, we have propose a complete approach to express canonical formation rules with graph grammars rules. We have put basis to handle the representation of inference rules, retrieving the original and central notion of the Peirce's cut, which enable reasoning on conceptual graphs.

6 Conclusion

In this article, we reach our main point, the formalization of conceptual graph canonical derivation rules using graph grammars. We provide the needed algebraic model for conceptual graphs to allow the use of graph productions on them. Especially, we explicitly relate conceptual graphs to their support. This feature is a key point to express the concept restriction with structural expression, namely embedding expressions. Other canonical rules have a natural graph grammars expressions.

The next step of our work is the expression of inference rule. We have shown that we need some meta-types to deal with the cut notion. We have put the first basis to express these rules with graph grammars rules. The influence of the introduction of meta-type in the simple graphs model remains to be established.

Aside of the a la prolog implementation of conceptual graph processors, we think that the graph grammars approach is a practible and powerful one to deal with graph reasoning. In this paper, we focuse on the canonical derivation rules, as they are the primitives of conceptual graph operations, such as the graph projection [23]. Given our algebraic model to express conceptual graphs, we have a bridge between the conceptual graph framework and the graph-grammars field, hence a way to refine and extended the conceptual graph theory.

Graph grammars principles, say the pushout vision of the rule applications, stand as a powerfull one to handle the description of conceptual graph manipulations. As an example, a recent work by Salvat [24] that deal with "if-then" rule use a kind of these diagrams to describe the way they act. But it lacks the integration of the type hierachies together with the conceptual graphs, then the type restriction, especially when performing a projection, cannot be viewed as a morphism[10]. We think his "schema of projection" [24], could be restated using pushout diagrams, if we consider conceptual graphs describe by our model. This work is not the subject of this article, but it remain an interesting issue to carry on.
Our approach is currently being implemented within the PROGRES framework.

References

1. Gerard Ellis and Mark Willems. Analysing semantic networks using graph grammars. In *Working Notes for the AAAI Spring Symposium: Propositional Knowledge Representation*, Stanford University, March 25-27 1992.
2. H. Erhig. Introduction to graph grammars with applications to semantic networks. *Computer Mathematical Applications*, 23 (6-9):557–572, 1992.
3. G. Sabah. Formalisation et extension des graphes conceptuels. In *Actes des 4ème Journées nationales du PRC-GRD Intelligence Artificielle*, pages 57–119, Marseille, October 1992. TEKNEA.
4. J.F. Sowa. *Conceptual Structures*, volume Information processing in mind and machine of *The systems programming series*. Addison-Wesley, 1984.
5. M. Chein and M.L. Mugnier. Conceptual graphs: Fundamental notions. *Revue d'intelligence artificielle*, 6(4):365–406, 1992.
6. J. Sowa. Definitional mechanisms for conceptual graphs. In V. Claus, H. Erhig, and G. Rozenberg, editors, *Graph-Grammars and Their Application to Computer Science and Biology*, number 73 in Lecture Notes in Computer Science, pages 426–439, Bad Honnef, October 1979. Springer-Verlag.
7. B.C. Ghosh and V. Wuwongse. Inference systems for conceptual graph programs. In W.M. Tepfenhart, J.P. Dick, and J.F. Sowa, editors, *Conceptual Structures: Current Practices, 2nd International Conference on Conceptual Structures, ICCS'94, College Park, Maryland, USA*, number 835 in Lecture Notes in Computer Science, pages 214–229. Springer-Verlag, 1994.
8. H. Ehrig. Introduction to the algebraic theory of graph grammars (a survey). In V. Klaus, H. Ehrig, and G. Rozenberg, editors, *Graph-Grammars and Their Application to Computer Science and Biology*, number 73 in Lecture Notes in Computer Science, pages 1–69, Bad Honnef,, October 1979. Springer-Verlag.
9. J. Esch and R. Levinson. An implementation model for contexts and negation in conceptual graphs. Technical Report UCSC-CRL-95-22, Departement of computer and information sciences, University of California, Santa Cruz, CA 95064, May 1995.
10. J.F. Sowa. Relating diagrams to logic. In G.W. Mineau, B. Moulin, and J.F. Sowa, editors, *Conceptual Graphs for Knowledge Representation*, number 699 in Lecture Notes in Artificial Intelligence, pages 1–35, Quebec City, Canada, August 1993. Springer-Verlag.

[10] We also notice another problem in this approach. The lack of the pushout interpretation of the rule, precisely the absent notion of interface graph, leads to a redondant "gluing" of subgraph when performing the rule applications. This problem have been also notice by Salvat, as their rules may produce redonsant graphs.

11. D.D. Roberts. *The existential graphs of Charles S. Peirce.* Approaches to semiotics. Mouton, The Hague, Paris, university of Waterloo, 1973.
12. J. Esch. Contexts and concepts, abstraction duals. In W.M. Tepfenhart, J.P. Dick, and J.F. Sowa, editors, *Conceptual Structures: Current Practices, 2nd International Conference on Conceptual Structures, ICCS'94, College Park, Maryland, USA*, number 835 in Lecture Notes in Computer Science, pages 175–184. Springer-Verlag, 1994.
13. J. Esch. Contexts, canons and coreferent types. In W.M. Tepfenhart, J.P. Dick, and J.F. Sowa, editors, *Conceptual Structures: Current Practices, 2nd International Conference on Conceptual Structures, ICCS'94, College Park, Maryland, USA*, number 835 in Lecture Notes in Computer Science, pages 185–195. Springer-Verlag, 1994.
14. M. Wermelinger and J.G. Lopes. Basic conceptual structures theory. In W.M. Tepfenhart, J.P. Dick, and J.F. Sowa, editors, *Conceptual Structures: Current Practices, 2nd International Conference on Conceptual Structures, ICCS'94, College Park, Maryland, USA*, number 835 in Lecture Notes in Computer Science, pages 144–159. Springer-Verlag, 1994.
15. J.W. Esch. Contexts as white box concepts. In G.W. Mineau, B. Moulin, and J.F. Sowa, editors, *Conceptual Graphs for Knowledge Representation*, volume supplementary proceedings, August 1993.
16. M. Nagl. A tutorial and bibliographical survey on graph grammars. In V. Klaus, H. Ehrig, and G. Rozenberg, editors, *Graph-Grammars and Their Application to Computer Science and Biology*, number 73 in Lecture Notes in Computer Science, pages 70–126, Bad Honnef, October 1979. Springer-Verlag.
17. A. Schürr. PROGRES: A VHL language based on graph grammars. In Ehrig, Kreowsky, and Rozenberg, editors, *Graph Grammars and Their Application to Computer Science, 4th International Workshop, Bremen, Germany*, number 532 in Lecture Notes in Computer Science. Springer-Verlag, 1991.
18. H. Ehrig. Tutorial introduction to the algebraic approach of graph-grammars. In H. Ehrig, M. Nagl, G. Rozenberg, and A. Rosenfeld, editors, *Graph Grammars and their application to computer science, 3rd international workshop*, number 291 in Lecture Notes in Computer Sciences, pages 3–14. Springer-Verlag, 1987.
19. A. Habel and H.-J. Kreowski. May we introduce you: hyperedge replacement. In H. Ehrig, M. Nagl, G. Rozenberg, and A. Rosenfeld, editors, *Graph-Grammars and Their Application to Computer Science, 3rd International Workshop*, number 291 in Lecture Notes in Computer Science, pages 15–26. Springe-Verlag, 1987.
20. M. Nagl. Set theoric approaches to graph-grammars. In H. Ehrig, M. Nagl, G. Rozenberg, and A. Rosenfeld, editors, *Graph-Grammars and Their Application to Computer Science, 3rd International Workshop*, number 291 in Lecture Notes in Computer Science, pages 41–54. Springe-Verlag, 1987.
21. A. Habel. *Hyperedge Replacement: Grammars and Languages.* Number 643 in Lecture Notes in Computer Science. Springer-Verlag, 1992.
22. R. Freund and B. Haberstroh. Attributed elementary programmed graph grammars. In G. Schmidt and R. Berghammer, editors, *Graph-Theoric Concepts in Computer Science, 17th International Workshop, WG'91, Fishbachau, Germany*, number 643 in Lecture Notes in Computer Science, pages 75–84. Springer-Verlag, 1991.
23. M. Chein and M.L. Mugnier. Conceptual graphs are also graphs. Technical Report RRI-Chein-95, LIRMM, $ftp : //ftp.lirmm.fr/pub/LIRMM/papers/1995/RRI - Chein - 95.ps.gz$, 1995.
24. E. Salvat. Sound and complete forward and backward chainings of graph rules. In *ICCS'96 (à paraître)*, 1996.